**Accessibility**

* To ensure that access is in a testable state, have examples of:
  + Physical Users
    - Those who consume the API
    - Those who consume the Web
    - Those who consume both
    - Those who administer the above
  + Virtual Users
    - Domain accounts for servers
    - Database user accounts

**Observability**

* Logging of relevant verbosity for:
  + Load Balancer routes used and sessions created.
  + Run time and origin server of searches in web layer
* Snapshot
  + Take a database snapshot (showing each transaction in each table) before and after a certain scenario.
* Monitoring
  + Status level monitoring of each app
    - Including responsiveness over time.
    - Allow to grow a significant dataset to combat small sample size fallacies.

**Controllability**

* Inject controllability into each layer:
  + External and internal services – isolate ID Resolver by mocking these directly. Less variability involved.
  + Webservers – able to start, stop, and interrupt each application for website, api, cache and batch.
  + Database – able to start, stop and interrupt database replication and snapshot at various stages.

**Simplicity**

* Split this into two:
  + External – the external services (card gateways) probably adhere to certain data contracts and change less as clients value stability.
  + Internal – probably more complexity here, checks which follow data through the system, reporting on state at key points will expose information, using a common reference (client id for example)

**Unbugginess**

* Historical bug review, based on the decomposition of each application:
  + Take a sample of bugs from the recent past identify any hotspots.
  + Also, mark any areas of the application that may be untested, subject to less testing, or just lacking an oracle.

**Unbugginess (cont)**

* + Overlay these two and look to create tests which ‘light the way’ in these areas.

**Smallness**

* ID Resolver has apps for each job and databases are partitioned.
* Look to expand on the impact of shared resources such as the Load Balancer and its impacts to ID Resolver to establish the system boundary.

**Decomposability**

* A two stage process:
  + Note the language used for each layer, app and component in the technical teams.
  + Note the language used for each layer, app and component in the business context.
* Overlay the two and mark common terminology and differences.
  + Commonality – check for deeper understanding here ensure it is not superficial.
  + Different – Consider tests to expose function in these areas.
* Create a high level map of structure and function, with drill down maps of areas that require more exposure.

**Similarity**

* Split this into two comparative elements:
  + Technical – are there any systems that operate similarly on a similar technology stack?
  + Business – are there any other businesses that operate in the same space? These can be internal and external?

**Tool Availability**

* Tooling would be useful to interrogate each layer for state and possibly subject to load:
  + Load Balancer
  + External Services
  + API
  + Batch File Processor
  + Database

**Data Availability**

* ID Resolver is constrained by having both personal and card data.
* A muddied dataset of a similar size to the live environment would provide greater testability, as the underlying system demands would reflect the live environment.