**Q3 A**

Why might it make sense to teach analysis and design phases and activities sequentially, like a waterfall, even though in practice iterations are used in nearly all development projects?

Usually, analysis and design and implementation are taught separately, as though in a waterfall project because the knowledge and skills for each are separate topics. It is also valid to teach them in a waterfall fashion because within each iteration of an adaptive project a modified waterfall mini project often occurs. Also teaching analysis and design sequentially like a waterfall helps to organize the course and helps to explain the specific work that is done in each SDLC phase. Teaching an iteration approach first can confuse the work that is done in analysis and design.

Q3B

The pure waterfall attempts to specify completely the requirements during the analysis phase and then tries to "freeze" those requirements so that things do not change. Another solution is to have a modified waterfall so that additional analysis (i.e. specifying new requirements) can be done as the project progresses. However, neither situation is well suited to handle big changes during the life of a project.

Q3C

There are a variety of techniques to deploy new applications to production, so choosing the right strategy is an important decision, weighing the options in terms of the impact of change on the system, and on the end-users.

This document provides an overview of commonly used application deployment and testing patterns. It looks at how the patterns work, the benefits they offer, and things to consider when you implement them.

Suppose you want to upgrade a running application to a new version. To ensure a seamless rollout, you would typically consider the following:

* How to minimize application downtime, if any.
* How to manage and resolve incidents with minimal impact on users.
* How to address failed deployments in a reliable, effective way.
* How to minimize people and process errors to achieve predictable, repeatable deployments.

The deployment pattern you choose largely depends on your business goals. For example, you might need to roll out changes without any downtime, or roll out changes to an environment or a subset of users before you make a feature generally available. Each methodology discussed in this document accounts for particular

.

**Deployment strategies**

When you *deploy* a service, it's not always exposed immediately to users. Sometimes, it's only after the service is *released* that users see changes in the application. However, when a service is *released in-place*, deployment and release occur simultaneously. In this case, when you deploy the new version, it starts accepting production traffic. Alternatively, there are deployment strategies for provisioning multiple service versions in parallel. These deployment patterns let you control and manage which version receives an incoming request. The deployment patterns discussed in this section offer you flexibility in automating the release of new software. What approach is best for you depends upon your goals.

**Recreate deployment pattern**

With a recreate deployment, you fully scale down the existing application version before you scale up the new application version.

The following diagram shows how a recreate deployment works for an application.

Version 1 represents the current application version, and Version 2 represents the new application version. When you update the current application version, you first scale down the existing replicas of Version 1 to zero, and then you concurrently deploy replicas with the new version.

**Key benefits**

The advantage of the recreate approach is its simplicity. You don't have to manage more than one application version in parallel, and therefore you avoid backward compatibility challenges for your data and applications.

**Considerations**

The recreate method involves downtime during the update process. Downtime is not an issue for applications that can handle maintenance windows or outages. However, if you have mission-critical applications with high service level agreements (SLAs) and availability requirements, you might choose a different deployment strategy.

**Rolling update deployment pattern**

In a rolling update deployment, you update a subset of running application instances instead of simultaneously updating every application instance, as the following diagram shows.

In this deployment approach, the number of instances that you update simultaneously is called the *window size*. In the preceding diagram, the rolling update has a window size of 1. One application instance is updated at a time. If you have a large cluster, you might increase the window size.

With rolling updates, you have flexibility in how you update your application:

* You can scale up the application instances with the new version before you scale down the old version (a process known as a *surge upgrade*).

You can specify the maximum number of application instances that remain unavailable while you scale up new instances in parallel.