**Traditional SDLC**

The first example will be the Traditional SDLC, often built within what is known as 'silos'. Silos are individual teams throughout the production process with their focus on their specific scope. The representation for this will be the unique pictures encapsulating each silo.

**Dev’s to QA**

Anything developers produce in their environment has to be passed to the Quality Assurance team before reaching the operations area. The example will start with the developers' silo writing the code, that is their scope, and when it meets their brief, they hand it off to QA. From that moment, the developers' are finished and no longer involved in the process. However, this would be in an ideal world.

**QA back to Dev’s**

If the QA team finds an issue, maybe a bug or missed objective with their code, it will be returned to rectify. This 'back and forth' could happen once, twice, or numerous times before the QA silo approves the work and moves the responsibilities to the operations silo.

**QA to Ops**

Once the QA has completed their checks, they will move the code to the operations silo. They have achieved their goal and now have no responsibility for the process as their job is complete.

**Ops back to QA**

Now that it is in the hands of the operations team, they try to deploy this code, they find it is not deployable, or a bug stops the production. This roadblock for the operation teams requires them to throw it back to the developers to rectify it. At this time is where it can often create a rift in the holistic development cycle.

**Not my problem!**

The developers and QA have said the code meets the brief, but the operation teams say the polar opposite. The viewpoints of the developers and the operations teams are different because they are encapsulated in their environment and often do not have visibility into the neighbouring system. If this usually occurs, it can be from the developers not seeing the production side and operations have little visibility of the code. A blame or ' finger-pointing ' scenario could play out because work moves to a stand-still until something can continue the production process. Both developers and operations would claim it is not their responsibility, and developers would say their code meets the brief and QA approved it. While operations will say their system is perfect for production, the only new thing for them is the code that gives them an issue.

**The issue**

Because of the lack of visibility between silos, and if this scenario played out, it would be catastrophic. Valuable investment of time and resources are required to even start working forward on a solutio. When the developers create additional code changes and deliver them to the operations team, they could delay the changes to maintain server uptime and stability. While developers want to keep providing the new code and its features to the operations. This solve the initial issue, it is often against the other teams' success metrics.

**Summary**

From this example where things went wrong at multiple stages of the production process, we saw a disjointed product due to the developers and operations individual objectives. Because of the development teams issues and not being able for operations to smoothly transition the code to production, it creates a slow time-to-market and affects the stakeholders. Further delays from the operations team deploying any of the developers new code that has features or updates to the platform will often be an issue for customers to see the best and most current version of the product.