This document covers online applications and any corresponding small footprint client applications. Applications that are specifically not in scope for this document are: NGE, UM, WCAp, MDM, NGC, Total Loss, RepairCenter, eGlass, GlassMate, and FastPhoto.

Software Engineering Principles

Statement of Purpose: We design and test software that models business processes as we currently understand them. As our understanding changes our design and testing changes. The software we provide to customers is a design that has been internally validated. We continually improving the model by improving the internal validation and external validation of our design. We believe the following principles create the best software models:

1. We believe in doing things well over doing things quickly.
2. We believe engineering products safe to fail over fail safe.
3. We believe in resiliency (decreasing internal complexity and anticipating change) over expediency.
4. We believe in instrumentation over narratives.
5. We believe in organizing complexity over eliminating complexity.
6. We believe in continuous validation over code stability.
7. We believe in knowledgeable teams over expert individuals.
8. We believe in automated systems over documented processes.
9. We believe in creating an architecture that facilitates solution autonomy over allowing anyone to work on anything.
10. We believe in using more qualified, fewer people over using large numbers to meet customer commitments.
11. We believe in continuous improvement over large organizational initiatives.
12. We believe in consistency over next best new thing.

Our architecture, component ownership, and development plans are designed to support our engineering principles.

Architecture

1. Customer Solution
    1.1. Purpose: General Market solution with at least one customer specific component
    1.2. Examples: MPI, ICBC, Progressive, ABRA
2. General Market Solution

2.1. Purpose: Collection of core components and application components that serve as solution space for a Mitchell market segment

2.2. Examples: CarStar, San Diego Mercedes Benz

3. UI Application

3.1. Purpose: Provides a branded set of UI components to a Mitchell market segment

3.2. Examples: WorkCenter, RC Connect 2.0, IA Connect, WC Connect, RC Mobile

4. APD Product UI Components

4.1. Purpose: Provide summary, detailed, list, and manager views of core domain objects

4.2. Examples: Estimate card, estimate list, estimate manager, message card, message list

5. Customer Solution UI Components

5.1. Purpose: Provide custom views of core domain objects, customer specific data, or customer specific objects

5.2. Examples:Assignment pull

6. APD Product UI Framework

6.1. Purpose: Provide common graphical components to construct core UI components, customer UI Components, core UI applications, or customer UI applications

6.2. Examples: Left navigation, UI container, stylesheets

7. Enterprise Platform UI Framework

7.1. Purpose: Provide common graphical design guidelines, standard component definitions, color palettes, and user experience standards

7.2. Examples: Product design solution structure, product design patterns, product design style, and product design building blocks

8. APD Product Application Services

8.1. Purpose: Provide coordination of APD Application Domain Services. Does not provide customer specific logic.

8.2. Examples:

9. Customer Solution Application Services Decorators

9.1. Purpose: Intercept requests to APD Product Application Services to provide pre/post request processing.

9.2. Examples: Assignment rules, estimate retrieval rules, filtering of results

10. APD Application Domain Services
    10.1. Purpose: Provide API for APD domain objects
    10.2. Examples: Job Service, Org Service, Estimate Service
11. Enterprise Platform Services
    11.1. Purpose: Provide API for enterprise objects. Enterprise are APD agnostic.
    11.2. Examples: Docstore
12. Message Bus
    12.1. Purpose: Provide mechanism to publish events.
    12.2. Examples: Message Bus
13. APD Product Listeners
    13.1. Purpose: Provide asynchronous behavior for events for APD product components.
    13.2. Examples: NonDRP Notification
    13.3.
14. Customer Solution Listeners
    14.1. Purpose: Provide asynchronous behavior for events for customer specific requirements
    14.2. Examples: File Delivery
15. Enterprise Listeners
    15.1. Purpose: Provide APD agnostic behavior for APD events
    15.2. Examples: Notification

Ownership

Component ownership is defined to maximize:

- Quality software delivery
- Autonomy of outcomes
- Consistency within the respective software domains

These qualities necessarily drive maximum engineering output.

Customer Team Mission - Customer teams deliver useful features to customers given the current version of the APD product platform and the Enterprise product platform.

APD Product Platform Team Mission - APD product platform teams deliver useful features to facilitate customer solutions given the current version of the Enterprise product platform. Product teams mine the customer solutions to identify improvements to the APD product platform.

Enterprise Platform Team Mission - Enterprise platform teams deliver product agnostic software components to simplify APD Product Platform Teams and Customer Teams development.

The following table illustrates the ownership of different software components:

| Team | UI | Service | Data |
|---|---|---|---|
| Customer Teams | Customer UI Components: Notification Templates, Dispatch Reports, Customer UI widgets | Customer Solution Filters: Estimate Retrieval Rules, Assignment Retrieval Rules, … | |
| Core Product Teams | Core UI Components: Job List, Job Overview, Estimate Card, … | APD Core APIs: Jobs API, Claims API, Task API, Assignment API, … | Amazon S3 |
| Enterprise Platform Team/User Experience | Product design solution structure, product design patterns, product design style, and product design building blocks | Enterprise APIs: Document Storage API | Document Storage |
| Data Management | | | CCDB, EPFP, LDAP |

Development Plans

Our preference is for teams to work in their area of expertise. Our architecture is designed to support teams doing full stack development in the software components they own. Sometimes we create projects where teams need work done in other areas of the system to realize their solution. Our preference for delivery of the project is:

- Dependency-less development based on existing software components
- Dependency identified in team backlog and delivered accordingly
- Engineers from dependent team embedded in owning team

- Dependent team makes changes to code of owning team.

Even with this preference, owning teams have the responsibility to review changes and to determine what changes can be made to the owning team's code. That is, it is up to the owning team to identify which software components/subcomponents are available to certain development plans.