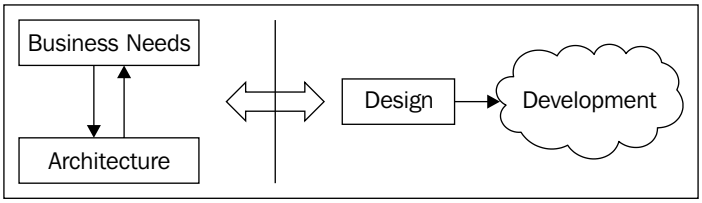


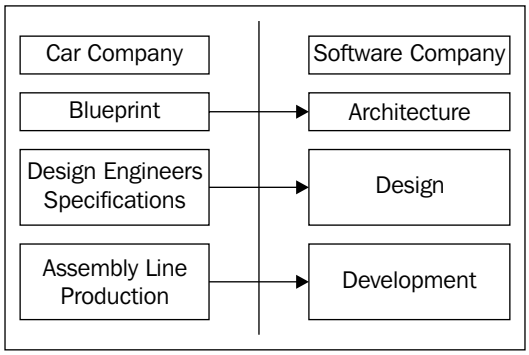
As we can see in the diagram, the actual business requirements and scope of the project are the deciding factors when working on the application architecture. Software design and development come next and, based on the design, the actual development work gets executed. A single problem can have many possible solutions, some of which will be more efficient than others. Before a developer starts chunking out code for a particular business requirement, it would be prudent and beneficial to give some thought and select the best approach from the possible list of options to assure that code performance, scalability and maintainability is not sacrificed in the long run.

In order to understand all of this by way of a simple analogy, consider a car manufacturing plant as an example. The mechanical engineers developing the high-level blueprint of the car would be the architects, and the blueprint itself would be the architecture of the car. This blueprint would include high-level specifications such as:

- Dimensions of the car and its components
- Engine capacity
- Type of car (hatchback, sedan, or SUV)
- Maximum passenger capacity, and load capacity
- Minimum build strength

So the blueprint would specify the limitations as well as the conditions that need to be fulfilled for any design of that car, and besides the blueprint there would be additional constraints such as the budget for the production costs. But this blueprint would not include details of how exactly the engine would be designed, what quality of steel would be used, what type of tires would be used, what type of plastics would be used for the dashboard and other parts, and so on. All of this would actually be decided by the design engineers, who will make sure that their choices fit the blueprint specifications in the best possible way. The engineers will also consider production and design techniques that other car companies might have followed, so that they don't re-invent the wheel.

The actual assembly line production will follow the designs and techniques specified by the engineers and will involve tasks such as cutting metal, choosing the right machines, assembling the individual components, painting, safety tests, and so on, to create a complete working car. The following figure will correlate this example with the equivalent aspects of software development:



From the figure we can see how the car company example loosely translates to software architecture, design, and development. Now let us take another analogy, this time more closely related to the software industry. Consider a company that needs to build a bulk emailing program for its social networking website. A software architect will first understand the high-level requirements of the program, such as:

- How many average emails need to be sent on a daily or hourly basis?
- How often will the emails need to be sent?
- Will there be attachments involved? If yes, what will be the biggest attachment size?
- Does this program need to be extensible and re-usable (for other similar websites or applications in that company)?

Based on the answers to the above questions, the architect will come up with an application architecture which covers all aspects of the actual business needs. The architecture will decide how the emailing program should be developed: a Windows Service, or a Web Service, or a console utility, or some batch program run by a scheduler.

But the architecture would not include details such as:

- How should the program handle exceptions?
- How will we make sure that the code is efficient in terms of performance, and does not hang while sending bulk emails in a short period?