**Architecture feedback**

The task at hand is as follows:  
  
“discuss at least one reason you would not want to implement architectures 1 and 2”.

Architecture 1 seems like a step in the right direction in comparison with the current setup. But only half a step at best. The GUI parts interact via REST with the centralised *emailService* while the service itself uses *billingService* for its purposes. This still means that *emailService* contains all the logic and development means people getting in the way of each other and deliveries are thereby also heavier since the entire *emailService* has to be delivered. *emailService* is in effect a single point of failure. Also the GUI itself should not integrate directly with the email platform service – it should at least go via *emailService* which already has integration with the email platform. Seems like duplication here. Also going counter to the purpose services seem to be serving in this architecture.

Architecture 2 is a further evolution of architecture 1 where it splits parts of the integration layer into separate components in what appears so be domain-specific ways. Whereas architecture 1 had a central *emailService* that the GUI parts and external email platform were using – architecture 2 has dedicated services for parts of functionality that the application contains. Such as user info service, a service for billing and pricing as well as a service to interact with the email platform. The problem here is that services should be using other services and the GUI should go via their dedicated REST endpoints so that we can manage those in the backend rather than the GUI. For example I can see that Architecture 2 has *myEmailGUI* directly call *platformService* which integrates with the email platform. Another problem is that *pricingService* directly acts on the user store rather than going via the *userService*. Seems to be going against the spirit of that architecture 2 purports to embody.

Once you see Architecture 3 it becomes clear that we are going for a service architecture where each service has their domain and interactions are between services and the GUI does not create close coupling between itself and services it should not know about.