ASR (Architecturally Significant Requirements)

Any requirement that greatly influences our choice of structures for the architecture is referred to as an architecturally significant requirement, or ASR. It is the role of the software architect to identify requirements that are architecturally significant.

Types Of Requirements

- **Constraints:** Unchangeable design decisions, usually given, sometimes chosen.
- **Quality Attributes**: Externally visible properties that characterize how the system operates in a specific context.
- **Influential Functional Requirements:** Features and functions that require special attention in the architecture.
- **Other Influencers**: Time, knowledge, experience, skills, office politics, Architect geeky biases, and all the other stuff that sways your decision making

Limit Design Options with Constraints

A constraint is an immutable design decision that an architect either forced to make or choose to make.

There are just a few restrictions in most software systems. Although all constraints limit options, well-chosen constraints can simplify the problem and make designing a satisficing architecture easier. Constraints can make life difficult for architects by restricting possibilities to the point that we are unable to meet other needs.

Constraints can influence technical or business concerns. Business constraints limit decisions about people, process, costs, and schedule. Technical constraints limit decisions about the technology we may use in the software system

Example:

Technical Constraints	Business Constraints
Programming Language Choice	Legal Constraints
We own SqlServer so that's your database.	It must be ready in time for the Big Indian Sale and cost less than $500,000

Constraints, once decided, are 100 percent non-negotiable.

Capture Constraints

Constraint	Origin	Type	Context
Must ship by the end of Q3.	Product Owner/ Stack Holder	Business	Avoids end of fiscal year budget issues.
Must support Chrome web browser	Stack Holder	Technical	Officially supported browser.

Define the Quality Attributes (Non-Functional Requirements)

Externally apparent qualities of a software system and the expectations for that system's operation are described by quality attributes. The quality qualities of a system specify how well it should accomplish a specific action.

When you are designing a software architecture, it's useful to distinguish between functionality, constraints, and quality attributes because each type of requirement implies a different set of forces are influencing the design. For example, constraints are nonnegotiable whereas quality attributes can be nuanced and involve significant trade-offs

Capture Quality Attributes as "Scenarios"

A quality attribute is just a word (Scalability, Performance, Availability). To offer an unambiguous description of a quality attribute, Architect employ a quality attribute scenario. Quality attribute scenarios describe how the software system is expected to operate within a certain environmental context. There is a functional component to each scenario—stimulus and response—just like any feature.

Example:-

Quality Attribute	Scenario	Priority
Performance	A user sees search results within 5 seconds when the system is at an average load of 2 searches per second.	High
Scalability	During a scheduled maintenance time, new servers can be added.	Low

Functional Requirements

Functional requirements, often captured as use cases or user stories. All functional requirements are essential to the success of the software system, but not all system features have architectural significance. When a functional requirement drives architectural decision making, call it an **influential functional requirement**

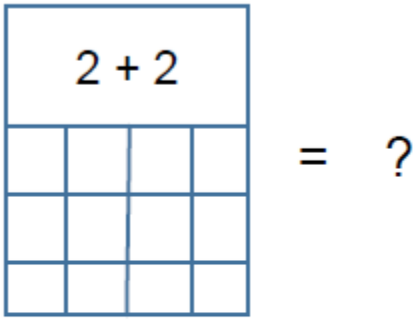
Influential functional requirements can be referred to as **architecture killers**. If architecture prevents Architect from implementing one of these high-value, high-priority features, Architect will be compelled to deconstruct architecture and begin over.

Identifying influential functional requirements

- Identify general classes of requirements that represent the same type of architectural problem

- Look for functional requirements that might be implemented within the same architectural elements. For example, features that require persistence go in one group whereas features that require user interaction go in another.
- Look for functional requirements that seem difficult to implement. These could be significant to the architecture
- Look for high-value, high-priority functional requirements.

Example :-



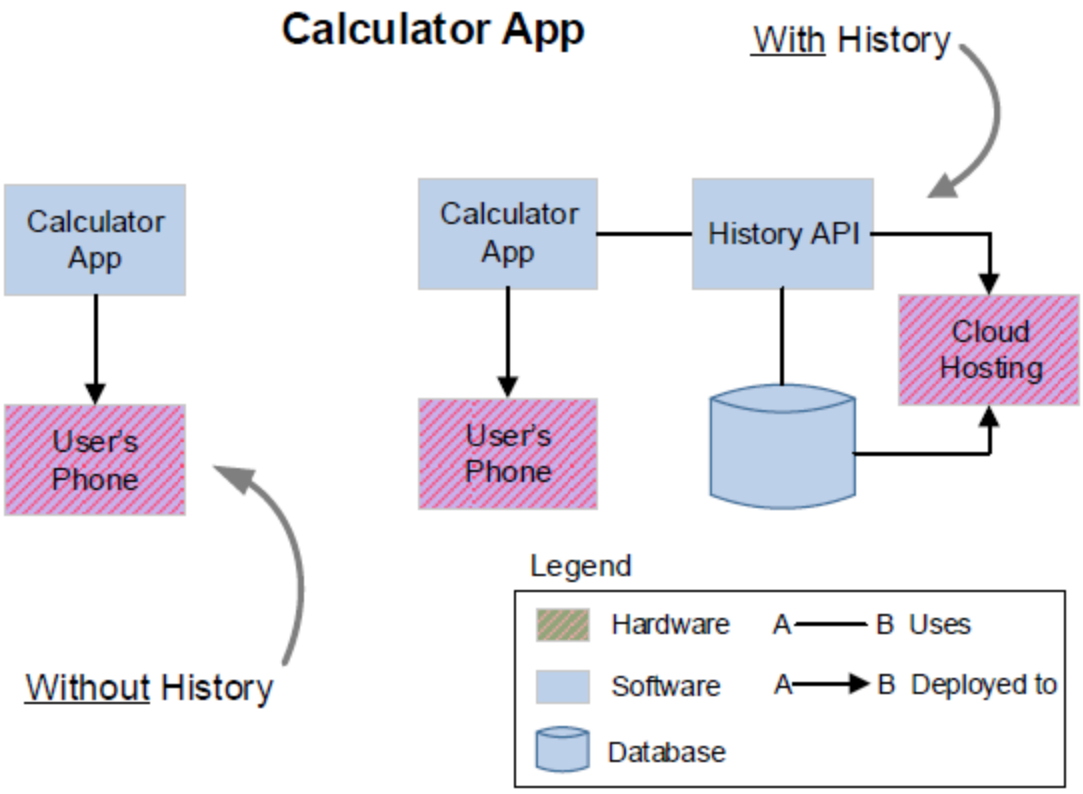
Calculator App

Here's an example. The simple calculator example: Adding two numbers together is an important functional requirement but has little influence on the architecture

So let's make it a little more interesting. As an Adder User, I can now review my addition history even if I've misplaced my phone. "People enjoy looking at previous work," the marketing department assures us. "It's going to be A-Mazing!" says "Wow Nice Feature"

Historical information? - We can save the user's actions to a local database. Wait… even if they've lost their phone?!?! Now we need a remote database server, which opens a ton of new questions. What happens when the user's phone is offline? What about availability? Scalability? Hosting costs? Syncing the app when the schema changes?

To solve the newly requested history feature, we need remote storage. This one feature takes the architecture in a new direction



Find Out What Else Influences the Architecture

"When all you have is a hammer, you will find plenty of nails to hit."