

CS 398: Application Development

Week 03 Video: Analysis & Design 1

Phase 3: Analysis & Design

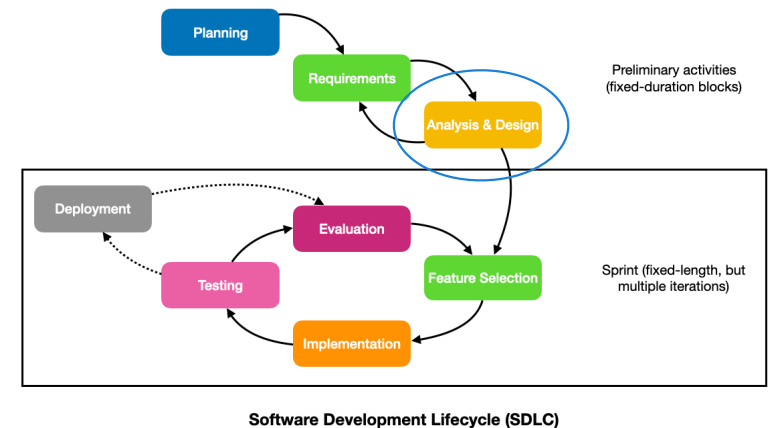
At this point, we have gathered Requirements, and we have a Product Backlog of potential features to implement.

Our features have been gathered from a number of sources: brainstorming with our team, directly provided by the customer, or extracted from our interview data with users.

However, none of our requirements have been vetted. We don't know if they are:

- Technically feasible
- Logically consistent
- Meaningful or useful

We would like to be in a position to make decisions on what we're including, but we need to address these concerns first.



Goals of Analysis & Design

1. We want to examine our high-level technical requirements and make sure that we've addressed the **business needs, user goals and technical requirements** prior to implementation.
2. We want to determine the **technical viability** of our functional requirements, and any **technical constraints** that they will place on our design.
3. Finally, we want to determine the **structure and composition** of our system and document this for the implementation stage.
 - This typically includes generating diagrams, models etc. of our system.

Technical Assessment

Technical Impact

When we create functional requirements, we try and focus on determining our user's needs without any consideration for how we will implement them.

However, at some point, we need to determine the **technical impact** of each requirement:

- Will we be able to implement this feature? Is it even possible?
- Is this feature risky; does it require significant novel research, or is it complex to meet?
- How would implementing this feature impact our system?


Technical Constraints

Pay particular attention to constraints or limitations that a feature introduces.

What other technologies do we require for this to be feasible? (e.g. frameworks, libraries, toolkits).

Does this put a constraint on our overall design? e.g. are we forced to use a particular OS, or deployment target?

- e.g. a requirement to deliver securely store data will have a ripple effect on the system, and affect every other component that works with that data.
- e.g. a feature that requires a specific version of a library will impact other features that use that library.
- e.g. supporting Windows as a deployment target may introduce technical challenges to deploying on macOS or Linux.



Technical decisions
can broadly affect your
design.

What to do with this data?

Each functional requirement from the requirements phase should be examined.

Your goal is to add technical detail to the existing requirements that you can use in Planning.

- Is this requirement risky? Add a statement to that effect and attempt to clarify the degree of risk!
- Does implementing one feature mean that you need to implement some other dependent functionality? Document that in the requirement as well!



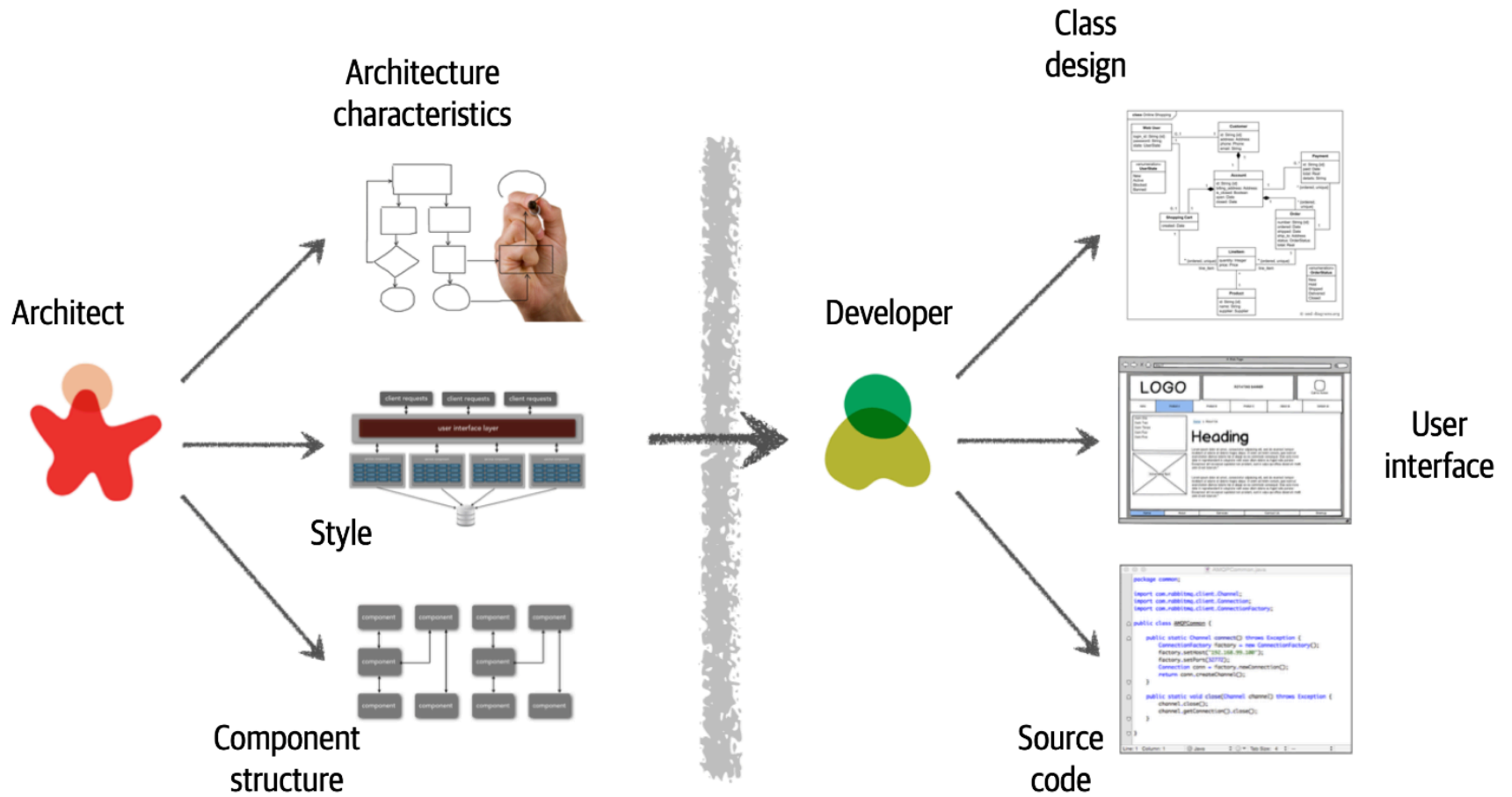
Do NOT attempt to disqualify or invalidate feature requests at this stage*. We don't make a determination of what to include or exclude until the Sprint planning stage.

*** An exception would be if something is infeasible, but even then, you should make a note in the requirement and provide the opportunity to change the requirement instead of rejecting it outright.**

Software Architecture

Architecture vs. Design?

- Both are concerned with how a system is structured and built.
- They reflect different levels of abstraction:
 - **Software architecture == high level of abstraction**, not to the level of actual code
 - Expose characteristics of the system (extensibility, scalability, performance)
 - **Software design == low-level of abstraction**, includes code
 - Expose characteristics of the code (readability, coupling, cohesion)



Software Architecture

Definition 1

“Expert developers working on a project have a shared understanding of the system design. This shared understanding is called ‘architecture’ [and] includes how the system is divided into components and how the components interact through interfaces.”

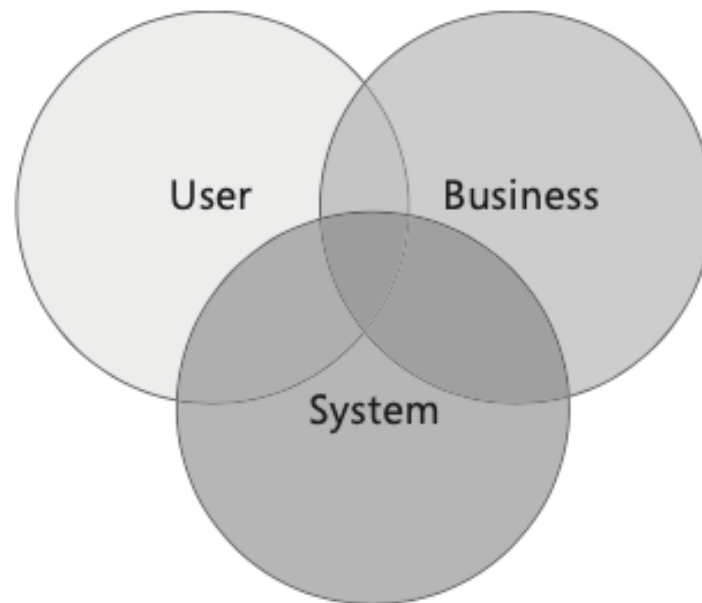
Definition 2

“Architecture is the set of design decisions that must be made early in a project [and that we would like to get right]”.

— Martin Fowler (2003)

Distinction between **BDUF** (Big Design Up-Front) and **SDUF** (Sufficient Design Up-Front).

We want to remain Agile, and we recognize that we *cannot* make every decision up-front!

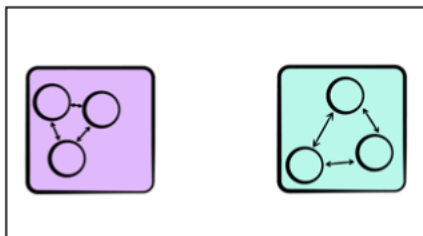


Architecture addresses the intersection of business goals, user goals and system qualities. The architect needs to determine how to deliver functional requirements in a way that addresses business needs (e.g. cost), and delivers desirable system characteristics (e.g. performance).

Structure

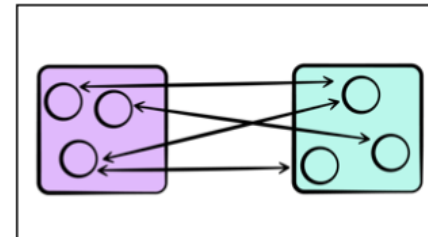
Modularity (Logical Structure)

Modularity refers to the logical grouping of source code into related groups. This can be realized as namespaces (C++), packages (Java or Kotlin). Modularity is important because it helps reinforce a separation of concerns, and also encourages reuse of source code through modules.



Cohesion

Cohesion is a measure of how related the parts of a module are.



Coupling

Coupling refers to the calls that are made between components.

When designing modules, we want high cohesion (where components in a module belong together) and low coupling between modules (meaning fewer dependencies). This increases system flexibility and scalability.

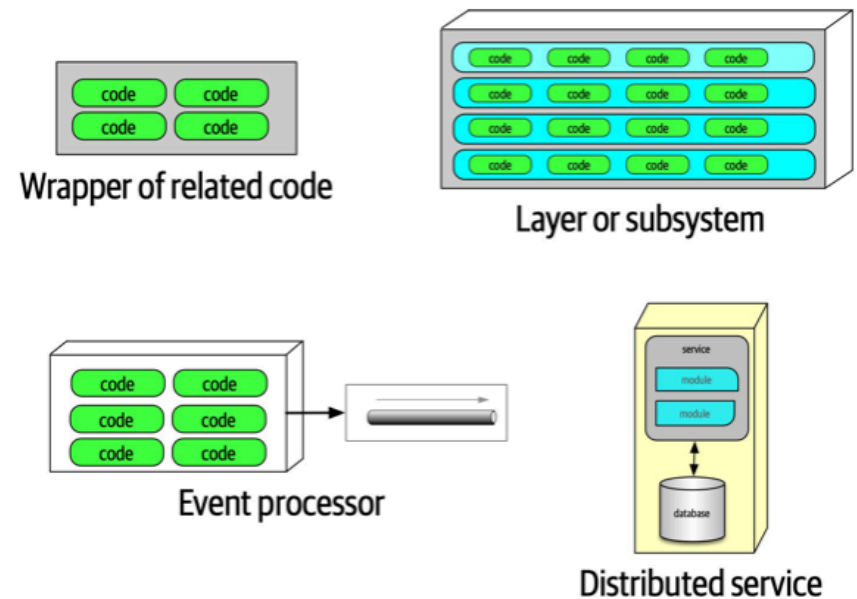
Components (Physical Structure)

Modules are logical collections of related code. Components are the physical manifestation of a module.

Library. A simple wrapper is often called a library, which tends to run in the same memory address as the calling code and communicate via language function call mechanisms.

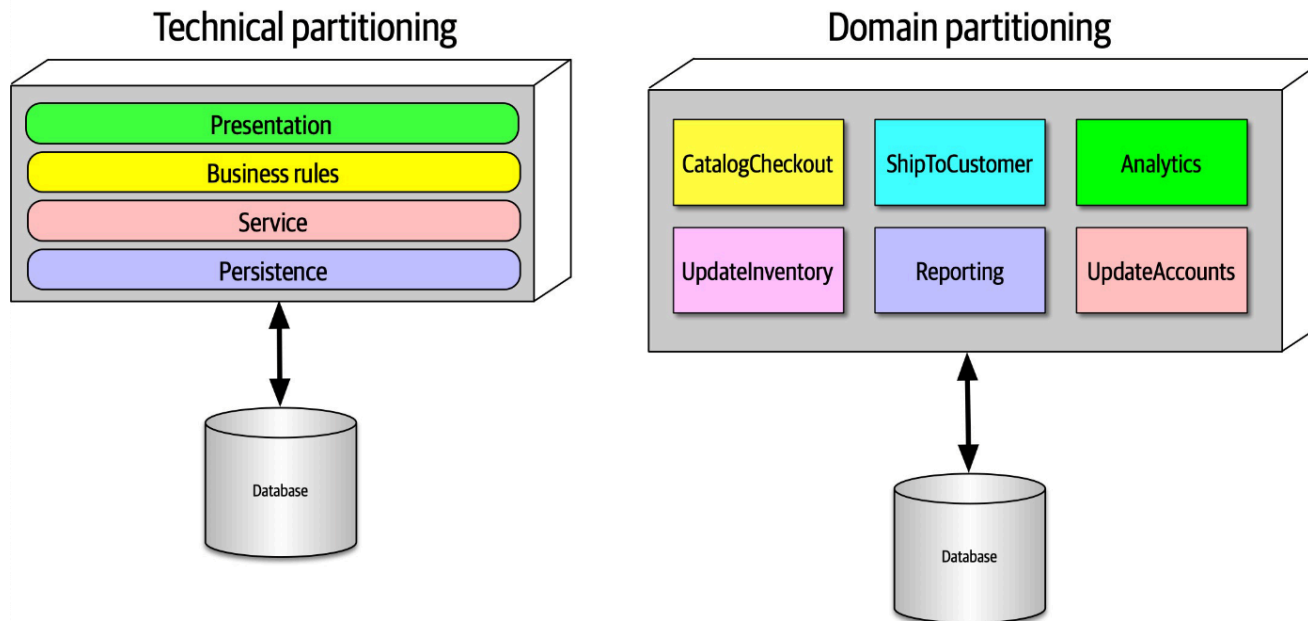
Layers or subsystems. Groups of related code deployed together that may communicate with one another directly.

Distributed service. A service tends to run in its own address space and communicates via low-level networking protocols like TCP/IP or higher-level formats like REST.



Partitioning

- **Technical partitioning:** we group functionality according to technical capabilities. e.g. presentation or UI layer, business rules or domain layer and so on.
- **Domain partitioning:** we group functionality according to the domain area or area of interest. e.g. a payment processing module, a shopping cart module, a reporting module and so on.



Architectural Characteristics

Non-Functional Requirements

The architecture phase is concerned with the non-functional requirements or “architectural characteristics” [Richards & Ford 2020] that emerge from the structure of our system.

These are all of the things that we might consider that aren’t directly related to the functional requirements.

An architectural characteristic meets three criteria:

1. specifies a non-domain design consideration (i.e. non-functional requirement),
2. influences some structural aspect of design,
3. is critical or important to the success of the application.

Examples of an architectural characteristic include software performance, and data security. These tend to not be identified as functional requirements, but should be considered!

Operational characteristics

These are often relevant for large-scale software systems with dedicated uptime, and typically a large number of concurrent users. e.g. Gmail, Facebook.

Term	Definition
Availability	How long the system will need to be available (if 24/7, steps need to be in place to allow the system to be up and running quickly in case of any failure).
Continuity	Disaster recovery capability.
Performance	Includes stress testing, peak analysis, analysis of the frequency of functions used, capacity required, and response times. Performance acceptance sometimes requires an exercise of its own, taking
Recoverability	Business continuity requirements (e.g., in case of a disaster, how quickly is the system required to be on-line again?). This will affect the backup strategy and requirements for duplicated hardware.
Reliability	Assess if the system needs to be fail-safe, or if it is mission critical in a way that affects lives. If it fails, will it cost the company large sums of money?
Robustness	Ability to handle error and boundary conditions while running if the internet connection goes down or if there's a power outage or hardware failure.
Scalability	Ability for the system to perform and operate as the number of users or requests increases.

Structural characteristics

These include characteristics related to code structure, code quality and production concerns.

Term	Definition
Configurability	Ability for the end users to easily change aspects of the software's configuration (through usable interfaces).
Extensibility	How important it is to plug new pieces of functionality in.
Installability	Ease of system installation on all necessary platforms.
Reuse	Ability to leverage common components across multiple products.
Localization	Support for multiple languages on entry/query screens in data fields; on reports, multibyte character requirements and units of measure or currencies.
Maintainability	How easy it is to apply changes and enhance the system?
Portability	Does the system need to run on more than one platform? (For example, does the frontend need to run against Oracle as well as SAP DB?)
Supportability	What level of technical support is needed by the application? What level of logging and other facilities are required to debug errors in the system?
Upgradeability	Ability to easily/quickly upgrade from a previous version of this application/solution to a newer version on servers and clients.

Cross-cutting characteristics

Term	Defintion
Accessibility	Access to all your users, including those with disabilities like colorblindness or hearing loss.
Archivability	Will the data need to be archived or deleted after a period of time? (For example, customer accounts are to be deleted after three months).
Authentication	Security requirements to ensure users are who they say they are.
Authorization	Security requirements to limit access to prescribed functionality based on assigned user roles or capabilities.
Legal	What legislative constraints is the system operating in (Sarbanes Oxley, GDPR, etc.)? What reservation rights does the company require? Any regulations regarding the way the application is to be deployed?
Privacy	Ability to hide transactions from internal company employees (encrypted transactions so even DBAs and network architects cannot see them).
Security	Does the data need to be encrypted in the database? Encrypted for network communication between internal systems? What type of authentication needs to be in place for remote user access?
Usability	Level of training required for users to achieve their goals with the application/solution.

Identifying Characteristics

So how do you identify which characteristics are relevant? You should look at the functional requirement (and project goals if they apply) and determine if there are any characteristics that help address those needs. i.e. what is important to your users?

For example:

Domain Concern	What do we look for?	Relevant Characteristics
Time to market	Characteristics that ensure that we can execute quickly and reliably.	Agility; Testability; Deployability
User satisfaction	Does your user work directly with this product? These are characteristics that directly affect user's perception, and availability of your product.	Performance; Usability; Configurability; Localization
Privacy	Some types of data are sensitive (e.g. working in Healthcare). Anything that would protect the integrity and access to data.	Authorization; Authentication; Security
Competitive advantage	In a competitive market, we want to be able to introduce and modify features quickly, even after we've released to the market.	Testability; Deployability; Supportability

Logging NFRs

Once you've identified a characteristic to address, you need to document it as a non-functional requirement (NFR).

- Include details of how you will measure it (all NFRs need to be measurable)
- Document how you will verify that you have met the requirement.
- Include consideration of this NFR in your topology and design (below).
- Clearly communicate this as an undercutting requirement to your team.

Example of non-functional requirements that would be included in our product backlog:

- "The application should launch and restore all state in less than 5 seconds on our target platform. We will measure this by manually timing execution during integration testing".
- "When the Save button is pressed, the entire state should be saved, and control returned to the user in less than 10 seconds. We will write unit tests to check the amount of time consumed by this action and ensure that it never exceeds this threshold".