

# System Architect Essentials II

## Lesson Reference Documents

**© Copyright 2014**  
**Pegasystems Inc., Cambridge, MA**  
All rights reserved.

This document describes products and services of Pegasystems Inc. It may contain trade secrets and proprietary information. The document and product are protected by copyright and distributed under licenses restricting their use, copying, distribution, or transmittal in any form without prior written authorization of Pegasystems Inc.

This document is current as of the date of publication only. Changes in the document may be made from time to time at the discretion of Pegasystems. This document remains the property of Pegasystems and must be returned to it upon request. This document does not imply any commitment to offer or deliver the products or services provided.

This document may include references to Pegasystems product features that have not been licensed by your company. If you have questions about whether a particular capability is included in your installation, please consult your Pegasystems service consultant.

PegaRULES, Process Commander, SmartBPM® and the Pegasystems logo are trademarks or registered trademarks of Pegasystems Inc. All other product names, logos and symbols may be registered trademarks of their respective owners.

Although Pegasystems Inc. strives for accuracy in its publications, any publication may contain inaccuracies or typographical errors. This document or Help System could contain technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Pegasystems Inc. may make improvements and/or changes in the information described herein at any time.

This document is the property of:

Pegasystems Inc.  
1 Rogers Street  
Cambridge, MA 02142  
Phone: (617) 374-9600  
Fax: (617) 374-9620  
[www.pega.com](http://www.pega.com)

**Document Name:** sae2\_716\_LessonReferenceDocuments\_20141121.pdf

**Date:** 20141121

# Table of Contents

Orientation.....	1
Course Overview.....	2
Effective Application Development with PRPC.....	7
HR Services Application Operational Walkthrough.....	8
The Building Blocks of a PRPC Application.....	22
Managing the Building Blocks of a PRPC Application.....	33
Guided Application Development Using Guardrails.....	62
Designing Enterprise Applications Using Case Management.....	73
Best Practices for Case Management Design.....	74
Managing Enterprise Apps using Stage-Based Case Design.....	100
Best Practices for Effective Case Decomposition.....	129
Best Practices for Effective Process Decomposition.....	174
Guardrails for Case Management Design.....	198
Creating an Effective Data Model.....	205
Best Practices for Designing a Data Model.....	206
Best Practices for Managing Data.....	228
Best Practices for Managing Reference Data.....	241
Sharing Data Across Cases and Subcases.....	287
Guardrails for Data Models.....	296
Integrating with External Data Sources.....	302
Integrating with Databases.....	303
Guardrails for Integrating with External Data Sources.....	312
Creating Engaging User Experiences.....	319
Designing the User Interface for Reuse and Maintainability.....	320
Building Assignment Focused (Intent-Driven) User Interfaces.....	337
Best Practices for Designing the User Interface.....	350
Using Advanced User Interface Controls.....	367
Managing Data for Selectable List Controls.....	376
Building Dynamic User Interfaces.....	383
Validating User Input.....	390

Guardrails for Creating Engaging User Experiences.....	403
<b>Enforcing Business Policies.....</b>	<b>409</b>
Designing Business Rules for the Business User.....	410
Enforcing Business Policies using Service Levels.....	428
Notifying Users from Within a Process.....	445
Enforcing Business Policies Using Decision Rules.....	465
Enforcing Data Relationships with Declarative Rules.....	480
Guardrails for Enforcing Business Policies.....	496
<b>Process Visibility Through Business Reporting.....</b>	<b>501</b>
Preparing Your Data for Reporting.....	502
Building Business Reports.....	512
Guardrails for Business Reporting.....	524
<b>Best Practices for Preparing an Application for Testing Deployment.....</b>	<b>532</b>
Using Guardrail Reports to Ensure the Best Performance.....	533
Guidelines for Maintaining Requirements and Specifications.....	550

## Module 01: Orientation

This lesson group includes the following lessons:

- Course Overview

## Course Overview

**In this course, we learn best practices and design patterns for implementing case-based business applications using Pega 7.**



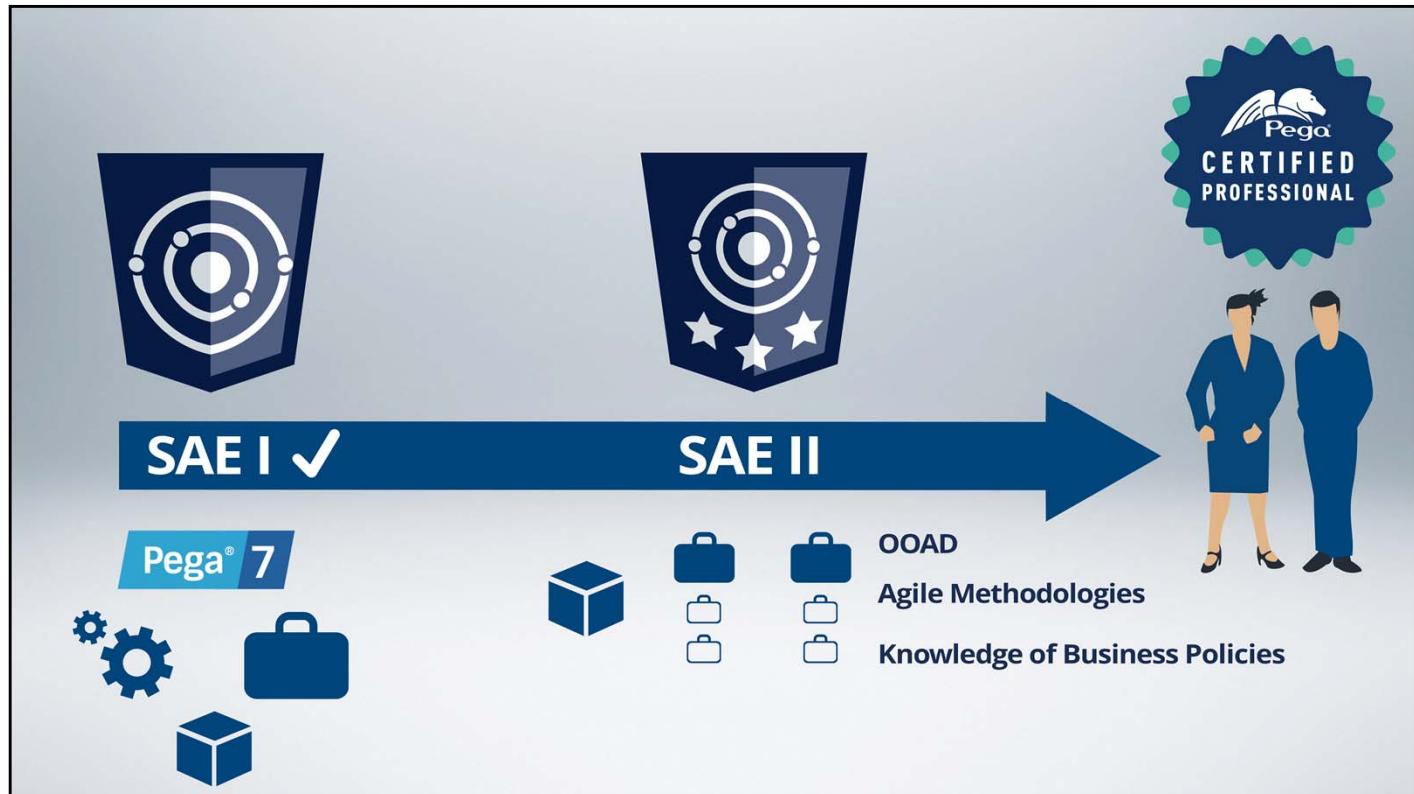
Welcome to the System Architect Essentials, Level Two course.

This course is designed for architects who have attended System Architect Essentials - Level One,

or have a solid working knowledge of how to create a basic case type.

In this course, we assume you know how to create the necessary components to create a basic case type using Pega 7 so our focus is on best practices and design patterns.

We'll extend a previously built application to support additional case management requirements and capabilities.



It will help to have some knowledge of – or experience with - object-oriented application design and familiarity, or experience with agile project methodologies.

Knowledge of your company's business practices and policies will also help you translate the methods learned in this course to your “real world” requirements.



This course is organized into five broad categories.

We'll start with case management design patterns.

Our goal here is to learn how to design case types that are easily communicated but contain enough implementation detail to be executed directly – without modification.

Next, we'll explore effective data management techniques and best practices

that make interacting with data, including data from external sources, more intuitive.

The third category is devoted to learning how to create an engaging user experience.

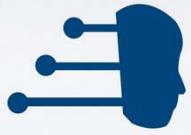
We'll study best practices and learn how to use advanced user interface controls.

Then, we'll look at best practices for automating business policies.

Finally, we'll cover how to provide business performance visibility through reporting.

**Pega® 7**

**LESSONS SHOULD BE TAKEN IN ORDER**

Video	Exercise	Quiz
		
		
Best Practices	Hands-on Experience	Assess Knowledge

Each lesson consists of a video and a transcript, and is presented as a case-study of a selected business problem.

In the video we explore best practices for creating the most effective solution.

Almost every lesson has an exercise to help reinforce the case study and provide you hands-on experience with Pega 7.

Each lesson includes a quiz to assess your knowledge of the topic.

Given that each lesson builds upon a previous lesson, we recommend that you take the lessons in the order they are presented.

We look forward to helping you learn what Pega 7 has to offer and how it provides you the power to simplify!

## Module 02: Effective Application Development with PRPC

This lesson group includes the following lessons:

- HR Services Application Operational Walkthrough
- The Building Blocks of a PRPC Application
- Managing the Building Blocks of a PRPC Application
- Guided Application Development Using Guardrails

# HR Services Application Operational Walkthrough

You've been tasked to create an application for hiring new employees from being a candidate through orientation. So far, you have implemented the Candidate case that is responsible for the interview and offer processes. Now that someone has accepted an offer you need to create cases and processes that revolve around onboarding the new employee.

**At the end of this lesson, you should be able to:**

- Overview of the HR Services application
- Review the Candidate case type
- Confirm your understanding of the intent of System Architect Essentials II
- Overview the Onboarding case type
- Confirm your understanding of, and ability to build, basic rules in Pega 7

System Architect Essentials I and II		
	SAE I	SAE II
What we do	Manage candidates for open positions 	Manage on-boarding for new employees 
How we do it	Single case type 	Multiple case types 
What we learn	Explore basic concepts and capabilities 	Explore application design patterns 

Let's begin with a review of the HR Services Application – and confirm our understanding of the scope of this course.

If you attended System Architect Essentials I,

the goal was to implement a proof-of-concept solution to manage job applicants for open positions.

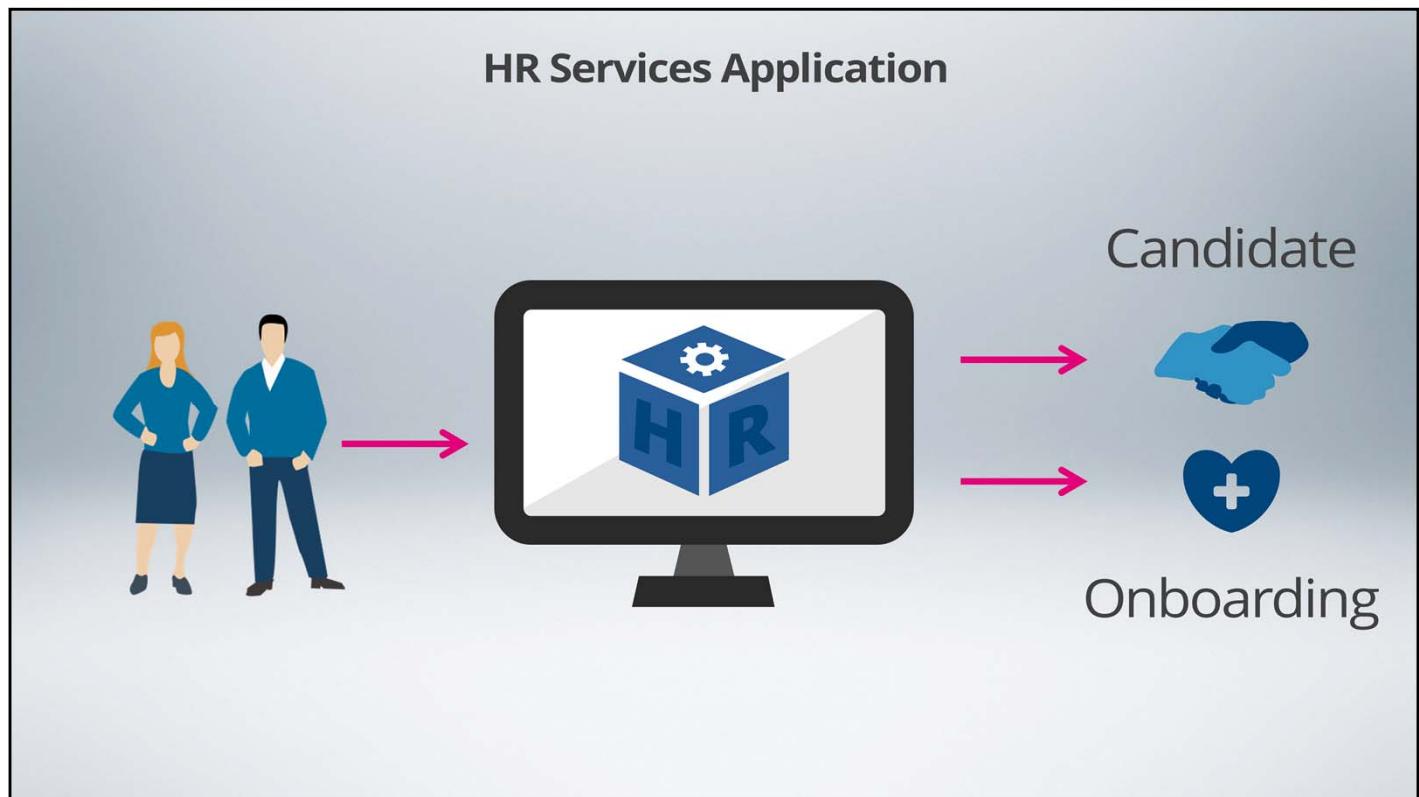
This proof-of-concept was designed to provide experience with the basic capabilities of the Pega 7 platform and reinforce its foundational concepts.

These “basic capabilities” – creating rule types such as flows, user interfaces, properties, and decisions; and navigating the Designer Studio – are ones that System Architects must understand, and know very well, to become productive members of an implementation team.

In System Architect Essentials II, the goal is to extend the HR Services application by implementing a solution to manage the on-boarding process for new hires.

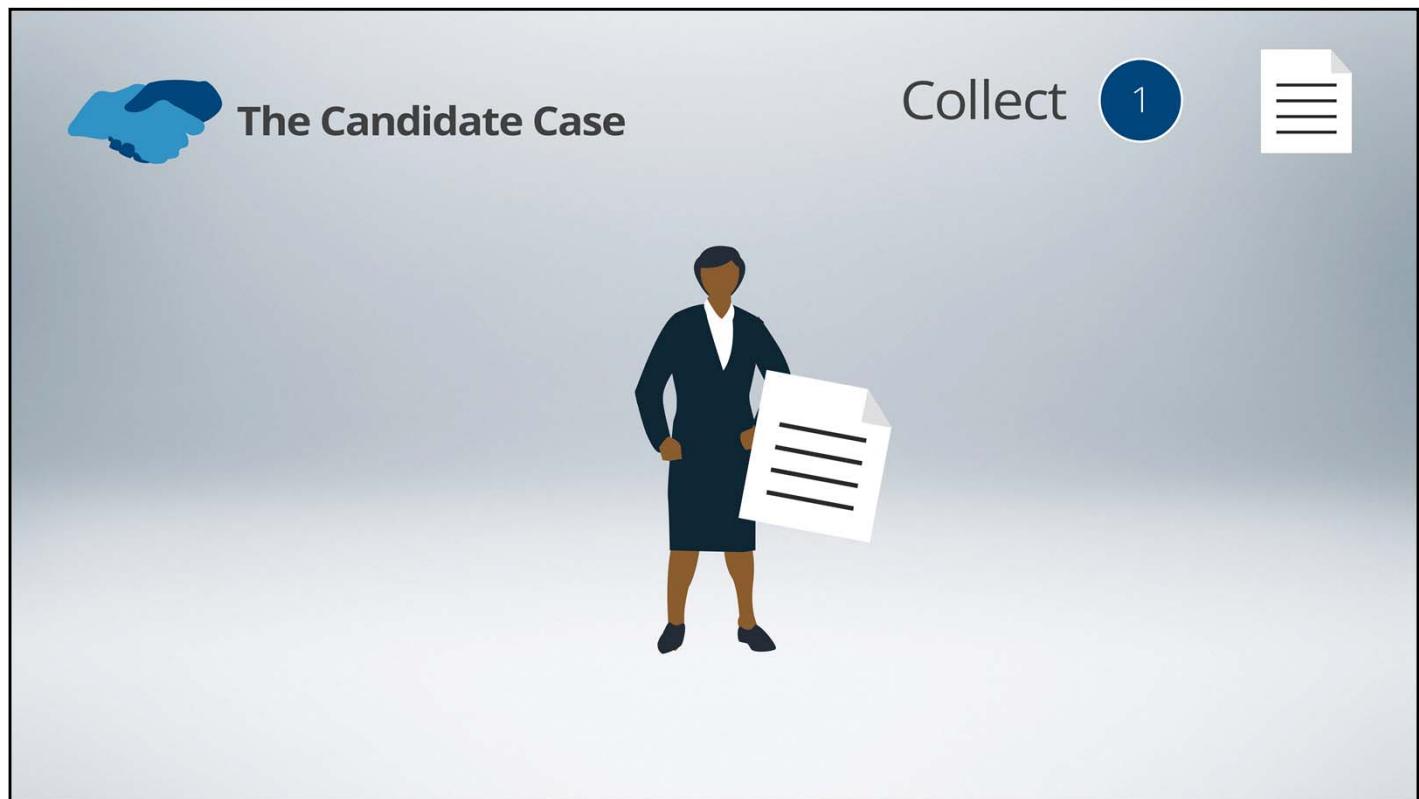
The solution built in System Architect Essentials II is more sophisticated than the Candidate solution, involving multiple case types.

We build upon the basics covered in System Architect Essentials I by learning design pattern principles and best practices to create a more robust solution. Whenever possible, we will seek to reuse elements of the Candidate solution, to leverage existing functionality rather than re-implement it.



The Candidate and Onboarding solutions comprise the HRServices application.

When authorized employees log in to the HR Services application,  
they can create the case that matches their need;  
either when a candidate expresses interest in a position,  
or when they accept a job offer.



Let's review the Candidate case type implemented in the System Architect Essentials I course.

The goal of the initial solution was to provide a proof-of-concept for solving a simple business problem: manage – automate, actually - the processing of applicants for an open position, which involves the following tasks:

collecting information about the candidate,

The Candidate Case

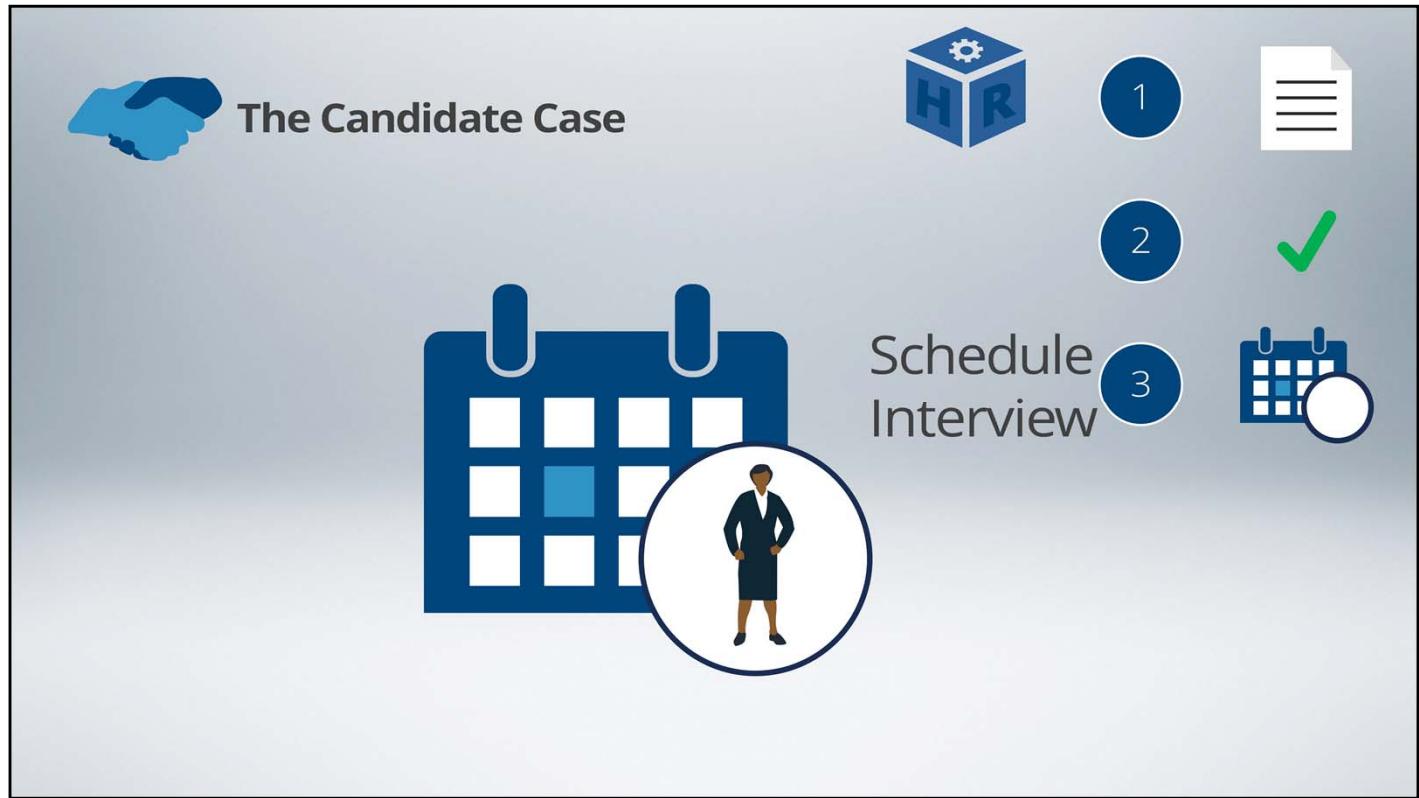
Match 2 ✓

Senior Accountant

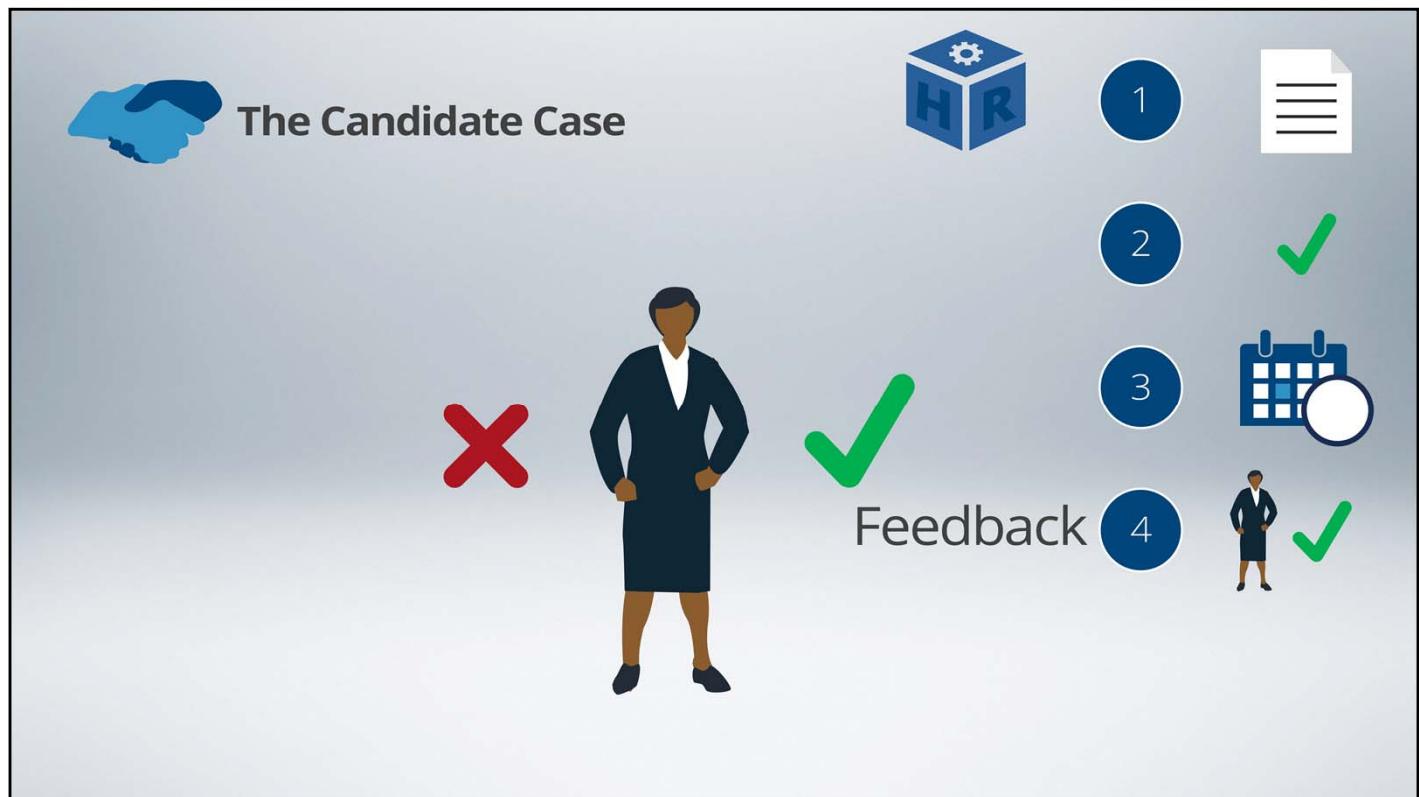
Project Manager

Account Executive

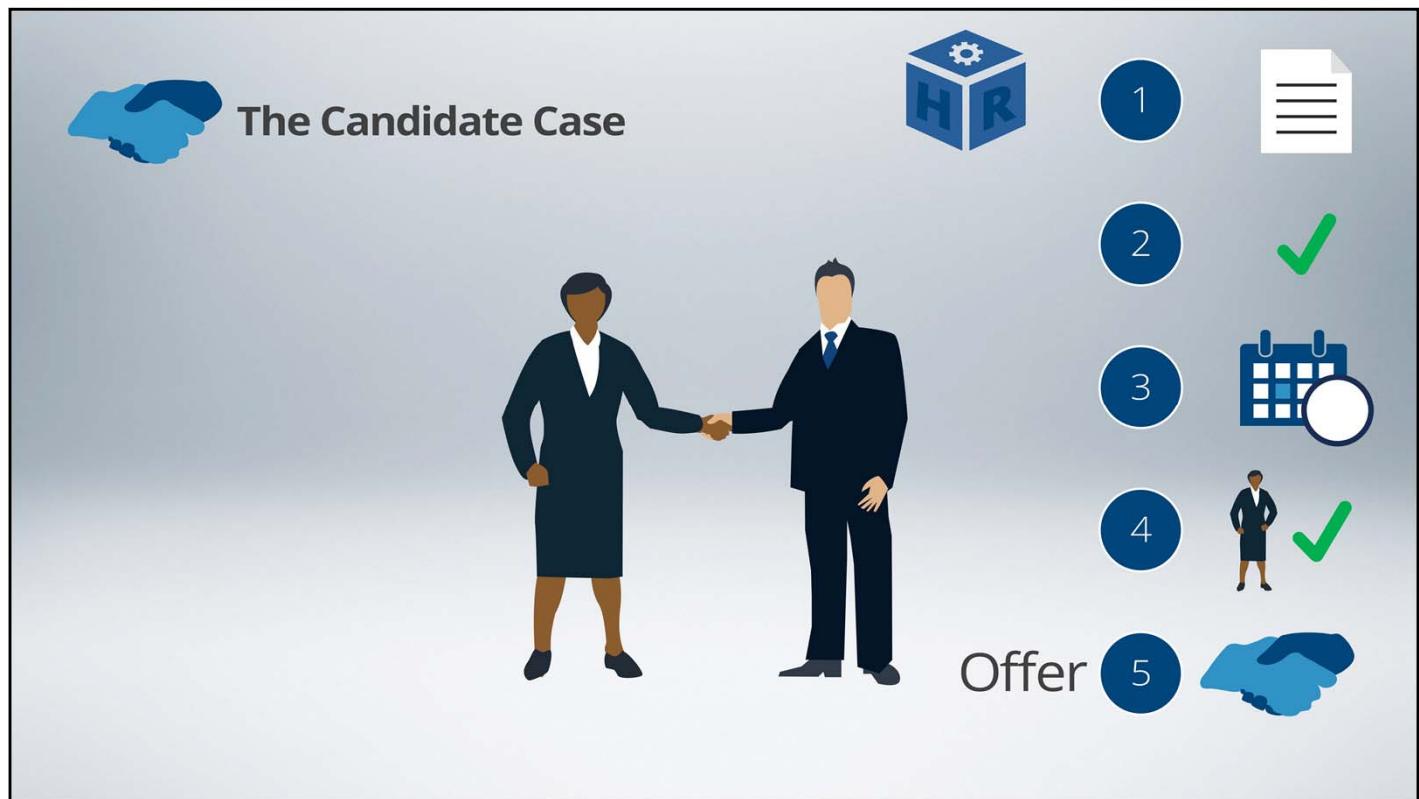
matching the candidate to an open position,



scheduling an interview with a hiring manager,



collecting and processing the feedback upon which a hiring decision is made, and



then extending an offer to the candidate.



## The Candidate Case



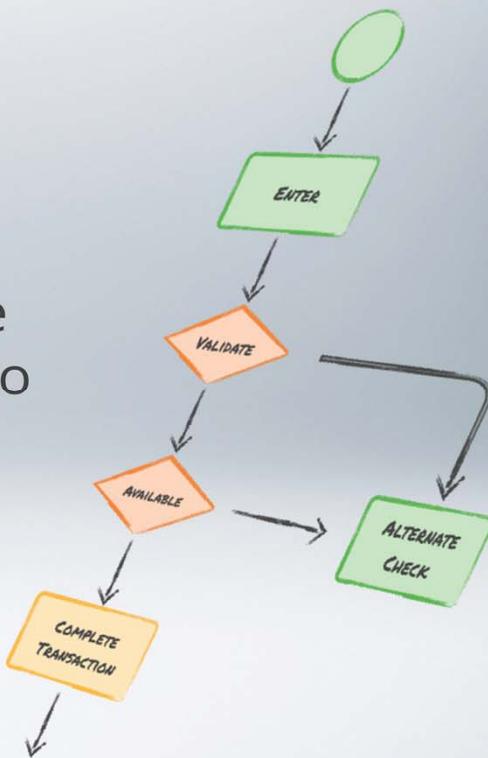
Shows how an Application is built using individual rules

<b>RULE</b> CASE TYPE 	<b>RULE</b> FLOW ACTION 
<b>RULE</b> FLOW 	<b>RULE</b> SECTION 
<b>RULE</b> DECISION 	<b>RULE</b> PROPERTY 

This proof-of-concept was designed to show how a stage-based case management application is built using individual components – which are commonly referred to as rules.

Once we understand how to build the individual components for a Pega case management application, we can then turn our attention to “design patterns”.

This is the goal of **System Architect Essentials II**.

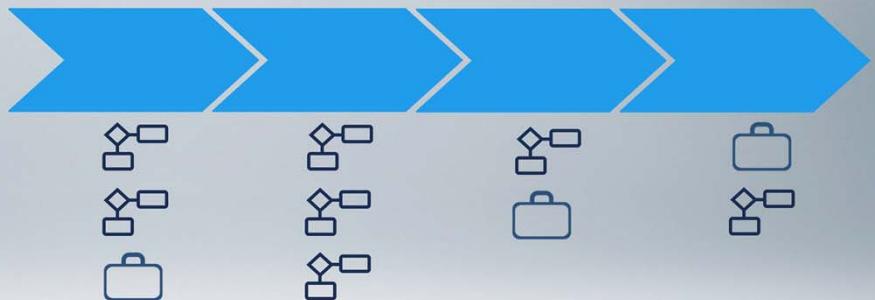


Once we understand – and know very well - how to build the individual components for a Pega case management application, we can then turn our attention to “design patterns” – and this is the goal of this course, System Architect Essentials II - design patterns.

## Business Problem

We need these things to happen in this sequence

## Pega Solution



## DESIGN PATTERNS – BEST PRACTICES

Our focus now is to learn how to read a user story – or business problem and translate that business problem into functional design patterns using Pega.

The goal of System Architect Essentials II is to focus on design patterns and best practices when building stage-based case management solutions using the Pega platform.

# Onboarding Case – Possible Solution Steps

1



Create Case

2



Schedule Orientation

3



Benefits Enrollment

4



“30-60-90” Program



Send Welcome Kit



Asset Provision



Setup Payroll Account



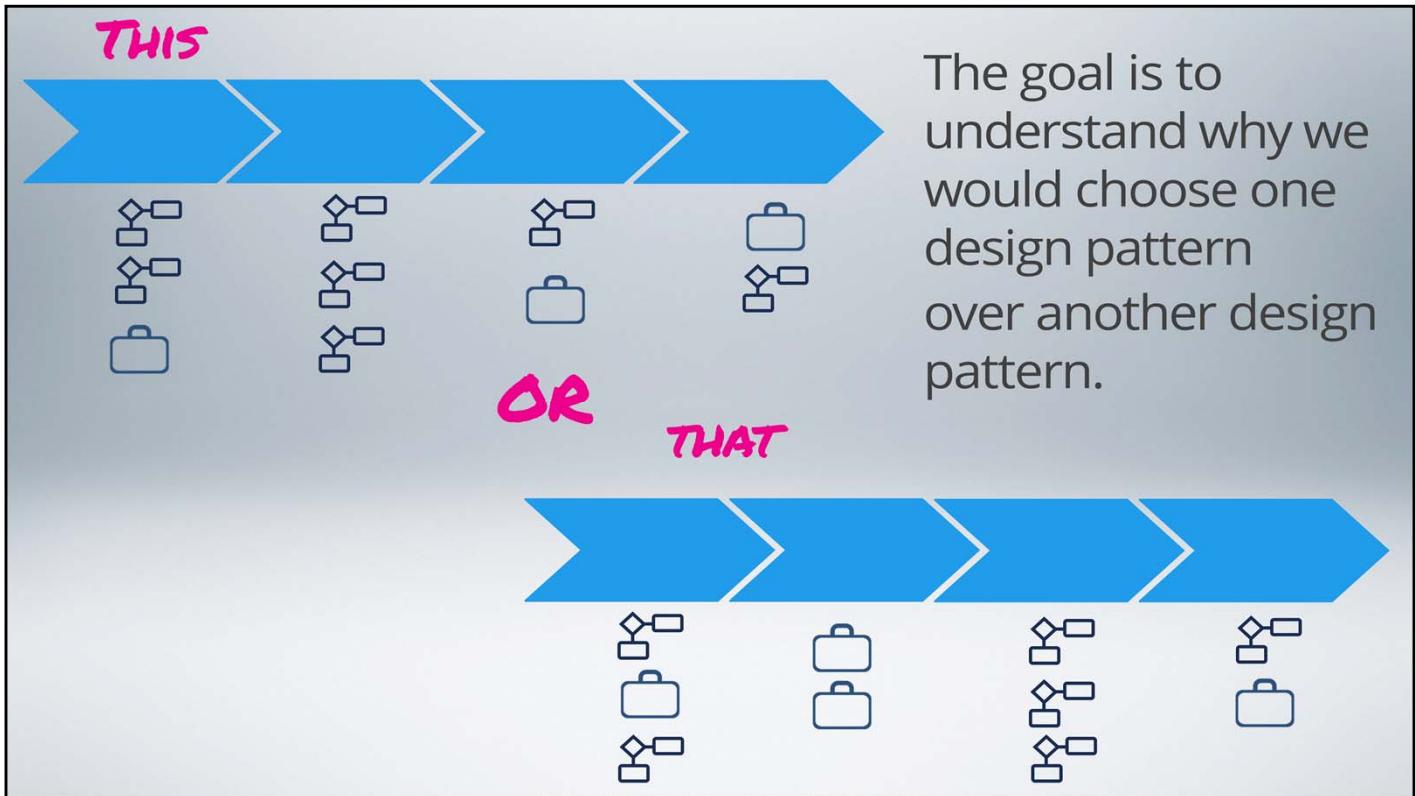
Feedback

We do not want to give away the design patterns just yet, so we will provide a four step map of the Onboarding case we plan to build a solution for in this course. The Onboarding case consists of creating an employee record for the new hire and sending a welcome kit;

then preparing for the employee's arrival which includes scheduling their orientation and enablement training sessions as well as making sure required assets such as hardware and software are provisioned.

During the employee's orientation, we want to automate their enrollment for benefits and setting up their payroll account.

Finally, every new employee goes through a “30-60-90” program, which is designed to monitor, and provide feedback for - their performance during the first 90 days.

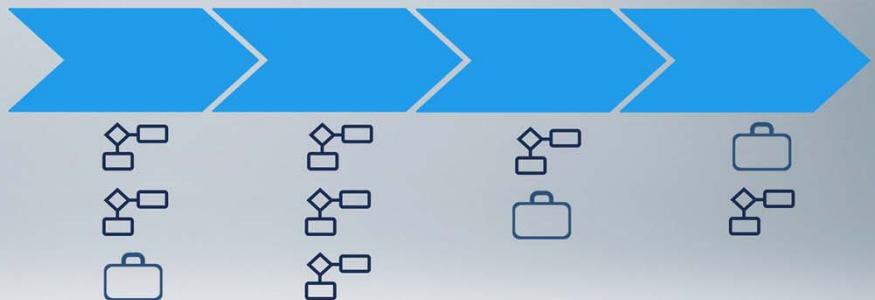


The goal is to understand why we would choose one design pattern over another design pattern.

## Business Problem

We need these things to happen in this sequence

## Pega Solution



## **DESIGN PATTERNS – BEST PRACTICES**

Again, the goal is to understand how to solve business problems using Pega's proven case and process management technologies. With a dedicated focus on Pega's industry-proven design patterns and best practices. Let's go to work.

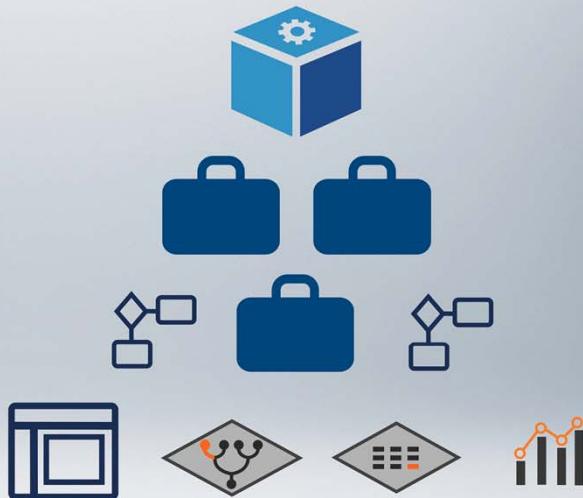
# The Building Blocks of a PRPC Application

In order to build applications, we need to examine the artifacts that create an application. We also need to understand the differences between these artifacts in order to build successful applications.

At the end of this lesson, you should be able to:

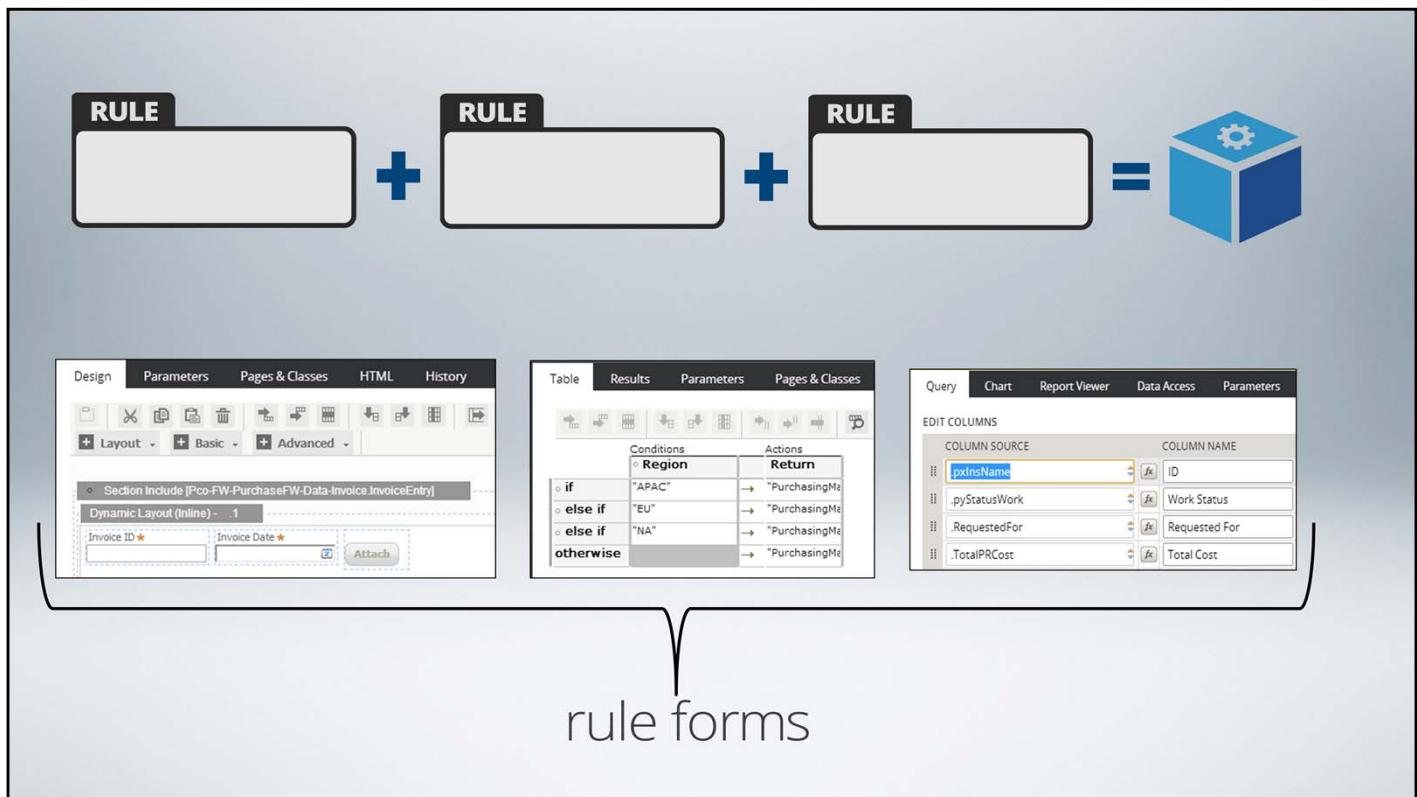
- Define the relationship between an application, case types and processes
- State the purpose of a rule
- Explain the relationship between rules and classes

Application  
Case Types  
Subcases + Processes  
Rules



## Rules define and configure applications

As with any application development the first thought is typically: What do I need to build? Often you will be tasked to either build an application or extend an existing application. An application consists of one or more case types which define what the application will do. Each case type consists of one or more subcases or processes. Processes define how the work defined for the case or subcase is completed. Processes are broken down into assignments that users need to complete and business logic to determine what assignment should be given to a user. Processes will require user interface screens, decision logic, and reporting components to define how work should be completed. These components are implemented using various rules such as sections, decision trees, decision tables, report definitions. Rules are how we define and configure our applications.



So what are rules? Rules are the building blocks of applications. There are many different types of rules and each rule type provides business friendly and easy to use forms that allow us to build our application. We use these forms to define the elements of our application such as our data model, user interface, business logic, and process. We refer to these forms as rule forms.

Rules can be of the following categories:

**Data Model**

**UI**

**Decision**

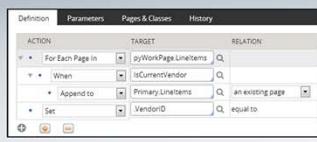
**Process**

Conditions	Actions
if "APAC"	→ "PurchasingMa"
else if "EU"	→ "PurchasingMa"
else if "NA"	→ "PurchasingMa"
otherwise	→ "PurchasingMa"

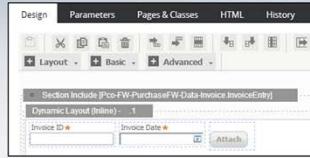
Rules are classified or organized into rule categories. A rule category is a logical grouping of related rule types. For example, in the Decision rule type category we would expect to find rule types for decision tables, decision trees and when rules. Each rule type has a corresponding rule form that allows us to define the specific elements of our application. For example a flow action rule form allows you to drag shapes on it to design the process.

## Rules can be of the following categories:

**Category**  
**Data Model**



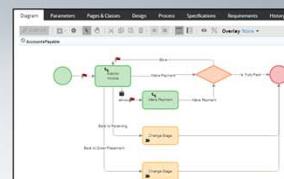
**Category**  
**UI**



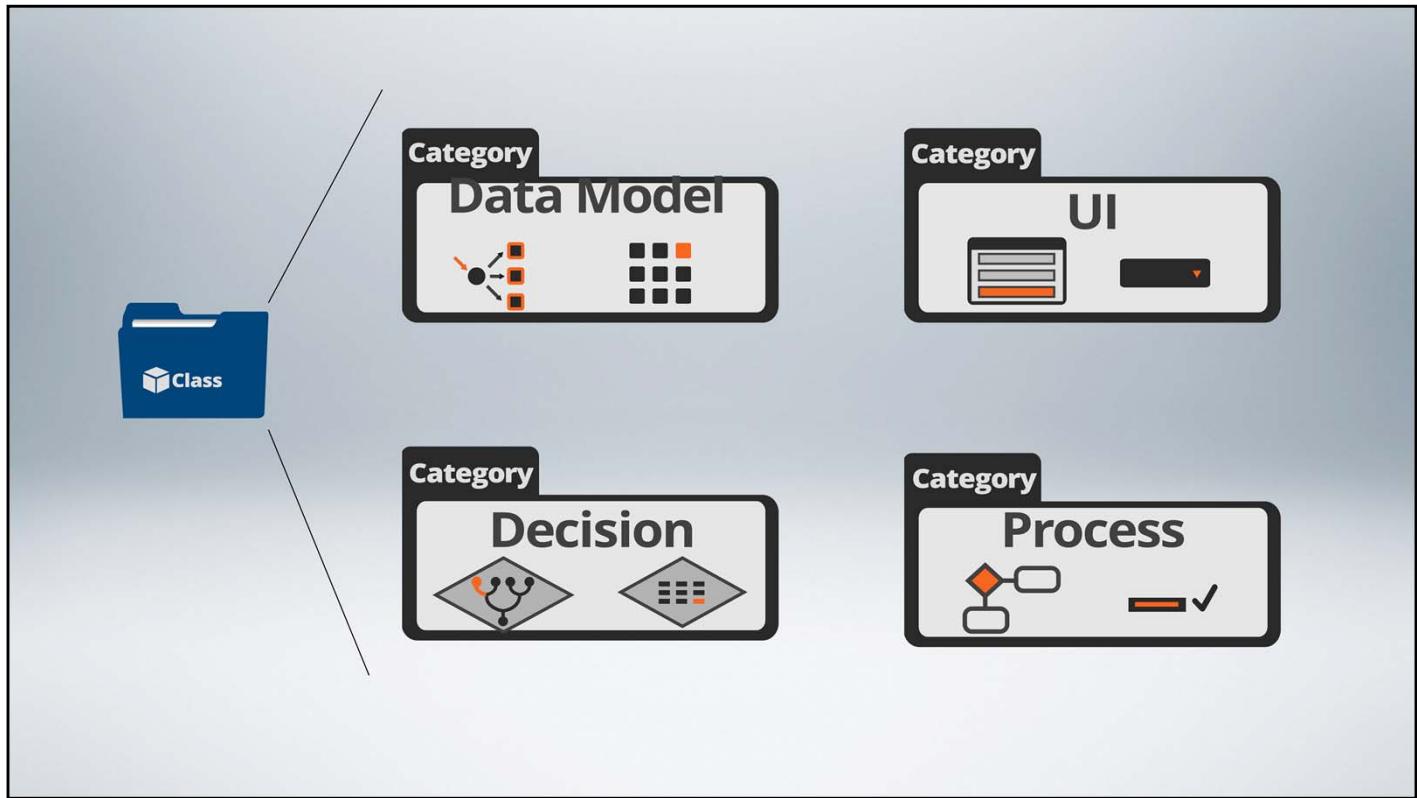
**Category**  
**Decision**

Table	
Conditions	Actions
"APAC"	→ "PurchasingM"
"EU"	→ "PurchasingM"
"NA"	→ "PurchasingM"
otherwise	→ "PurchasingM"

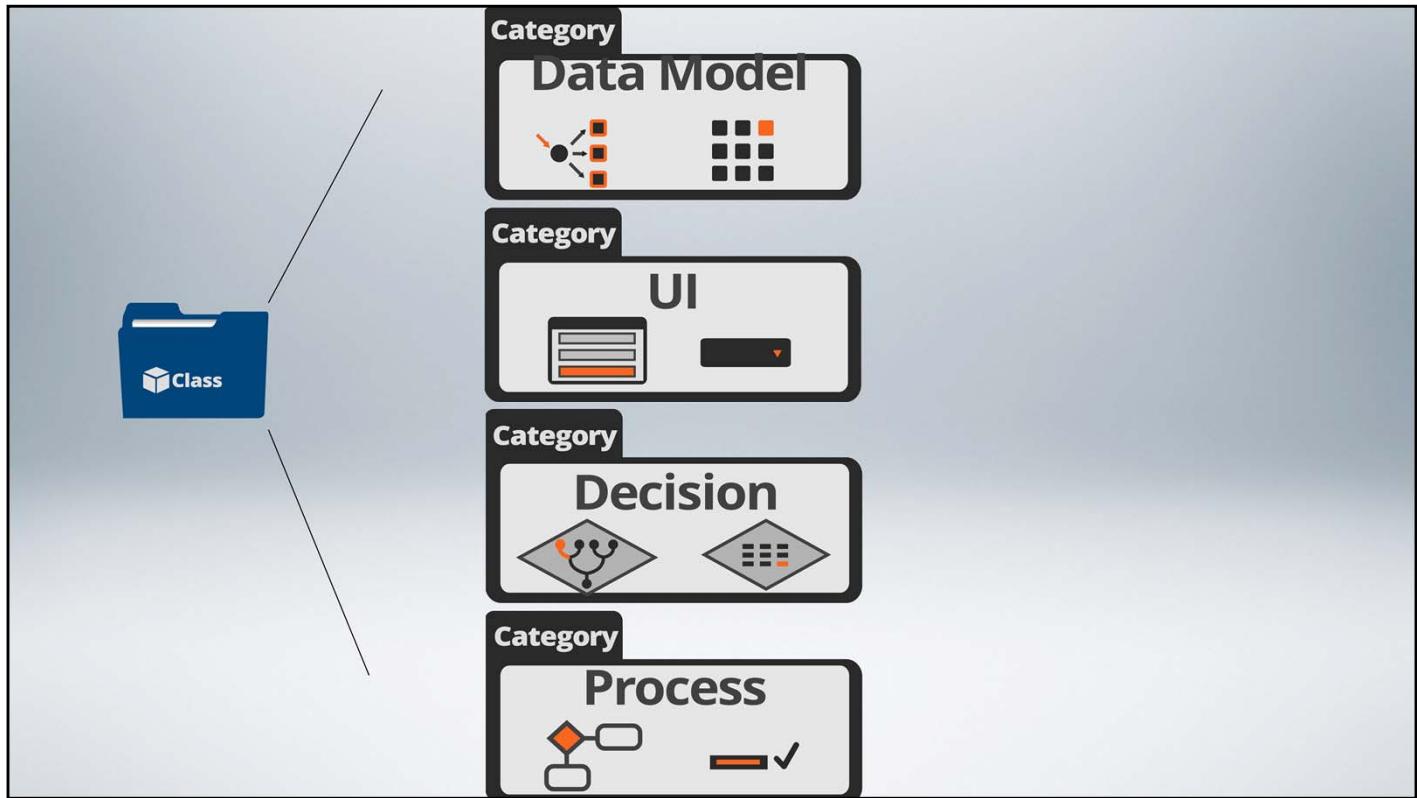
**Category**  
**Process**



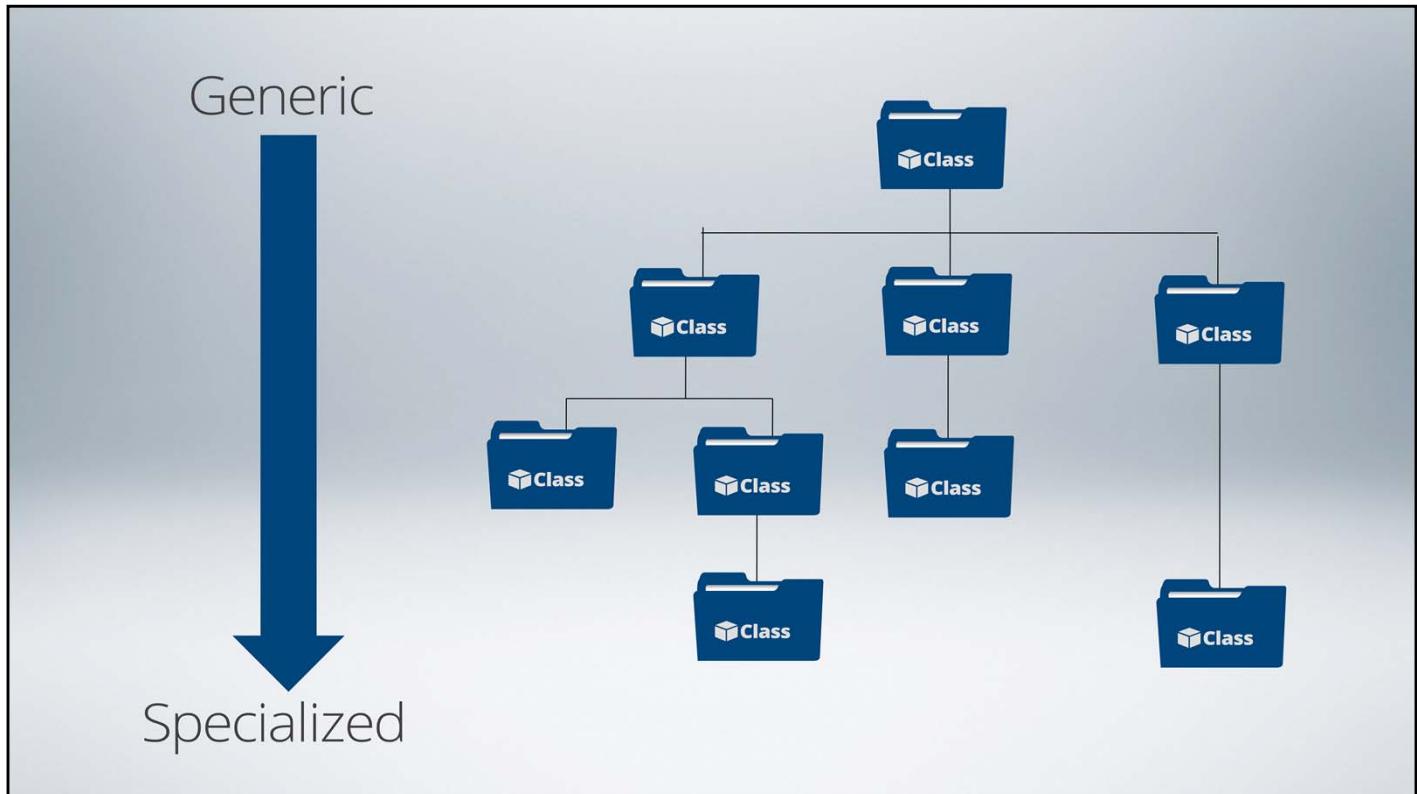
Rules are classified or organized into rule categories. A rule category is a logical grouping of related rule types. For example, in the Decision rule type category we would expect to find rule types for decision tables, decision trees and when rules. Each rule type has a corresponding rule form that allows us to define the specific elements of our application. For example a flow action rule form allows you to drag shapes on it to design the process.



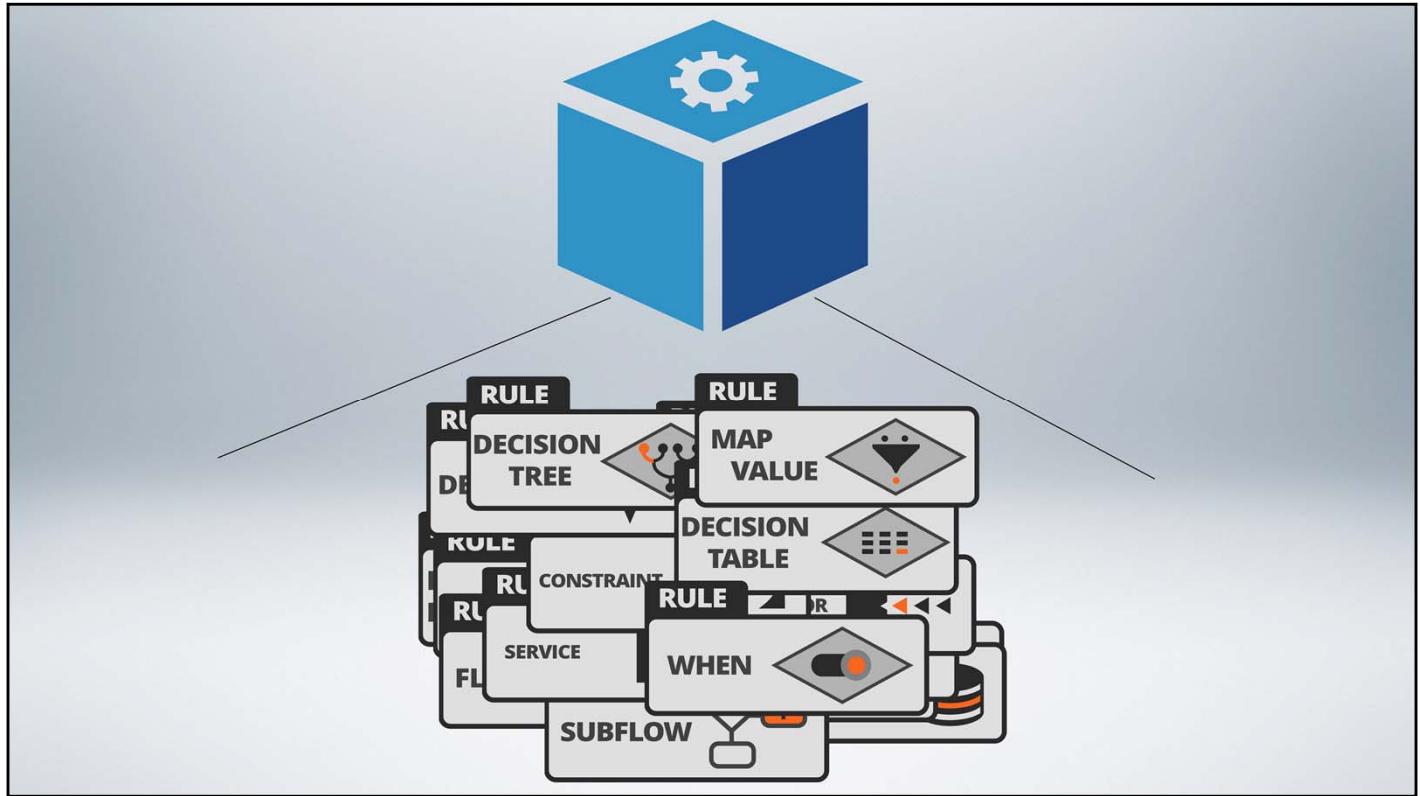
We've discussed that we have rule categories and that rule categories contain rules, but how do we organize the rule categories? A class is a container for rules, it represents the applicability of a rule, which we often refer to as the scope of the rule. A class defines capabilities, rules such as Properties, Flows, and Sections, that are available to the class and its subordinate classes.



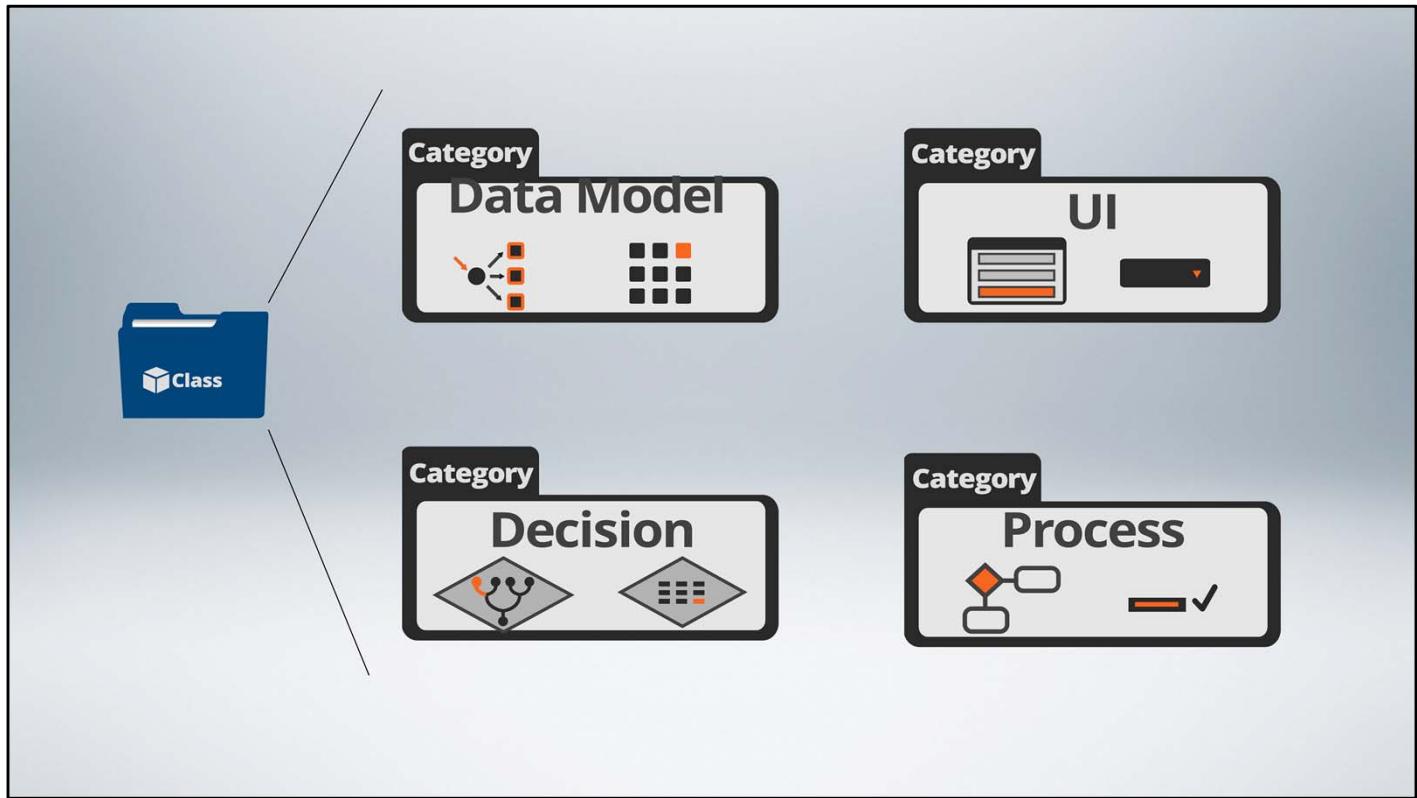
We've discussed that we have rule categories and that rule categories contain rules, but how do we organize the rule categories? A class is a container for rules, it represents the applicability of a rule, which we often refer to as the scope of the rule. A class defines capabilities, rules such as Properties, Flows, and Sections, that are available to the class and its subordinate classes.



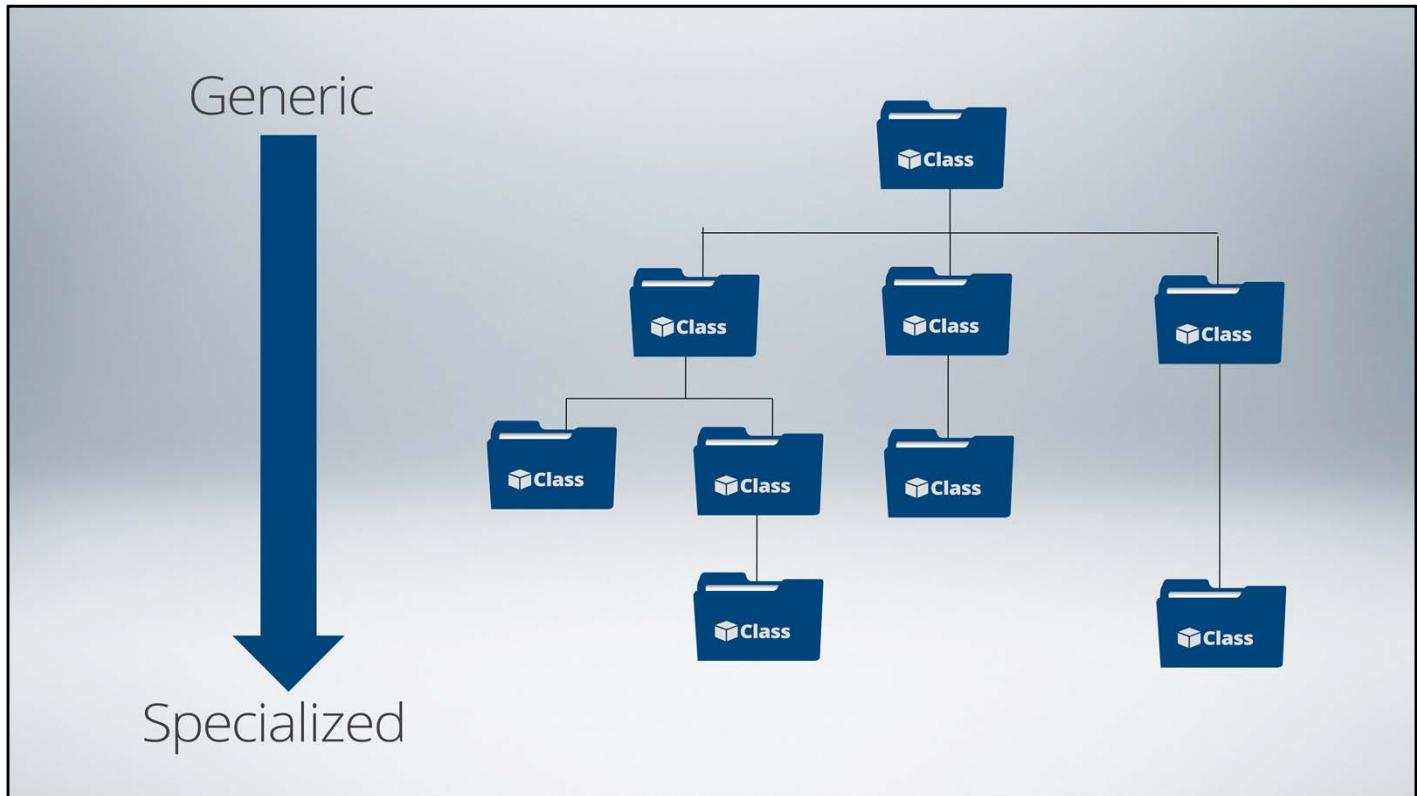
An application is organized as a hierarchy of classes based on the scope of their reusability. This hierarchy is referred to as a class structure and it governs how individual rules are applied. As you move down the class structure you go from having more generic rules at the top of the hierarchy to more specialized rules at the bottom of the hierarchy. By having more generic rules in the top of the hierarchy we create components that are easier to reuse in our application.



To recap, an application is a collection of rules.



These rules are then organized into classes.



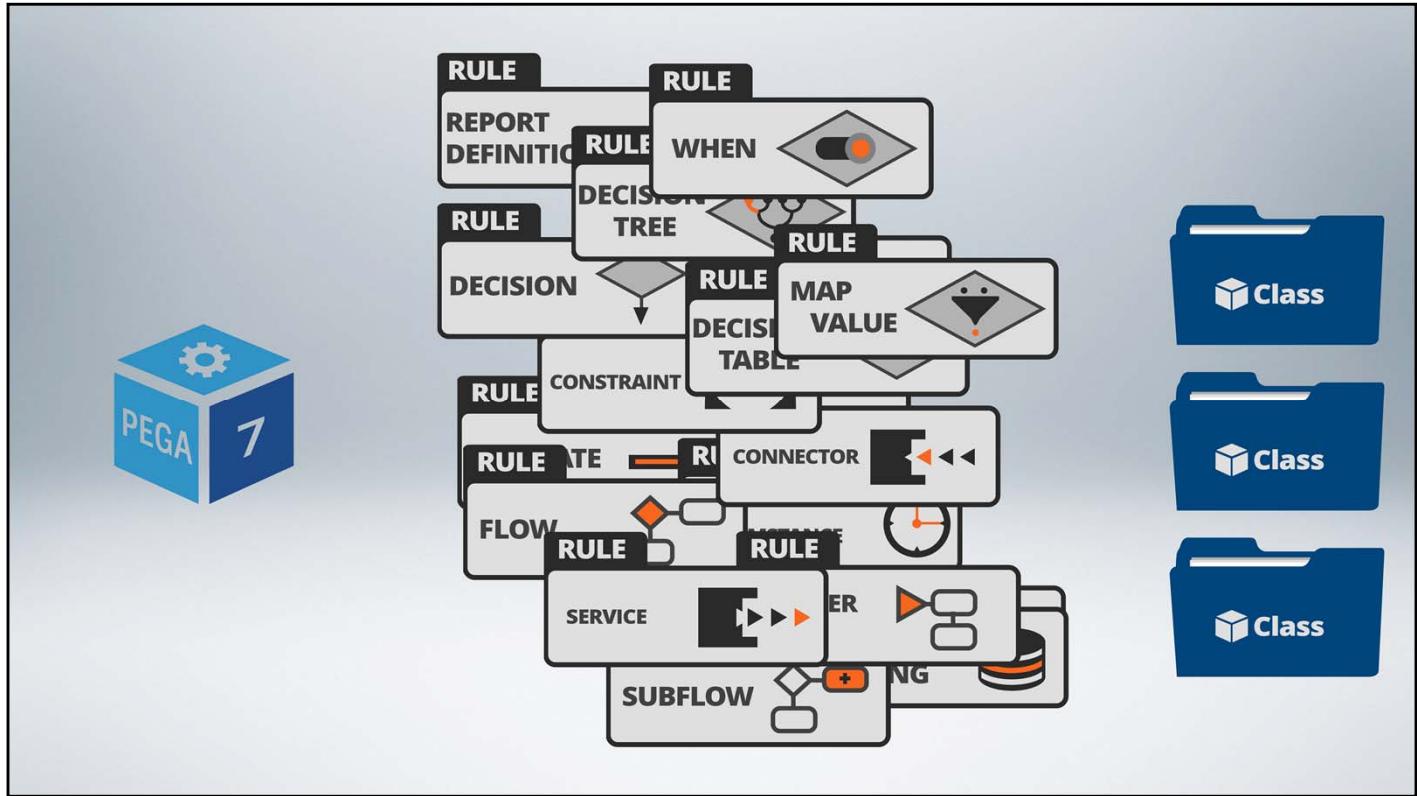
The classes form a hierarchy known as the class structure that has more generic rules at the top of the structure and more specialized rules at the bottom of the structure.

# Managing the Building Blocks of a PRPC Application Reference

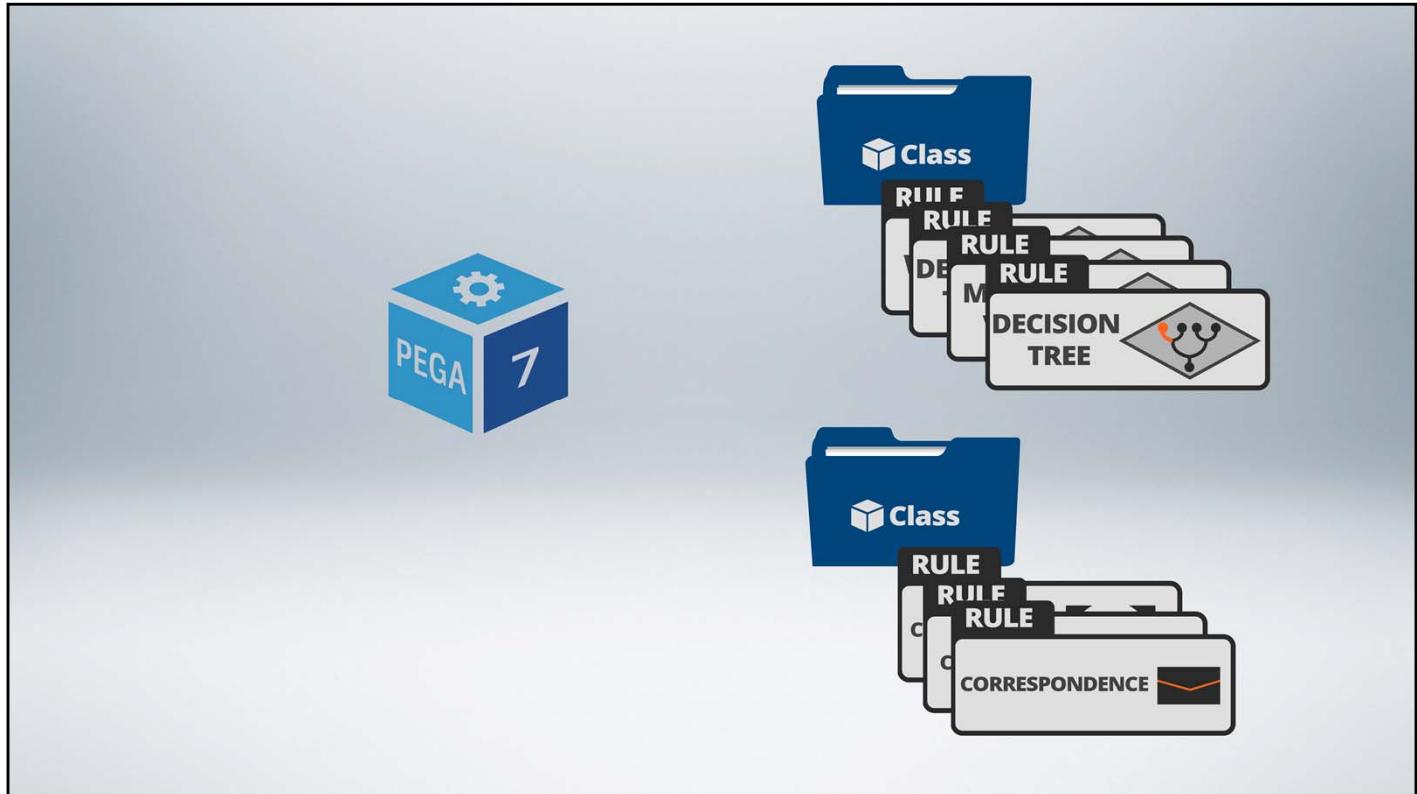
In this lesson we discuss how to group your classes together to have a deployable unit for an application. We also discuss rule inheritance and how that is critical in reusing parts or all of an existing application.

At the end of this lesson, you should be able to:

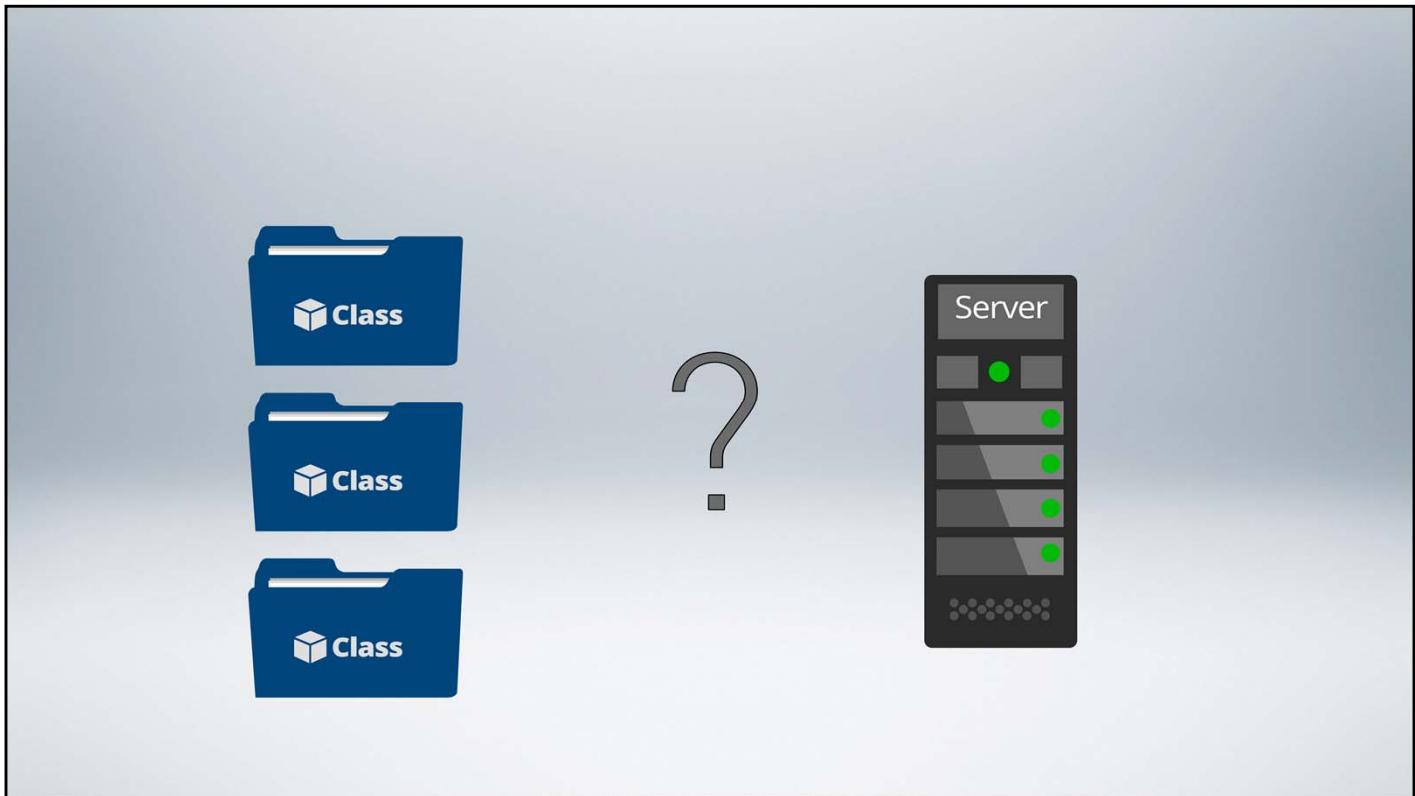
- Understand RuleSets
- Understand class inheritance and rule resolution
- Use the Application Explorer to view a class structure



Remember that a PRPC application is a collection of rules.



We then organize those rules into classes.



But how do we organize those classes and deploy them to a server?

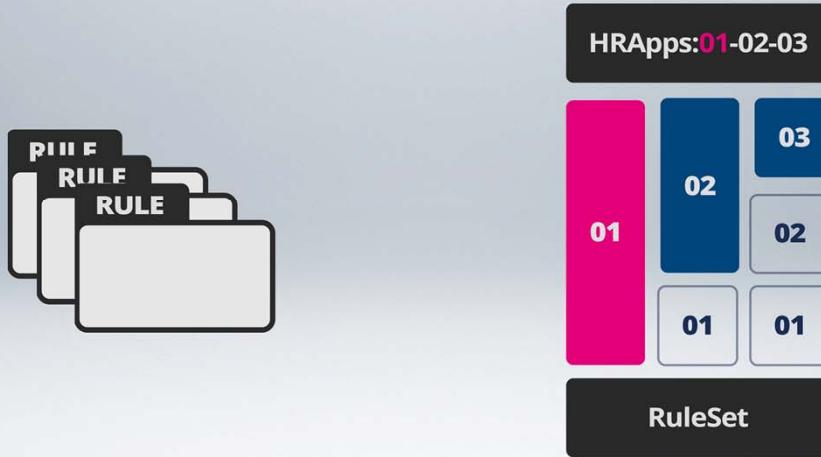
## RuleSet: a deployment unit comprised of rules



A RuleSet is a deployment unit for a “set” or group of rules. We create a RuleSet to identify, store, and manage a set of rules that define an application or a major portion of an application. Applications are really just stacks of rules applied in a specific order.

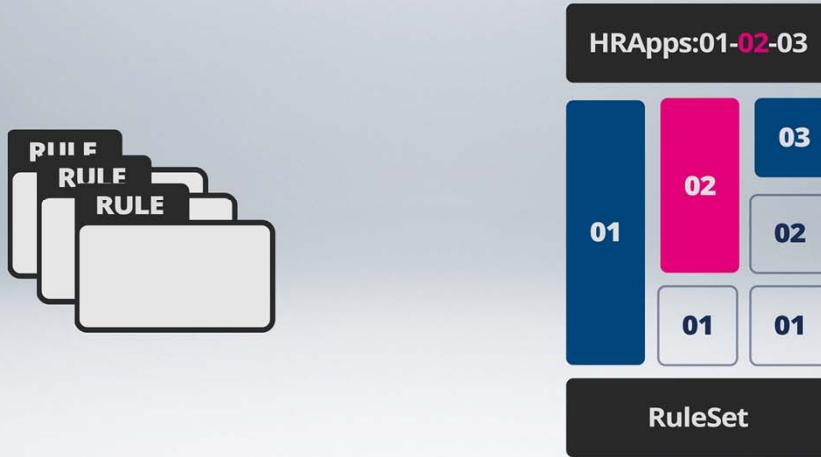
Every RuleSet has a name and a version. It is a best practice to give our RuleSet a name that helps us manage and identify its contents. The name must start with a letter and contain only letters, digits, and dashes. Typically, a RuleSet name represents the application content or purpose, and correlates to the classes to which the rules will apply.

## RuleSet: a deployment unit comprised of rules



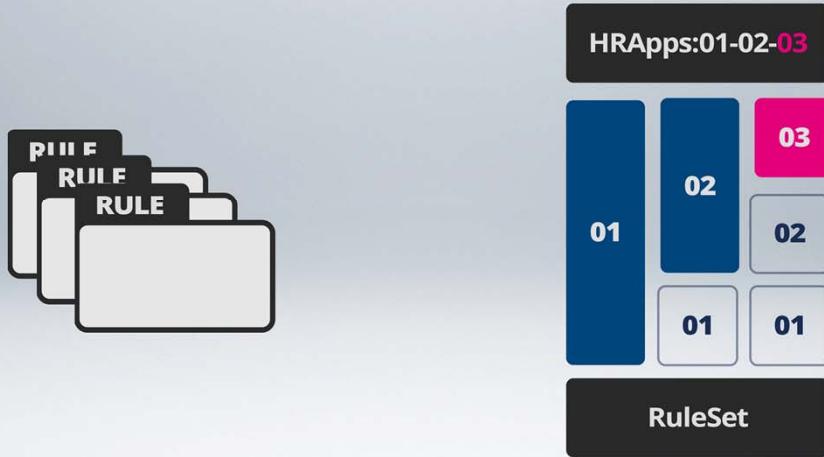
RuleSet versions consist of three two-digit numbers separated by dashes such as 01-02-03. The first two digits represent the Major Version; the initial or subsequent substantial release of an application. A major version change encompasses extensive changes to functionality.

## RuleSet: a deployment unit comprised of rules

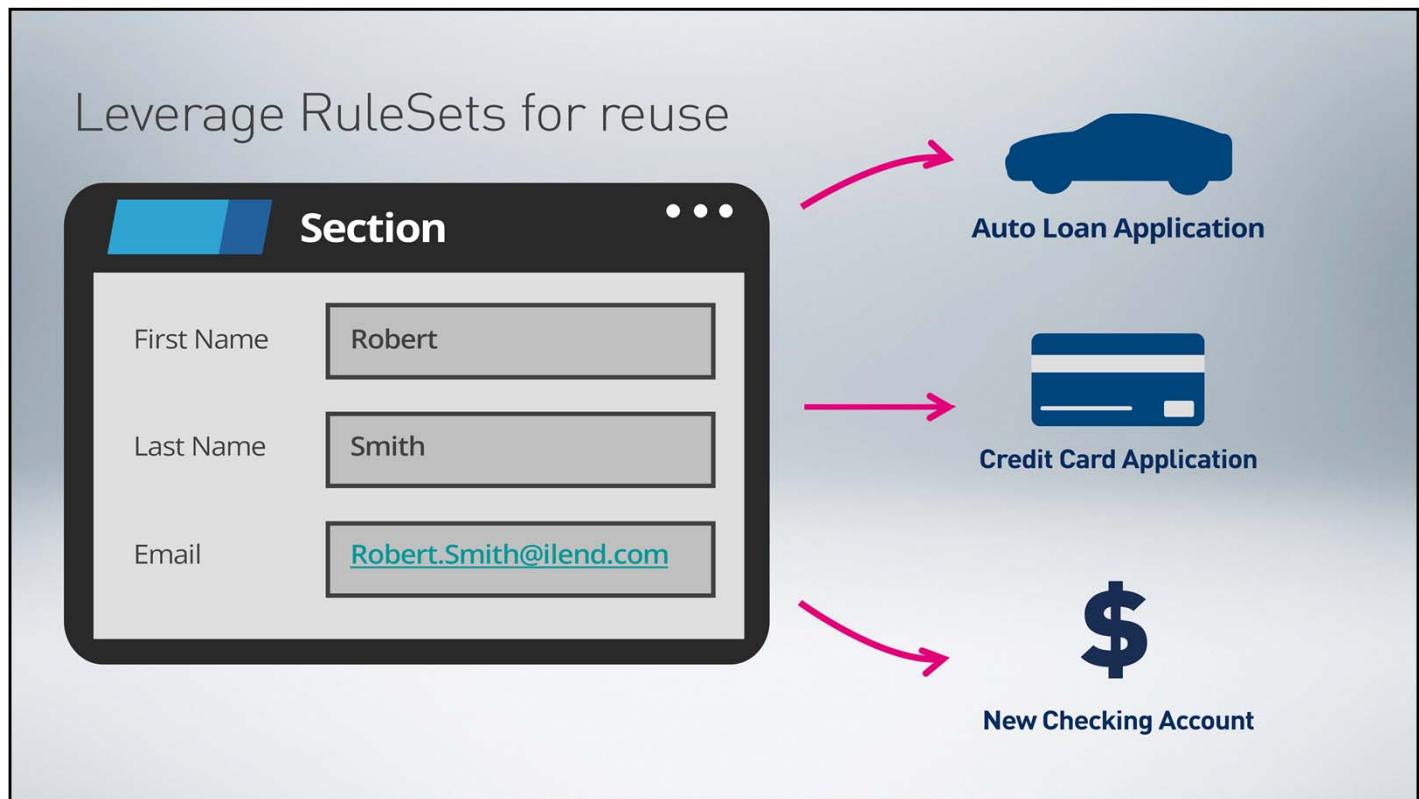


The second two digits represent the Minor Version. We increment these digits to represent an interim release or enhancements to our major release.

## RuleSet: a deployment unit comprised of rules



The third two digits represent a Patch Version which is also an interim release typically used for bug fixes.



Inside of a RuleSet is our application, which implements a business solution. When we implement a business solution in PRPC, we want to reuse as many elements of the application as possible. Perhaps they were already developed and tested in a previous version, and we know they work. Or, we need to use a bit of functionality – a process, a UI form, or an element of data – with two different types of work.

For example, we can use a section used to collect customer information for multiple work types – requesting an auto loan, applying for a credit card, or opening a checking account.

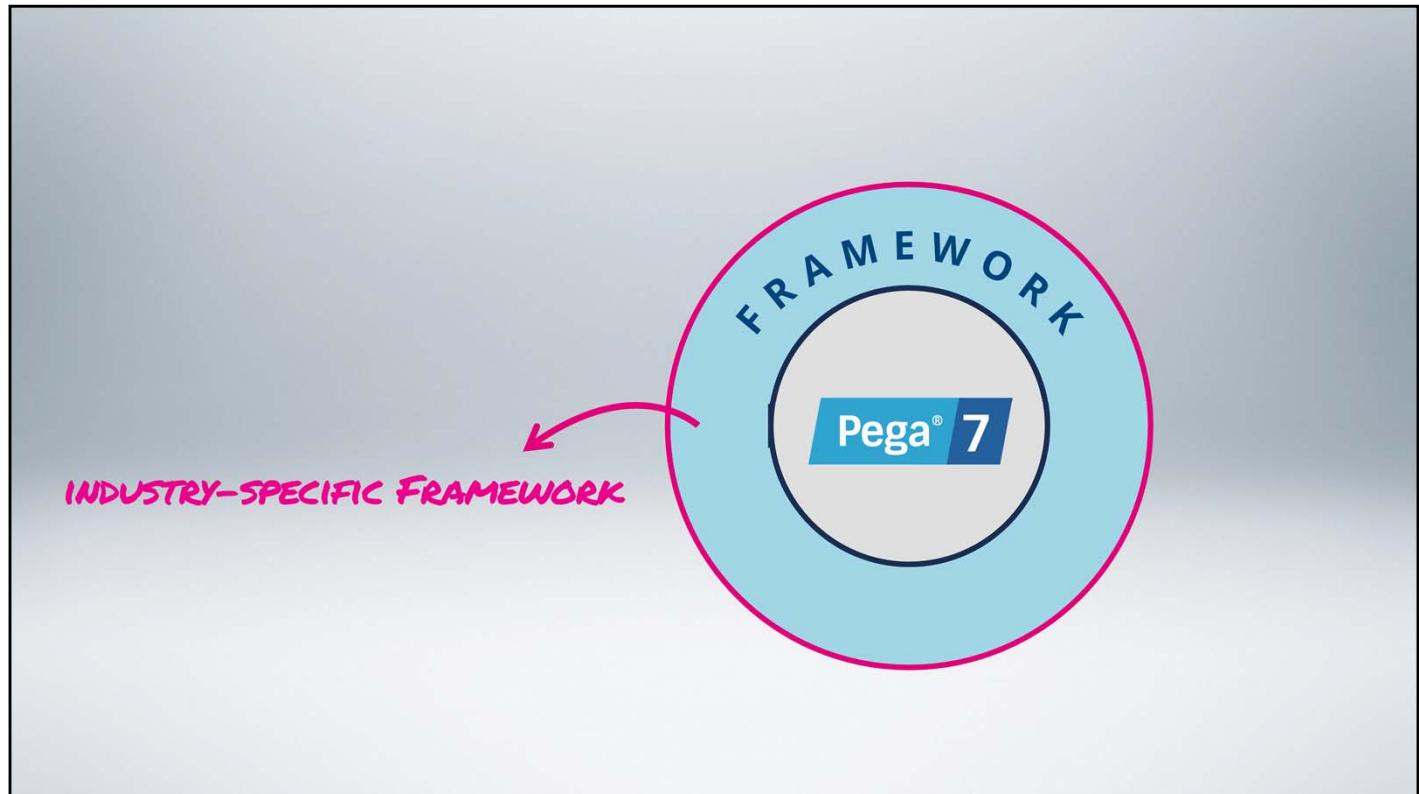
# Re-use and Inheritance

The process of using a feature that has already been developed elsewhere

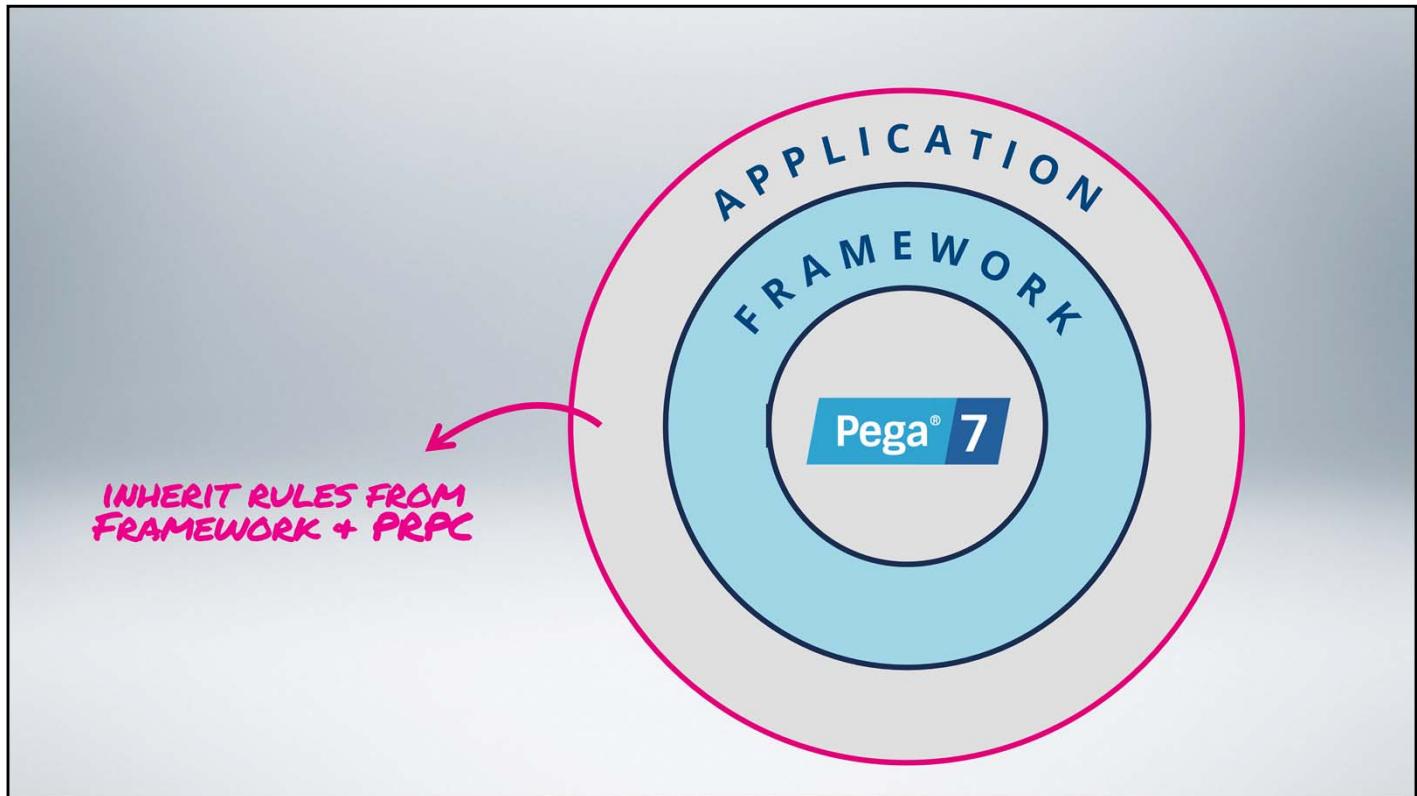


We maximize the value of the rules we create by utilizing the concept of inheritance. Simply put, inheritance is the process of using – or inheriting – a feature that's already been developed elsewhere. Let's look into how inheritance helps us to quickly develop reliable, robust applications.

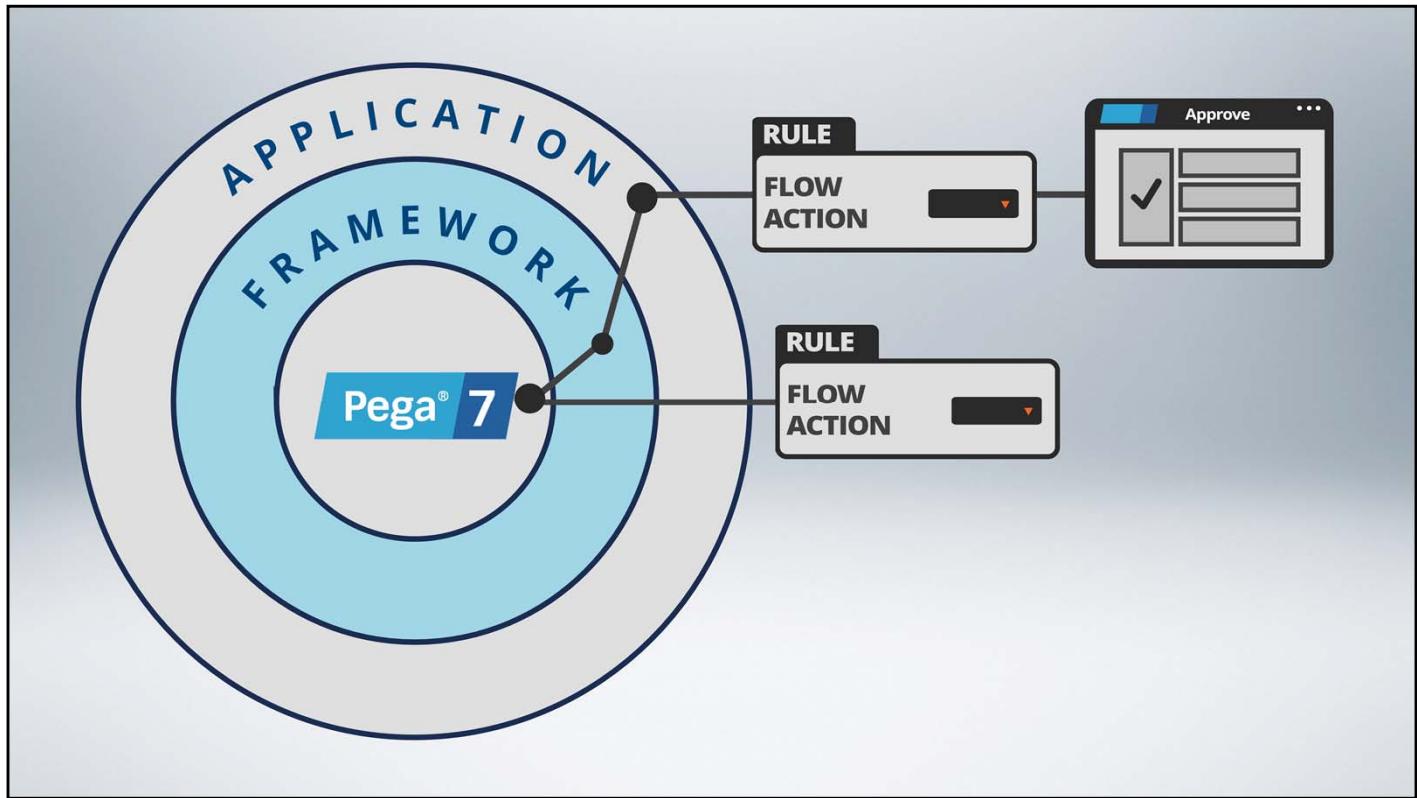
When we sit down with PRPC to begin developing an application, we start with a set of rules provided as part of the installation. The standard PRPC classes represent the most general, reusable rules.



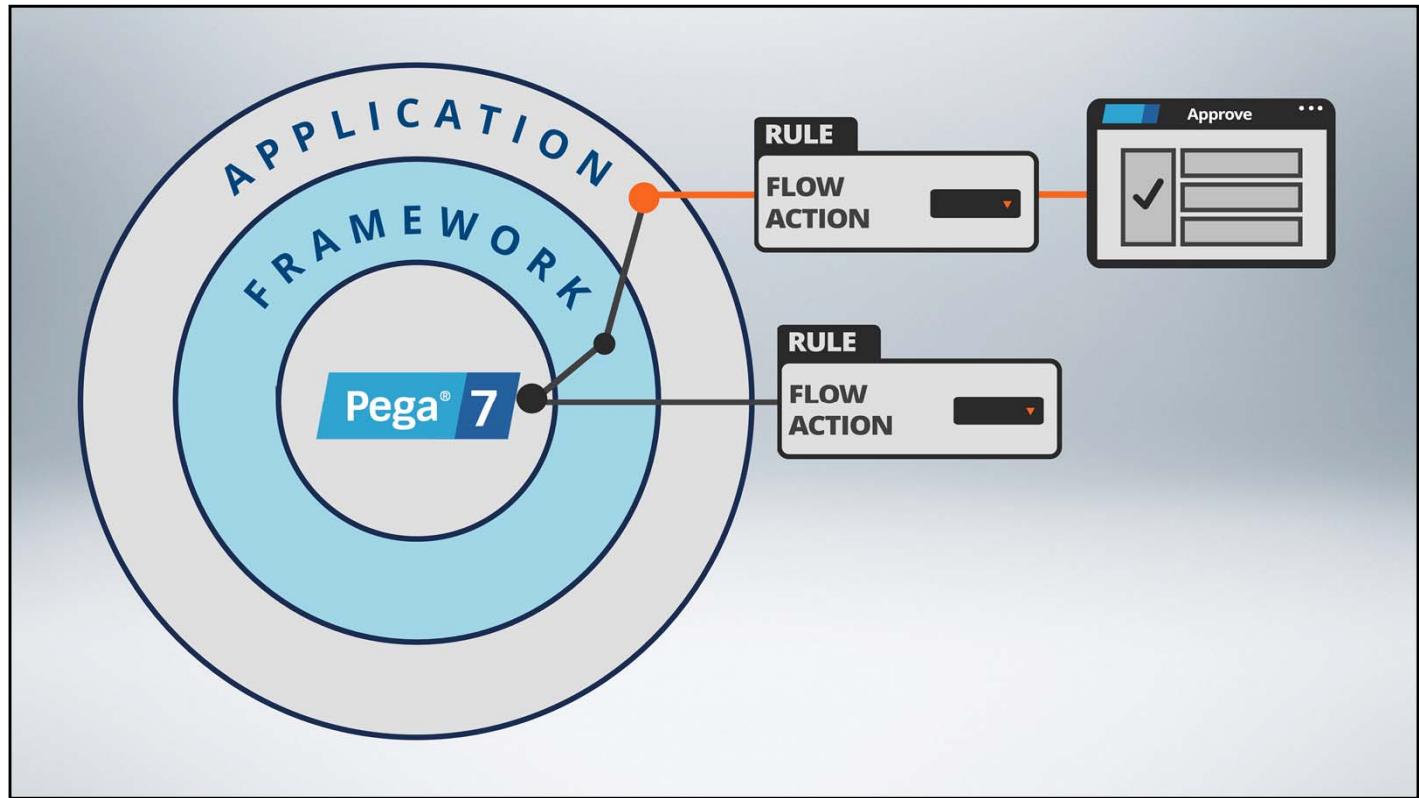
If we purchased an industry-specific framework – for example Customer Process Manager (CPM) for Insurance - we would then install the framework on top of PRPC. Now, we wouldn't want the framework to re-implement rules that were provided by PRPC unless it was necessary to do so. Rather, we would want the framework to inherit those PRPC rules.



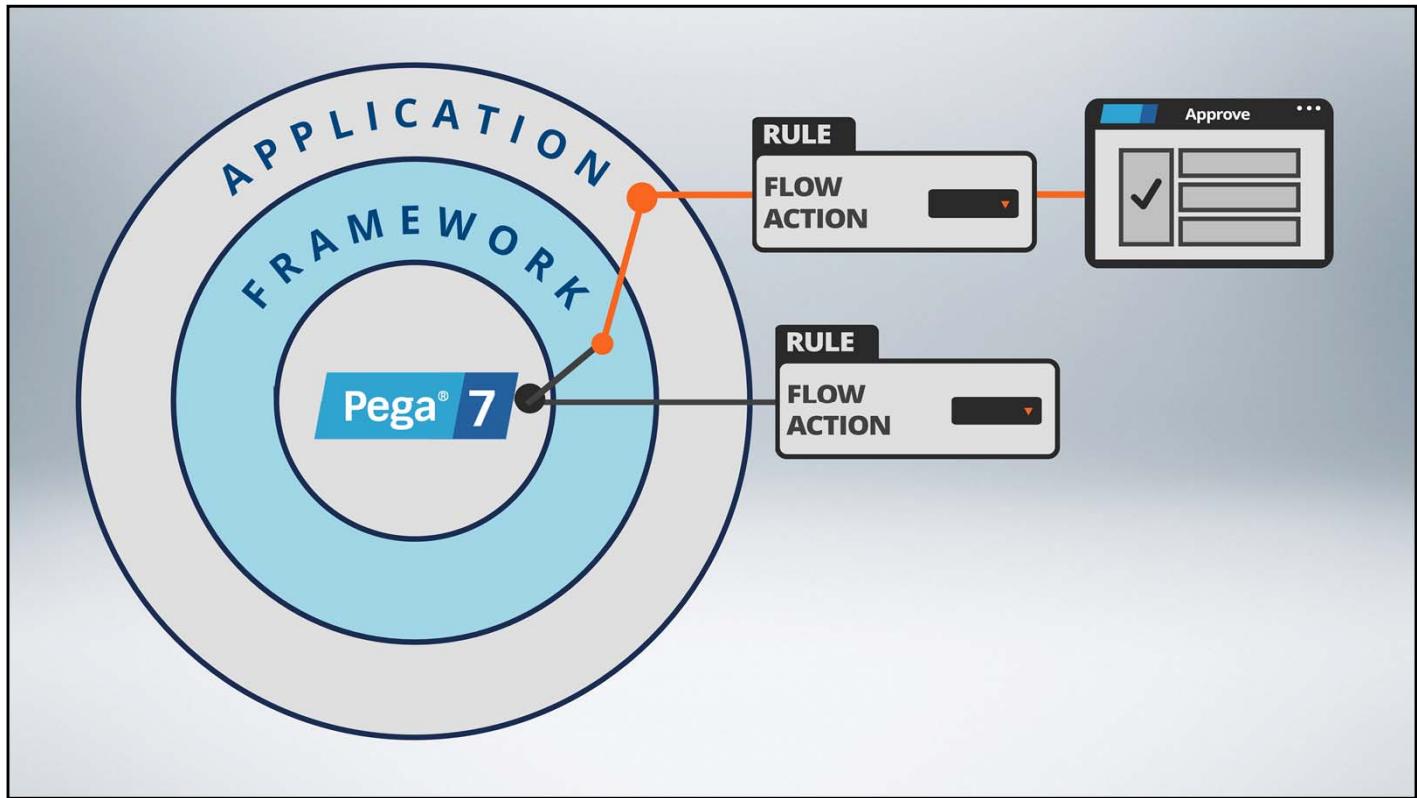
Now, if we're building our application on top of the framework, we'd like to reuse whatever rules the framework provides to us, so we inherit rules from the framework – and, by extension, PRPC too. The application contains the most specialized non reusable rules.



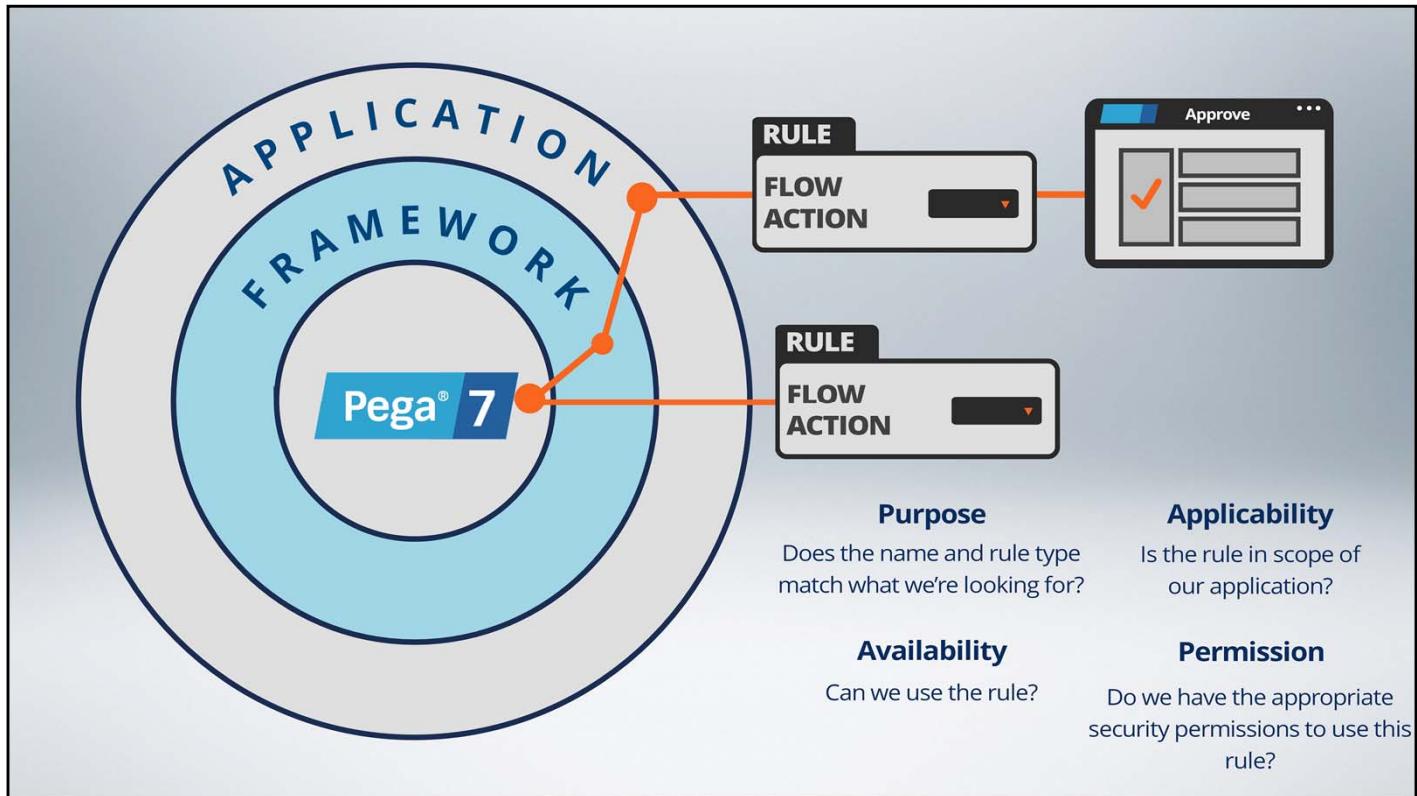
Let's say we need to create a user interface to process an Approve flow action. PRPC provides such a rule for use – and because our application inherits all of the rules provided by PRPC, we can automatically use it. When we run our process, PRPC searches for the rule it needs – it starts with our application, then moves on to the framework, and finally PRPC itself – and once it finds the rule, it uses it.



And if we want to specialize the rule, we can create a version in our application. This version overrides the one in PRPC, so users will see it – rather than the one in PRPC – when they reach the approval step in our process.



Imagine that we've reached the point in our process where a work item needs to be approved. Since our system may contain multiple user interfaces for our Approve action, how does PRPC determine which instance to use? The answer is rule resolution.



Rule resolution is a process that uses a Pega-patented algorithm to determine the appropriate rule to use when a process is run. It selects the appropriate rule, based on factors such as:

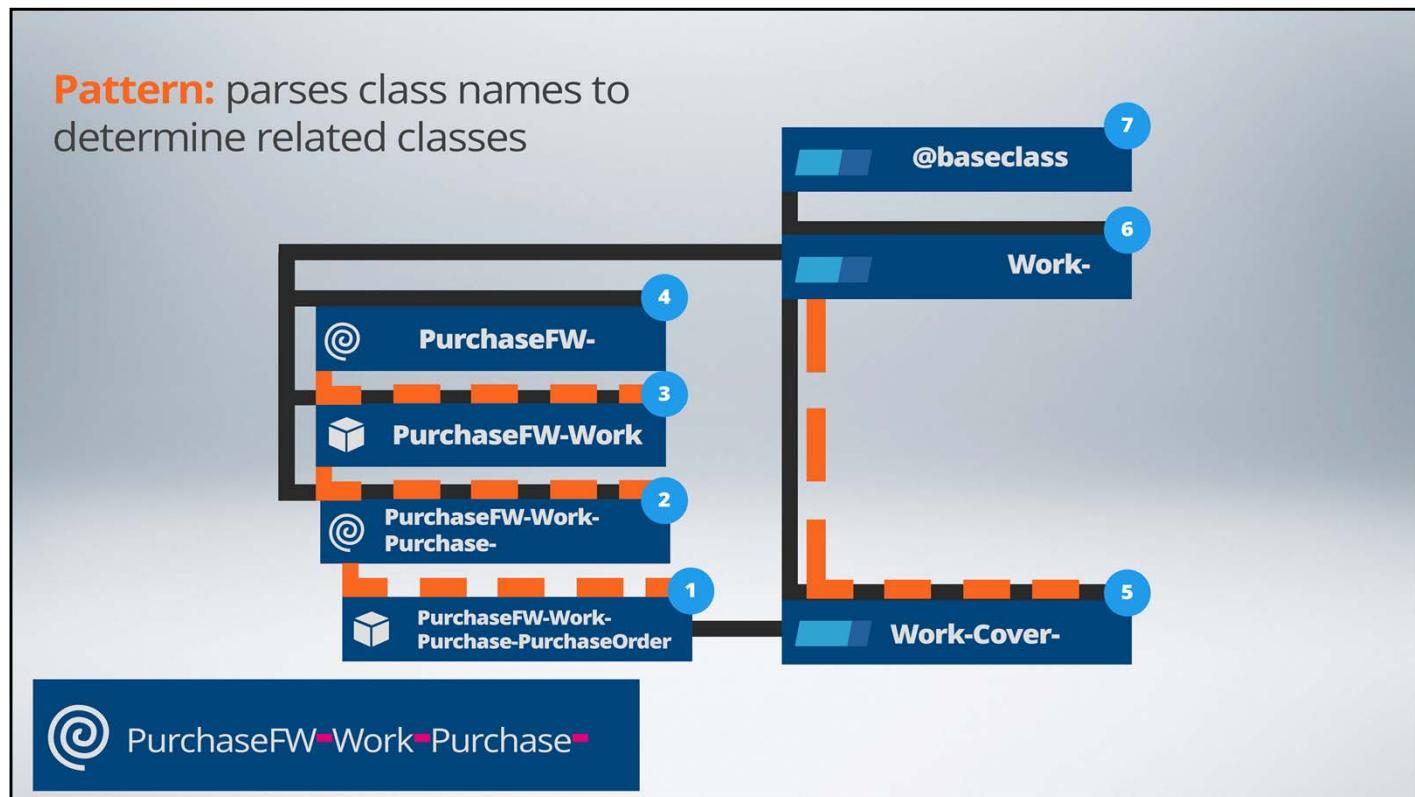
**Purpose** – What are we trying to do? If we need the Approve UI, we don't want PRPC to return a decision or expression named Approve. PRPC checks that the name and type of the rule matches what we're looking for. This factor is where class inheritance comes into play as it helps determine the applicability of a rule.

**Availability** – Can we use the rule? Perhaps the rule is under development, and we don't want PRPC to use it to process work just yet.

**Applicability** – Is the rule in scope – or intended – for our application, or another application? If we're processing loan applications, we wouldn't want an approval UI for a time-off request or a purchase order.

**Permission** – Do we have the appropriate security permissions to use the rule?

Rule resolution considers all of these factors, and then automatically and transparently selects the right rules to use as we run our process. There are some additional steps to the overall rule resolution process for more information see the PDN article [How the system finds rules through rule resolution](#).



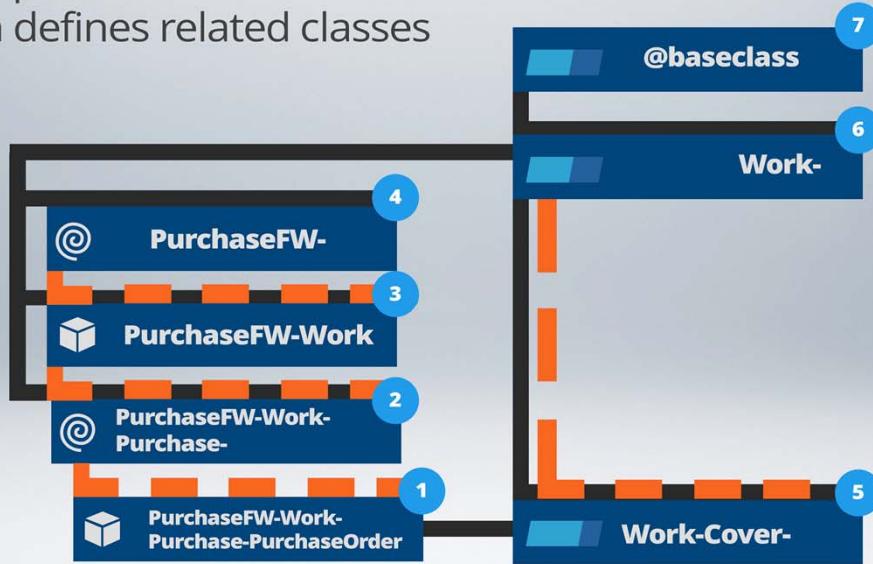
Let's examine inheritance a little more to better understand the Purpose stage in the rule resolution process. PRPC uses a dual-inheritance model. Dual inheritance is conceptually similar to single inheritance – which is common to many programming languages – except that it involves *two* pathways to inherit rules.

The first type of inheritance in PRPC is pattern inheritance.

Pattern inheritance is optional, and involves parsing class names.

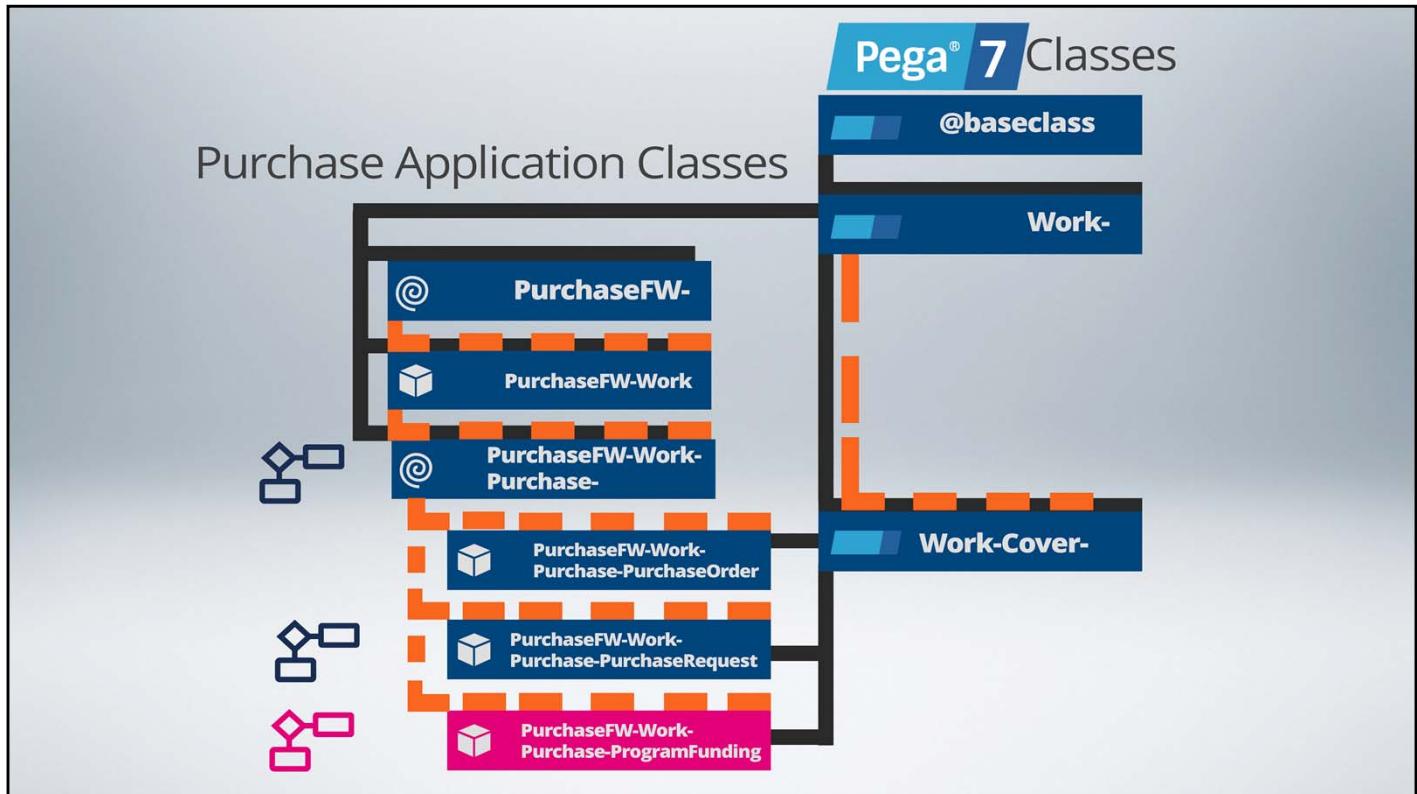
In PRPC, the hyphen character delimits class names; to determine the parent of a particular class using pattern inheritance, just look at the characters to the left of the right-most hyphen. Pattern inheritance is for sharing the objects in your application hierarchy.

**Directed:** specified in the rule form which defines related classes



The second type of inheritance in PRPC is directed inheritance.

Directed inheritance allows us to access rules in standard PRPC classes, such as Work-, Work-Cover-, and @baseclass. The directed-inheritance parent class must be specified on the rule form for a class instance. Directed inheritance is for utilizing the objects provided by standard packaged classes.



PRPC determines the inheritance independently for each class. As a result, the inheritance pathway may differ significantly for two classes in the same application.

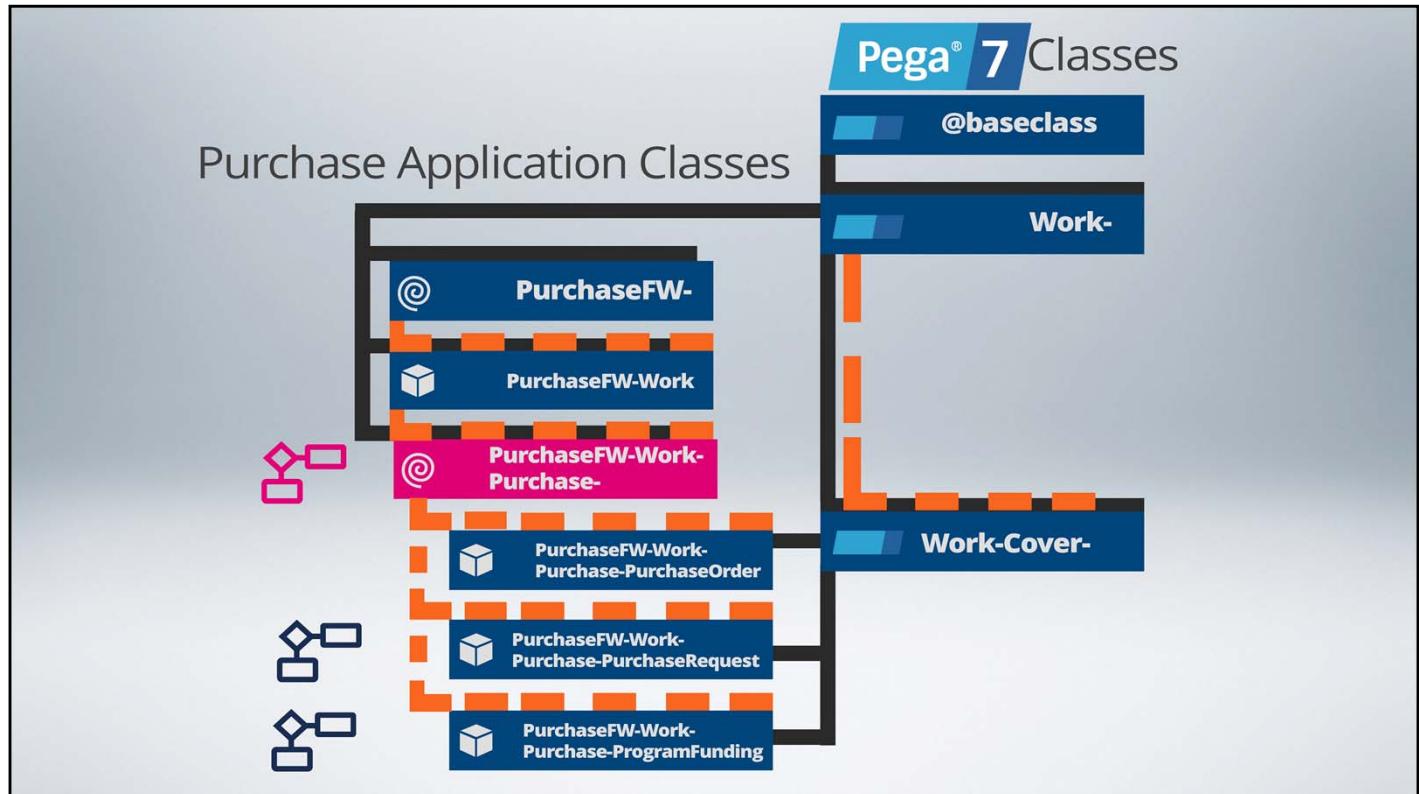
Let's look at an example of the Purpose stage as to how PRPC determines which rules to further evaluate in the rule resolution process.

Let's say that you are in the ProgramFunding class and want to use an Approvals process.

It just so happens that a few of the classes have an Approvals process, how does PRPC determine which potential rules should be evaluated?

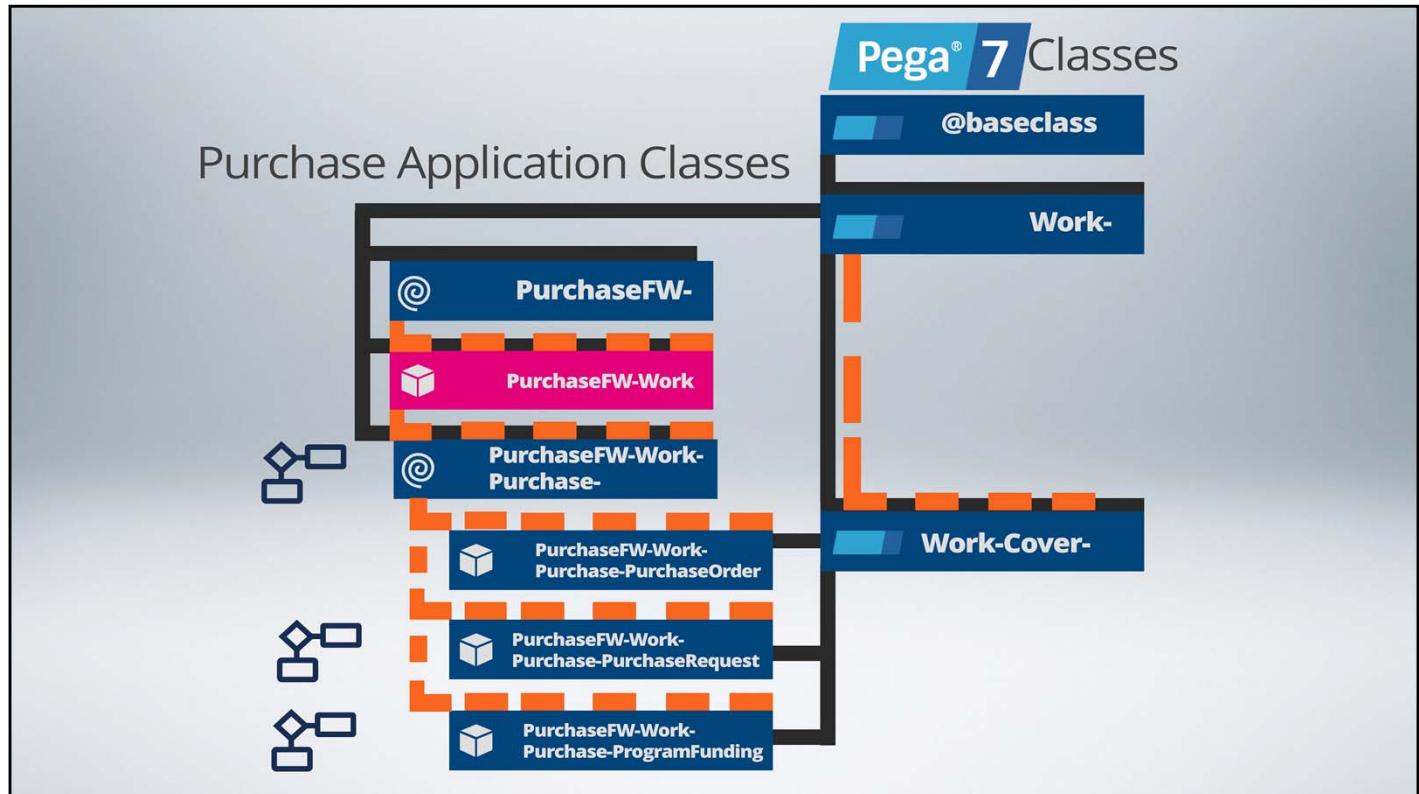
First, PRPC checks the ProgramFunding class to see if it contains an Approvals process, it does so that rule gets added to the list.

What class will PRPC check next?

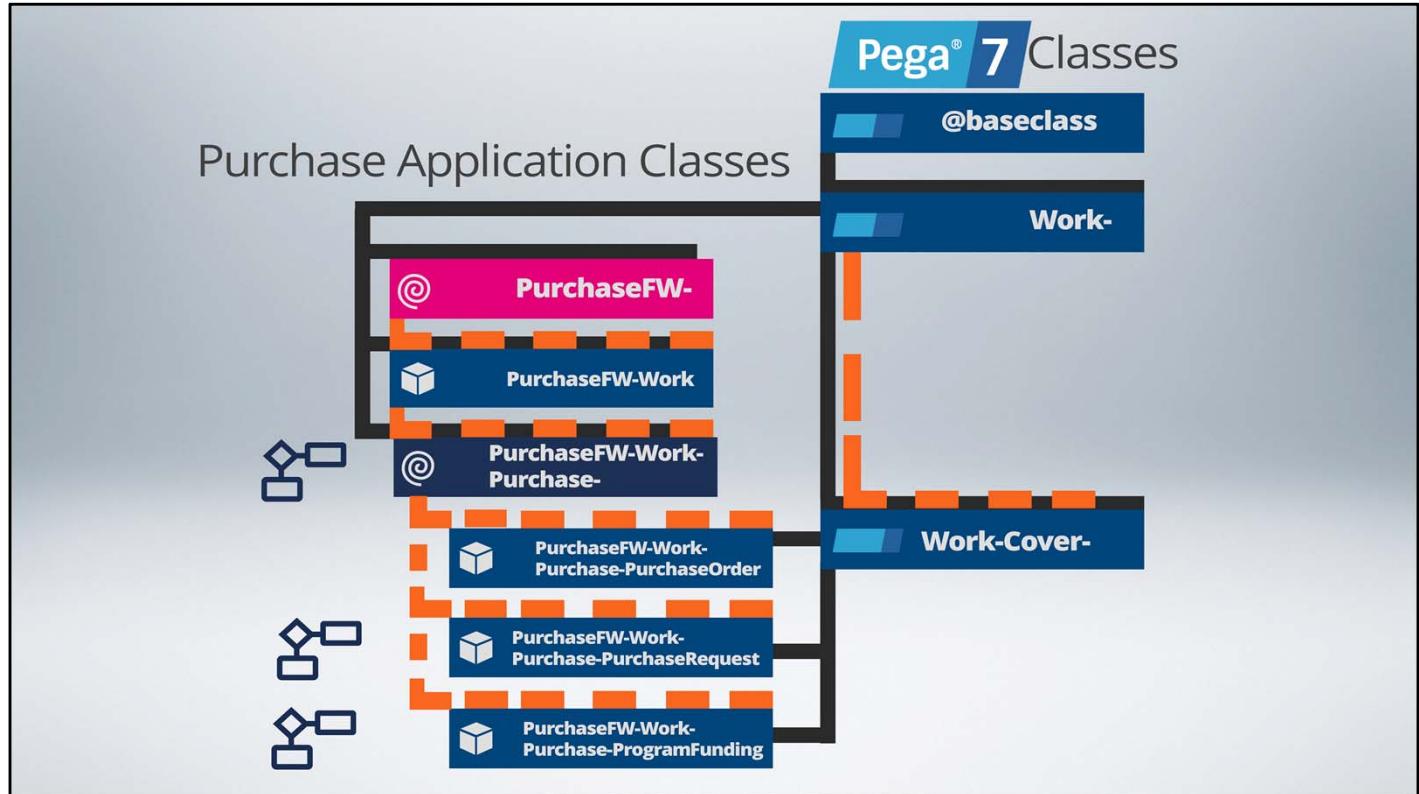


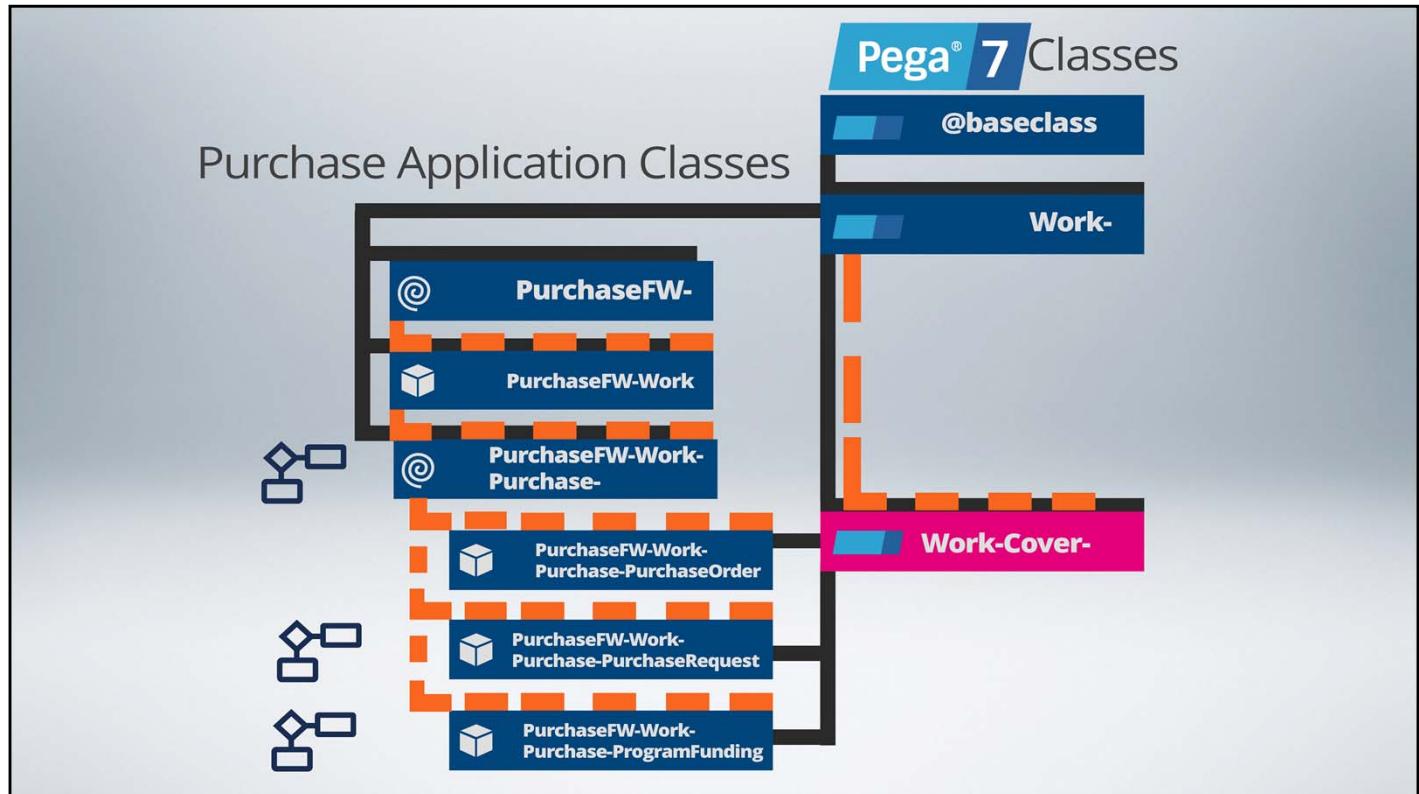
Remember that pattern inheritance comes first, therefore the next class to be checked is the Purchase- class.

It also has an Approvals process contained within it therefore this rule is also added to the list.

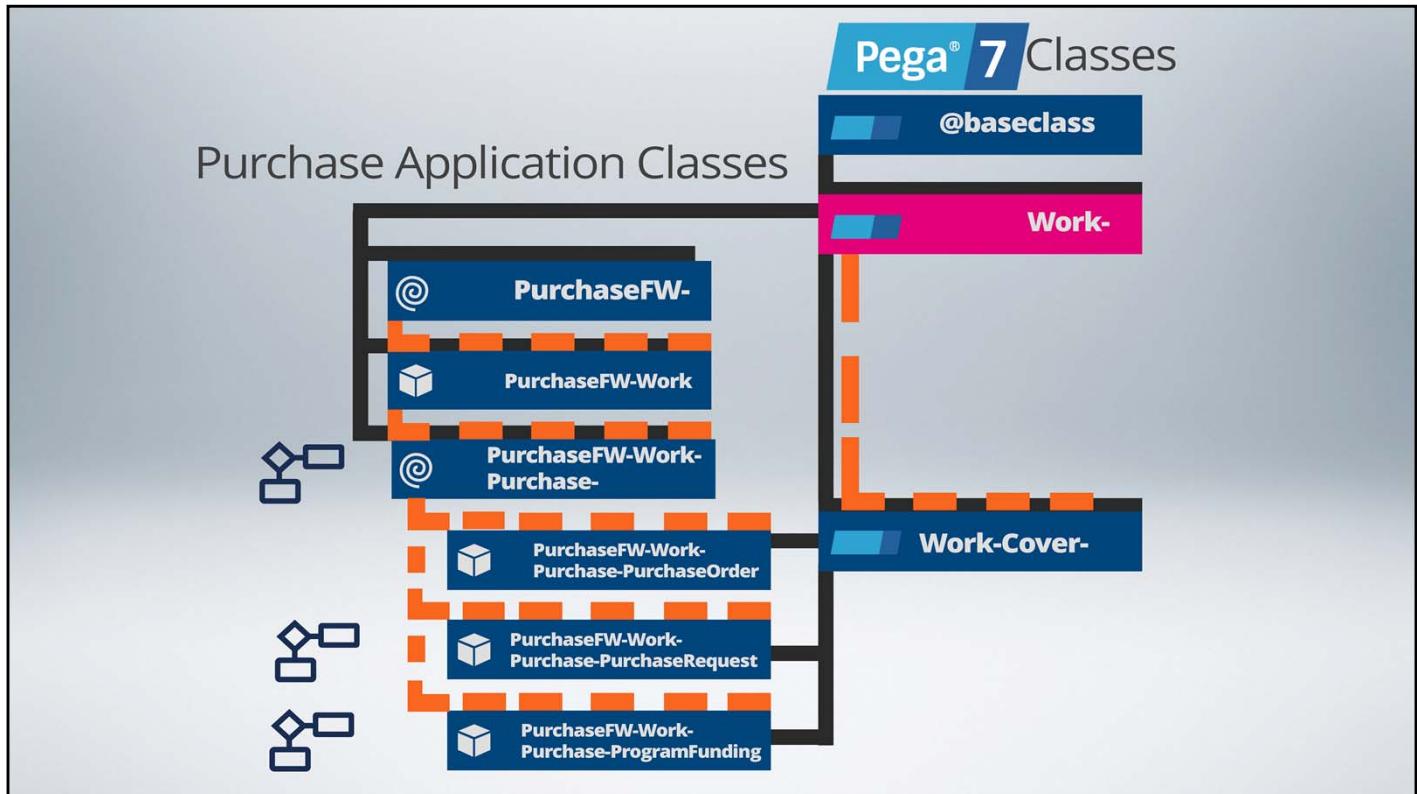


PRPC continues using the pattern inheritance algorithm to determine if there are anymore rules that match.

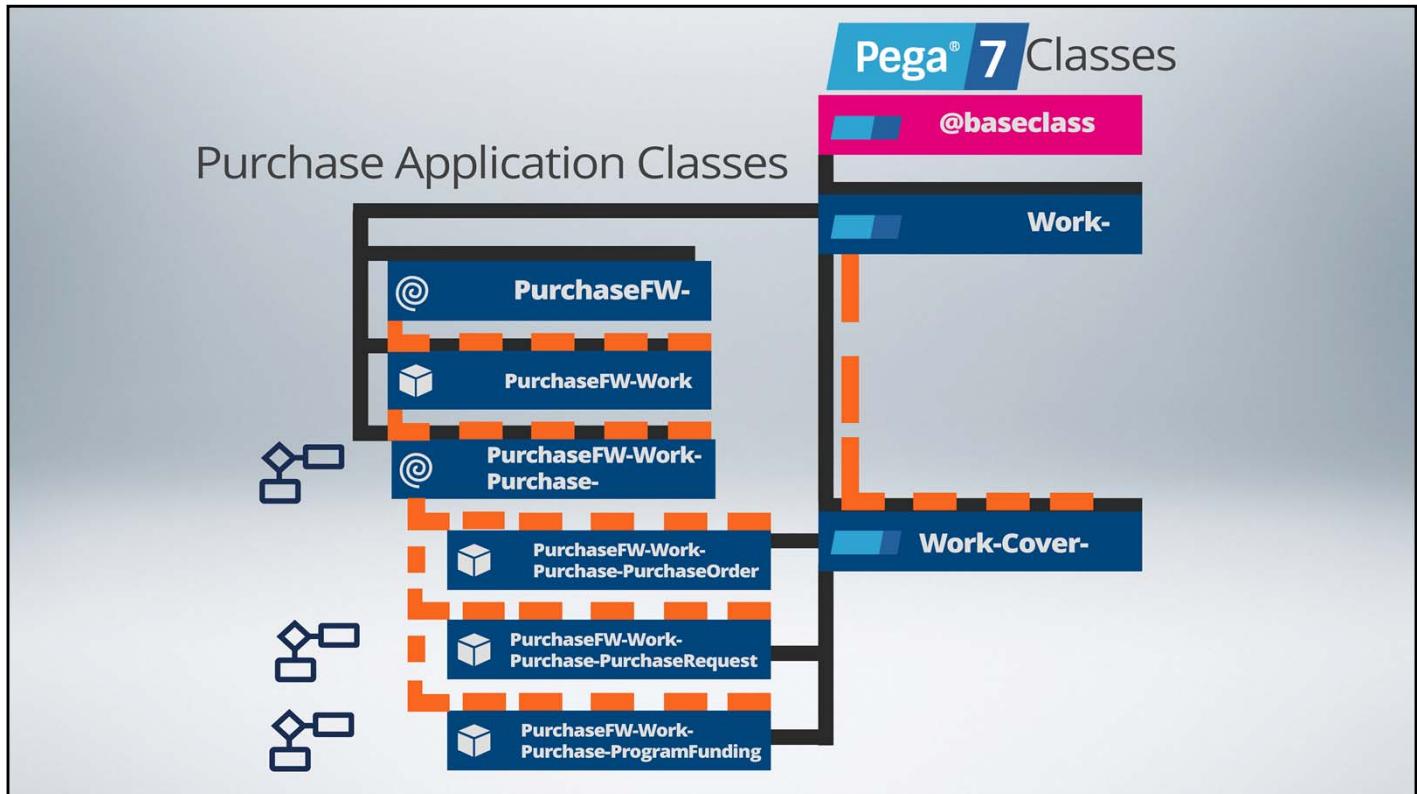




Once pattern matching is complete for the ProgramFunding class,  
 PRPC checks the class that ProgramFunding directly inherits from, in this case that's Work-Cover-.  
 What type of inheritance will be used next to determine where to go after Work-Cover-?



Remember that pattern inheritance is always done first, so the next place to look is Work-. After Work- the next place to look would be the directed parent of Work-Cover-, which in this instance happens to be Work-.



PRPC has already checked Work- to see if there is a rule so it doesn't do that again. PRPC then checks pattern inheritance for Work-. In this case there aren't any additional classes since PRPC would have already checked them when it did pattern inheritance for Work-Cover-. Therefore PRPC does directed inheritance for Work- which is @baseclass.

The additional steps in the rule resolution process will then determine which one of those rules to use.

## Demo



### Navigating an Application using Application Explorer

You are probably already really familiar with the Case Explorer to examine your application. But, what if we wanted to examine the class structure of our application? Or, what if we wanted to examine all the different artifacts for a given class? Or, what if we wanted to see what the inheritance path was for a given class? How would we accomplish this? To accomplish this PRPC uses something call the Application Explorer.

The Application Explorer is used to navigate the class structure and is organized by class, category and rule type.

The Application Explorer can also drill down into the other classes in the class hierarchy.

Notice that it has three classes under it, PurchaseFunding, PurchaseOrder and PurchaseRequest.

The text box at the top of the Application Explorer is used to determine which class should be examined.

We can either type in the name of the class we are looking for, or hit the down arrow to use the autocomplete option to list all the classes with whatever is currently typed in the box as the root of the name.

We talked a lot about inheritance in this lesson, the Application Explorer allows us to examine the inheritance path for a class. To do this right click a class and choose Inheritance.

This table is what PRPC uses to determine potential rules for execution in the PurchaseRequest class.

Now let's examine the relationship between what is created in the Case Explorer and Case Designer with what is created in the Application Explorer.

First in the Case Explorer let's look at the existing structure. There is a ProgramFund case with a Purchase Request subcase that has PurchaseOrder and Vendor Maintenance as subcases.

What does this look like in the Application Explorer?

Notice that in the case designer we have one structure which shows how the different cases created interact with one another whereas with the Application Explorer we see the different rules that make up each of those cases. It is important to note that just because a class is being used as a subcase does not mean that it will be a

subclass in the class structure.

What about the artifacts that are created in the Case Designer? Let's take a look at the Enter Request Details process.

So where is this process if we wanted to find it in the Application Explorer? First we need to determine the name of the process which is located underneath the editing icons for the process.

Now let's see where that is in the Application Explorer.

Notice in the Application Explorer we are explicitly looking at the PurchaseRequest class.

To find the PurchaseRequestEntry process we would need to expand Process, then Flow to see the list of processes.

Looking in Flow we can see the PurchaseRequestEntry process.

Now let's take a look at how we would find a flow action that is used in an assignment.

For this example let's look at the Add Line Items assignment.

The Flow action defined here is called AddLineItems.

Now let's find the that Flow action in the Application Explorer.

Flow actions are found under User Interface.

Then we expand Flow Action to see the different Flow actions for the class.

Here we see the AddLineItems Flow action.

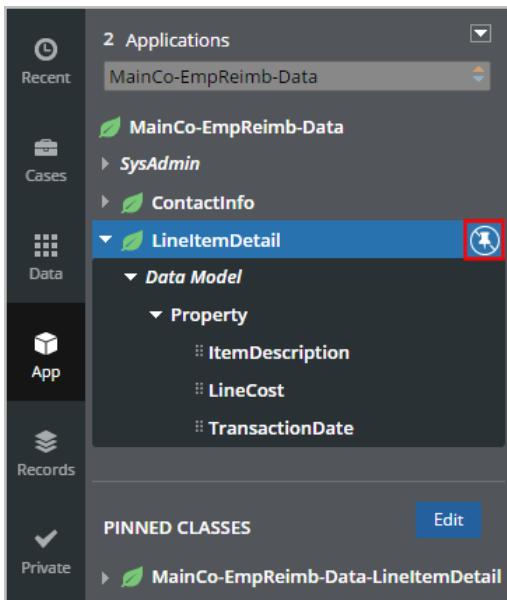
As we can see everything that is done in PRPC can be found using the Application Explorer. It is a great resource to use to view the structure of our classes and rules.

## Pega 7.1.6 Update Notes

### Pinned Classes section added to Application Explorer

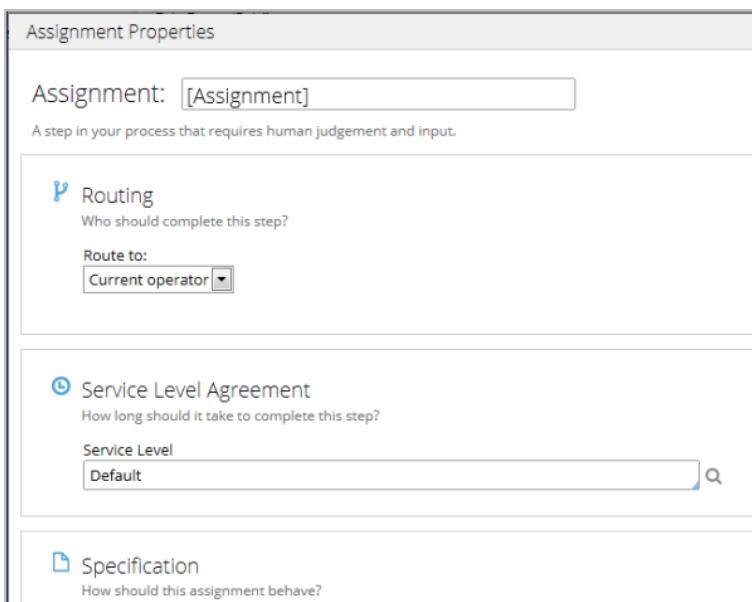
Allows you to customize a list of classes to access without explicitly switching context

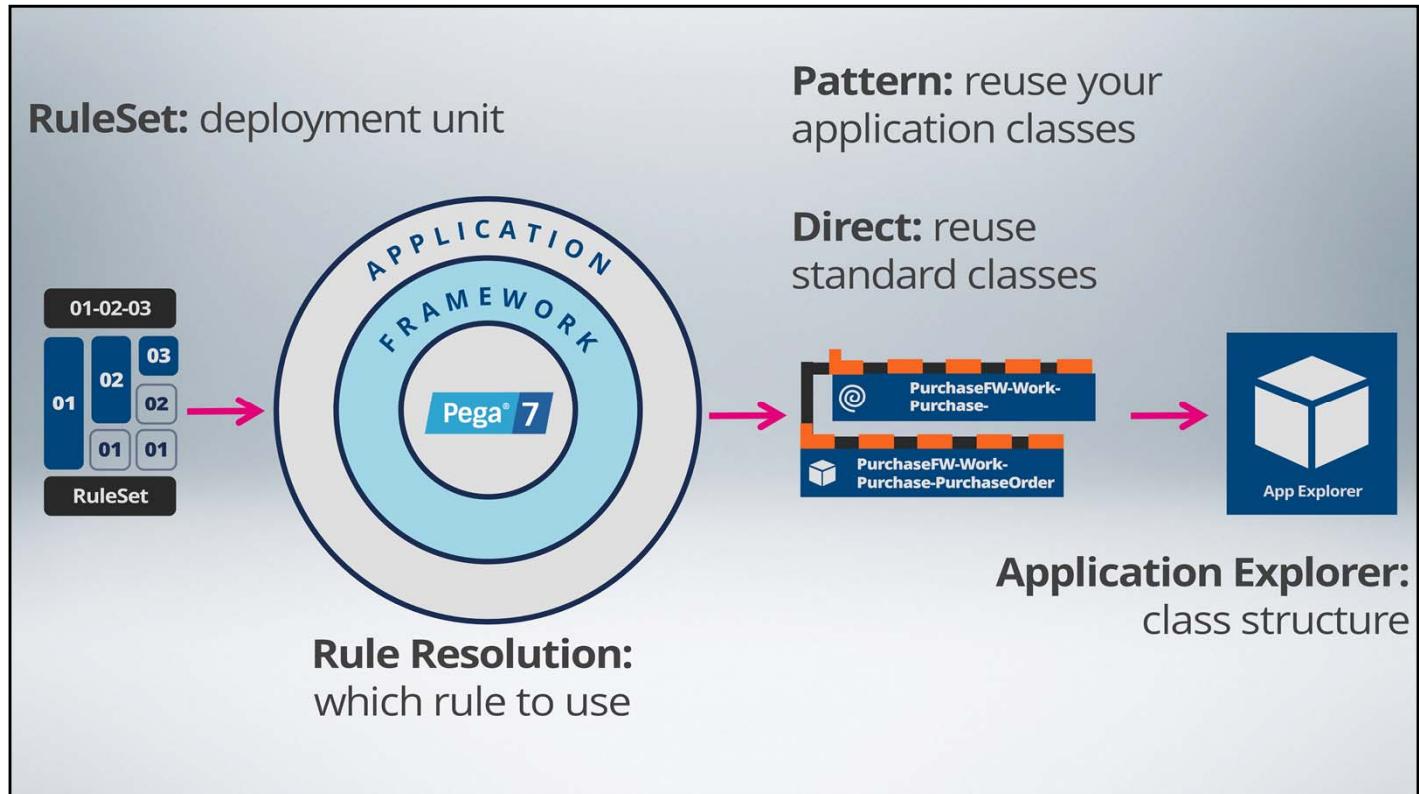
- Hover over class to see icon to add to pinned classes, or remove from pinned classes



### Property panels were reorganized for ease-of-use

- Tabs were eliminated.
- All fields are located on a single panel and are grouped by their function.
- The most commonly used fields appear at the top of the panel and are clearly visible.
- Field and functionality of interest to advanced users are organized into an expandable Advanced section positioned at the bottom of the panel.
- Business-friendly labels and headings, and instructional text make it easier to quickly understand the meaning and purpose of each field.





We have discussed how to manage the basic building blocks of a PRPC application and how each of these building blocks can contribute to creating a reusable application. RuleSets are used to deploy to a server a set of rules that make up our application.

What is inside that RuleSet? There are layers to an application consisting of a PRPC layer, a framework layer and an application layer.

As we move through the different layers the rules go from being more generic, the PRPC layer, to being more specialized, the application layer.

Having different layers helps us to create applications that are reusable, but what happens if rules have the same name? PRPC uses a rule resolution process to determine what rule should be used depending on what the application is doing.

There are a lot of factors in the rule resolution process. In this lesson we focused on the Purpose stage which uses class inheritance to determine candidate rules to execute. The two types of inheritance are pattern and directed.

Pattern inheritance is for sharing the objects in our application hierarchy.

Directed inheritance is for utilizing the objects provided by standard packaged classes.

In order to examine our application and to see the class hierarchy we have created we can use the Application Explorer. The Application Explorer allows us to navigate all the classes in an application and provides us an organized way to find the different rules in our application.

By taking advantage of RuleSets and class inheritance we can help create reusable applications.

# Guided Application Development Using Guardrails

In this lesson we look at how Guardrails guide you in developing applications in Pega 7.

At the end of this lesson, you should be able to:

- Understand the role guardrails play in identifying potential problems
- Use the dashboard to find existing guardrail warnings in your application
- View a guardrail warning in Pega 7

As you work with Pega 7 you may see messages appear:



This cell has conflicting row  
and column conditions

This rule contains 4 logic conflicts

These properties are not optimized

As you work with Pega 7 you may see messages appear.

## Warning signs



Best Practices for  
**Pega 7** Development

While not errors, these messages are important; they are known as guardrails. And, if not taken care of they could lead to problems in your application.

Guardrails help to ensure that the application we create follows the known best practices for Pega 7 development. It is really important that we follow them unless we have an extremely good reason not to.



## What happens if I don't follow best practices?



Increased development time

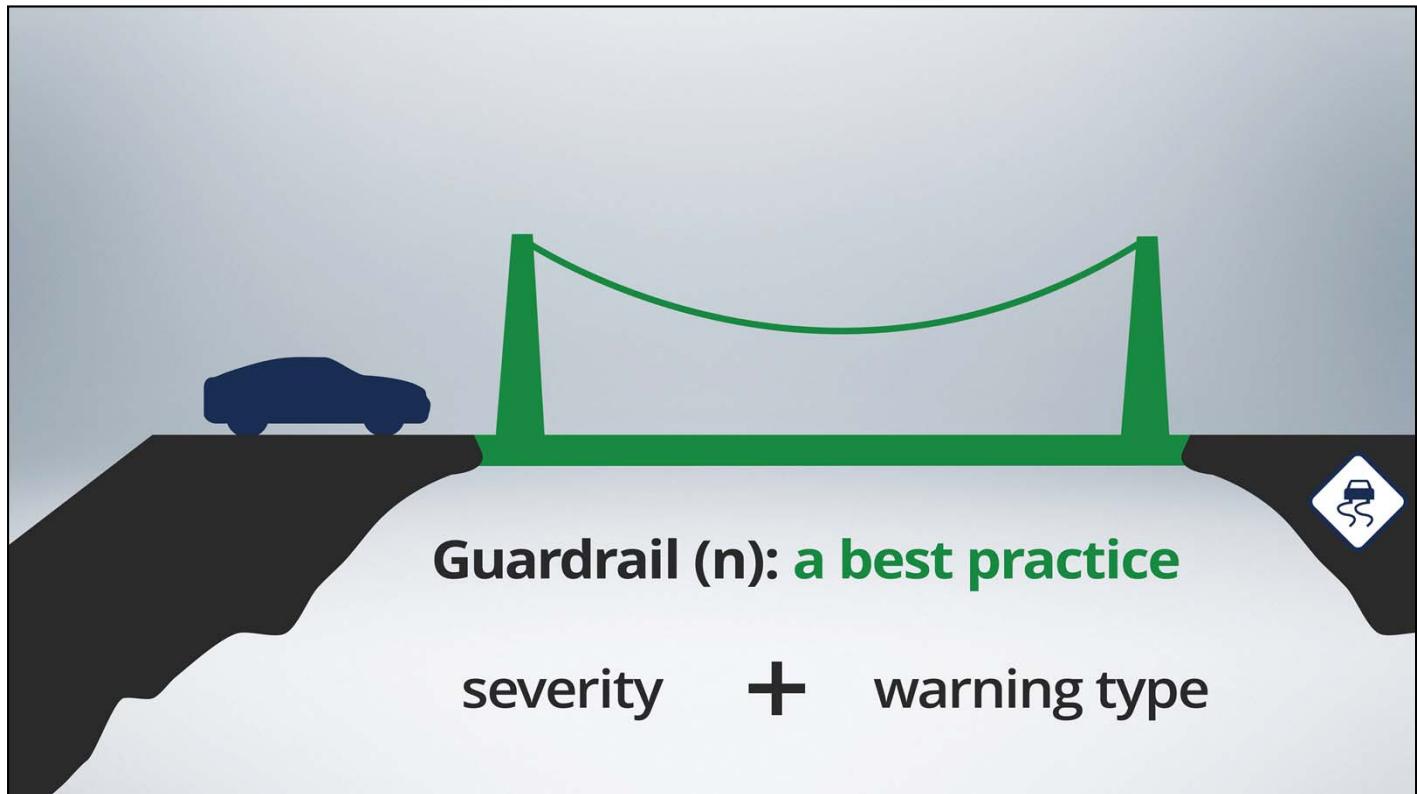
Features not implemented correctly

Less maintainable

When creating an application it is always a good idea to follow the best practices for whatever technology we are working with. What happens if we don't follow known best practices?

We don't exactly know the answer to that question but by not following best practices we are certainly adding increased risk that our application development effort could get off track, the application does not work as good as it could or should. It can also make the application difficult to maintain and update.

It is also possible that our application can get out of control and crash the project entirely.



Guardrails are best practices and guidance about situations that contain risky conditions or that might result in an undesirable outcome.

Every guardrail warning consists of a severity level and a warning type.

Pega professionals identify guardrails from developing, testing, and reviewing Pega 7 applications in a variety of industries and organizational settings. Guardrails ensure you and your team use Pega 7 according to their guidance, and help you avoid troublesome situations.



**severe (adj): high risk**



**moderate (adj): medium and long term effects**



**caution (adj): alternate way**

There are three different levels of guardrail severity. **Severe** - Most serious (highest level) indicates a serious guardrail violation.

These are typically things that pose the highest risk, such as things that can impact data integrity. For example, if we were to create a Data Transform and we added an action that Pega 7 determines to be unreachable we would trigger a severe guardrail.

**Moderate** - Indicates a moderate guardrail violation. These are typically things that can have medium and long term effects on the project if left unresolved, such as things that can impact maintenance.

For example, If a property used in a report is not optimized for reporting we will get a moderate warning because it could impact performance.

**Caution** - Indicates the least serious type of violation. These are typically things where there is a recommended better way to accomplish the goal.

For example, in a Correspondence rule if we add an inline `<script>` or `<style>` HTML tag we would trigger a caution guardrail because as a best practice it is good to avoid using those in a correspondence.



## Warning type (n): category for a guardrail



Data Integrity



Performance



Maintainability

**Warning types** are the different categories that a guardrail could apply to. There are many different types that guardrails can be categorized as.

Some examples of warning types include Data Integrity, Maintainability, and Performance. Data Integrity warnings revolve around the aspects of the rule that might adversely affect our data.

For example the Case Dependency guardrail is a Severe-Data Integrity warning which means that the case type name was removed and dependency information may be incorrect.

**Performance warnings** occur when options set in the rule might adversely affect application or system performance.

For example, the Unexposed Properties Exist guardrail is a Caution-Performance error when we are using properties in a report definition that has not been optimized.

Maintainability warnings occur when a rule uses custom code or activities, which can result in maintenance and upgrade issues over time.

The most common guardrail for this is the **Custom Code** guardrail which can happen in most of the rule types.

## Demo



## Using The Guardrails Dashboard

## Pega 7.1.6 Update Notes

### Updates for Pega 7.1.6:

- The default guardrail warning page has been simplified

The screenshot shows a simplified dashboard titled "Guardrail Warnings (last 7 days)". It includes a "View all warnings" link and a "Refresh data" button. Below this, there are two horizontal bar charts. The first chart, under "Introduced by you", shows 0 Severe, 0 Moderate, and 2 Caution warnings. The second chart, under "Introduced by team", also shows 0 Severe, 0 Moderate, and 2 Caution warnings. At the bottom, a section titled "Pega Task" indicates "No tasks assigned" and provides a "New task" link.

- A new Charts tab has been added

The screenshot shows the "Application - Guardrails" interface with the "Charts" tab selected. The top navigation bar includes "Compliance Score", "Summary", "All Warnings", and "Charts". The "Charts" tab is highlighted in blue. Below the navigation, there are filter options: "Include rules modified on or after" set to "9/15/2014" with a calendar icon, and a checkbox for "include justified warnings" which is unchecked. A "Apply Filters" button is present. On the left, there's a sidebar with links for "Key Guardrail Summary Metrics", "Key Warnings Over Time", and "Rules by Last Update Operator". On the right, a chart titled "Rules by Last Update Operator" is displayed. The chart has "Update Operator Name" on the y-axis (listing "Administrator", "Application Developer 01", and "[Empty]") and "Rules with Warnings" on the x-axis (ranging from 0 to 160). The legend indicates that blue bars represent "Rules with Warnings" and yellow bars represent "Total Rules Updated". The chart shows that the "Administrator" operator has the highest number of total rules updated, followed by "[Empty]" and then "Application Developer 01".

## Review



### Guardrail (n): a best practice



Three severity levels



Warning types are categories



Dashboard



We have discussed what guardrails are and why they are important for us to monitor when building applications.

By following guardrails we are implementing best practices in our application.

Guardrails are comprised of a severity and a warning type.

Severity indicates how serious a violation we have created and there are three levels severe, moderate and caution.

Warning types are categories for guardrails and there are types for all different aspects to creating an application in Pega 7.

In Pega 7, we can track the guardrail violations for a given application by looking at the dashboard. It provides information about the existing guardrail violations in our application, and which guardrails we specifically violated.

There will always be exceptions to guardrails. We will find that we need to “justify” certain guardrail warnings because there simply is no other way to implement the given solution. However, if we can minimize these exceptions, or justifications, we will build better applications that are Built For Change®.

## Exercise: Understanding Guardrails



## Module 03: Designing Enterprise Applications Using Case Management

This lesson group includes the following lessons:

- Best Practices for Case Management Design
- Managing Enterprise Apps Using Stage-Based Case Design
- Best Practices for Effective Case Decomposition
- Best Practices for Effective Process Decomposition
- Guardrails for Case Management Design

# Best Practices for Case Management Design

Learn how to maximize implementation efforts by expressing and interpreting business requirements consistently using Pega 7.

At the end of this lesson, you should be able to:

- State the difference between case and process
- Answer the question "Why case management?"
- State at least three strategic reasons for adopting case management
- State at least three best practices for case management design



# Case

# Process

A diagram illustrating a process flow. It starts with a decision diamond at the top, which branches into two parallel rectangular boxes. These two boxes then merge back into a single rectangular box at the bottom.

To fully understand the benefits of, and best practices for, case management design, let's answer the fundamental question:

Why case management? Who moved my....process?

Let's begin with confirming our understanding of these two terms: *case* and *process*.

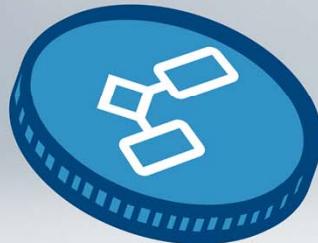
**case (n):** a business transaction to resolve



Case and process really are just two sides of a single coin. To put this in the simplest of terms:

A case is a business transaction we want to complete; it is *what* we do.

**case (n):** a business transaction to resolve



**process (v):** the path a case may take

Process is the path, or paths, the case follows as it is completed; process defines “how” we do what we do.

## Examples of Cases

### Cases



### have Status

Case number: 2013-02-245

Case Status: Pending Trial

### and at least one process



Loan ID: 76567-098

Case Status: Underwriting



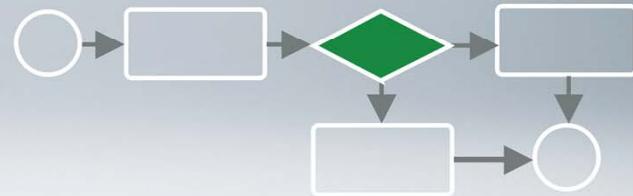
Claim number: 334-24532

Case Status: Adjuster Review



Bug ID: 123456

Case Status: Triage



Cases come in many shapes and sizes.

Law enforcement agencies have crimes to solve, mortgage companies have loan applications to approve, insurance companies have claims to settle.

We even have cases in our line of work - application development projects have defect reports - bugs - to fix.

Cases have a life cycle.

Crimes, loan applications, insurance claims and software defect reports, are all numbered, tracked, and managed.

All cases have at least one associated process, whether that process is formalized or not - or even structured or not.

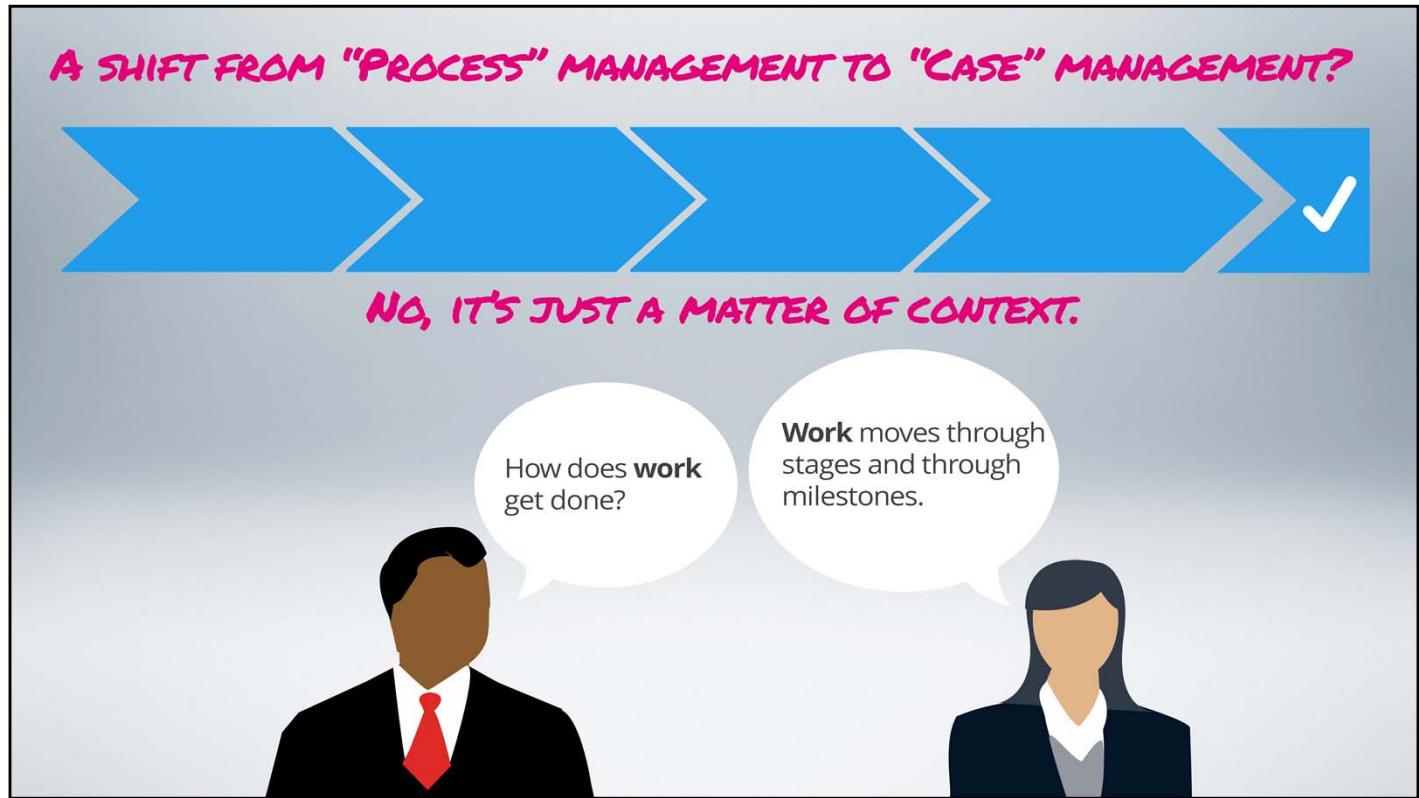
As a person works through a case, he or she is tracing a process for how to handle that case.



A case has stuff associated with it. It has people assigned to work the case; tasks to be performed; relevant documents and correspondences; a status; a history of actions taken;



and a resolution.



This makes the obvious answer to the question “what is case management” to be the management of these artifacts.

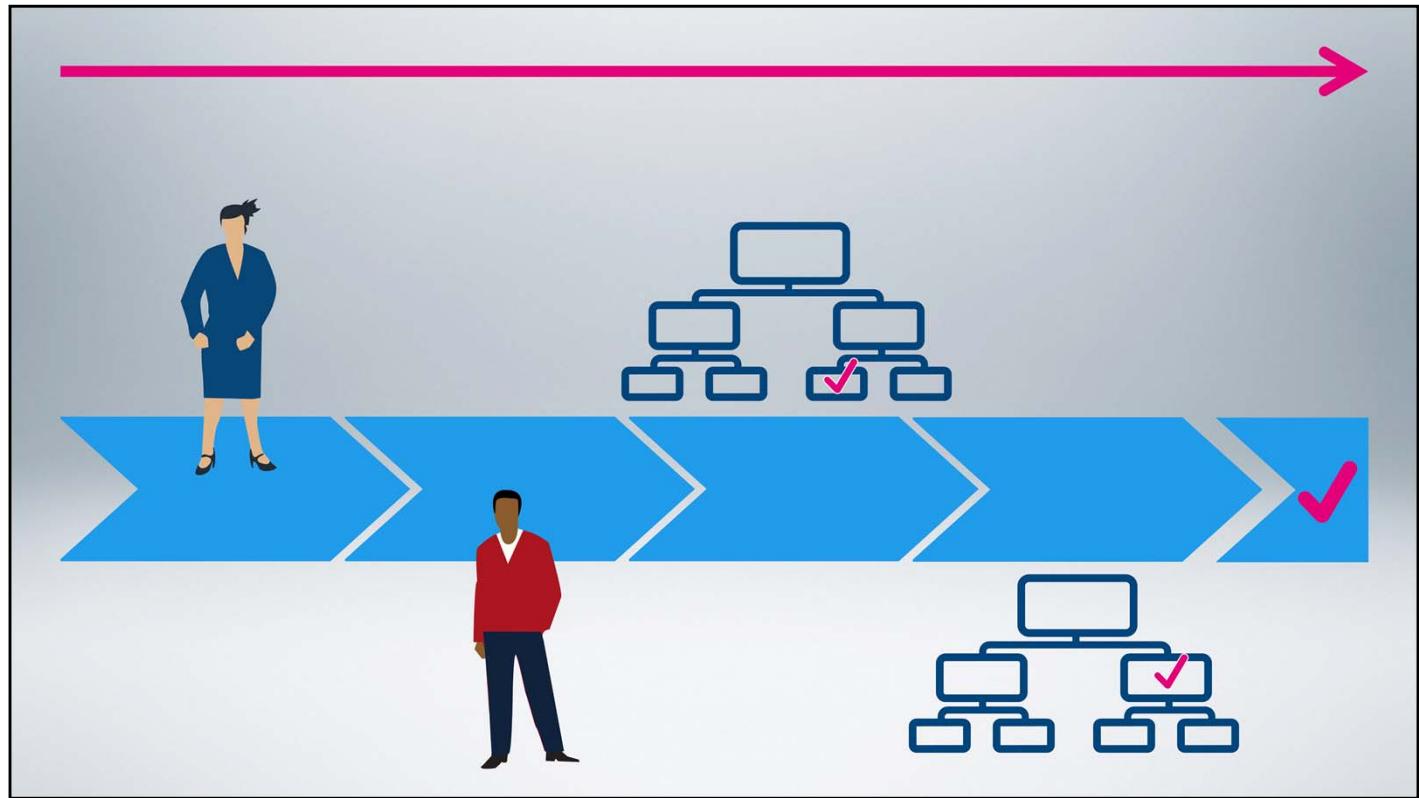
So..., why the seeming shift from process management to case management?

There really isn't a shift – it turns out to be a matter of context.

When we, here at Pega, talk to business people – the business visionaries, the business leaders, analysts and workers –

and we ask: “How does work get done in your organization?” we notice a very common pattern to the answer. They tell us:

“Work moves through stages and through milestones.”

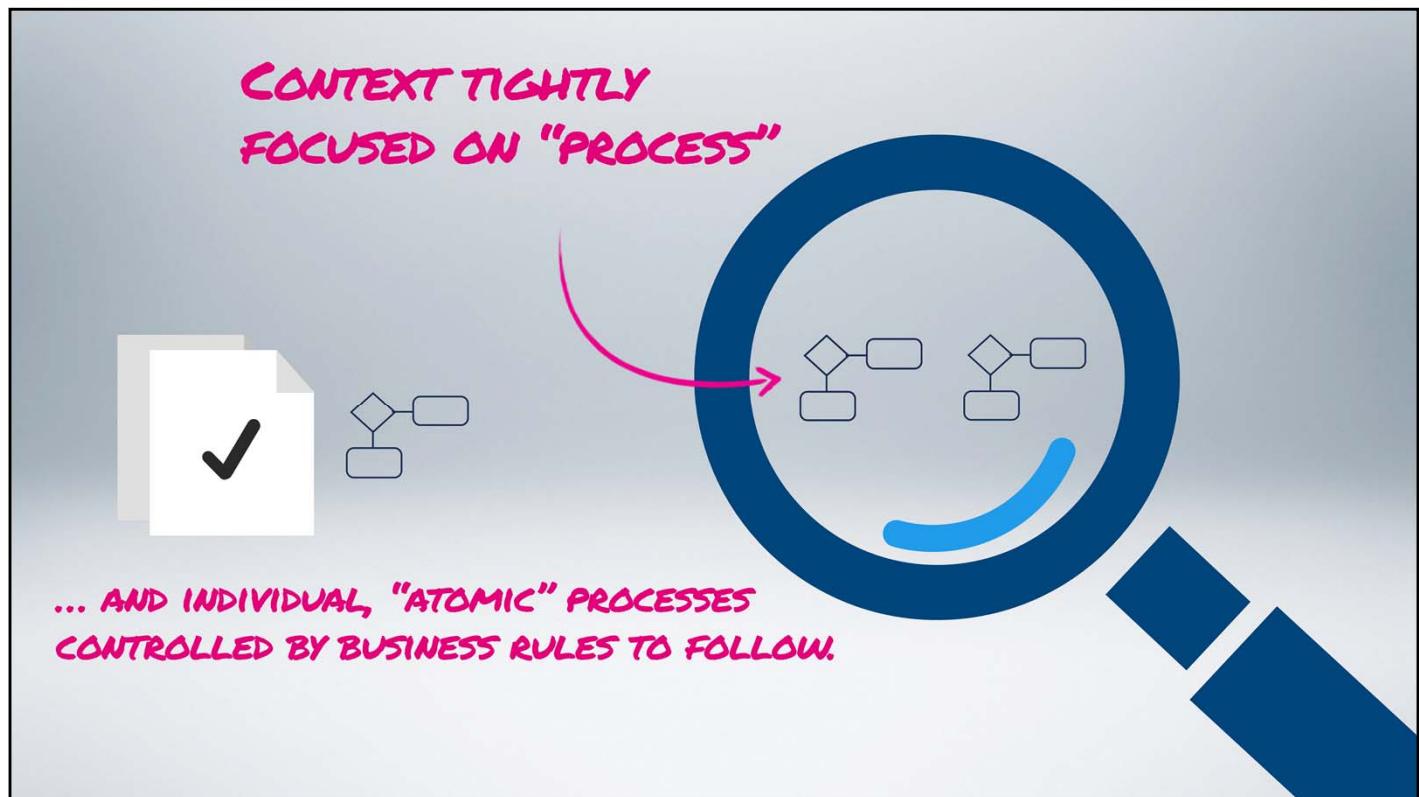


They understand the business transaction - the case - as it moves from one person to the other, from one part of the organization to another.

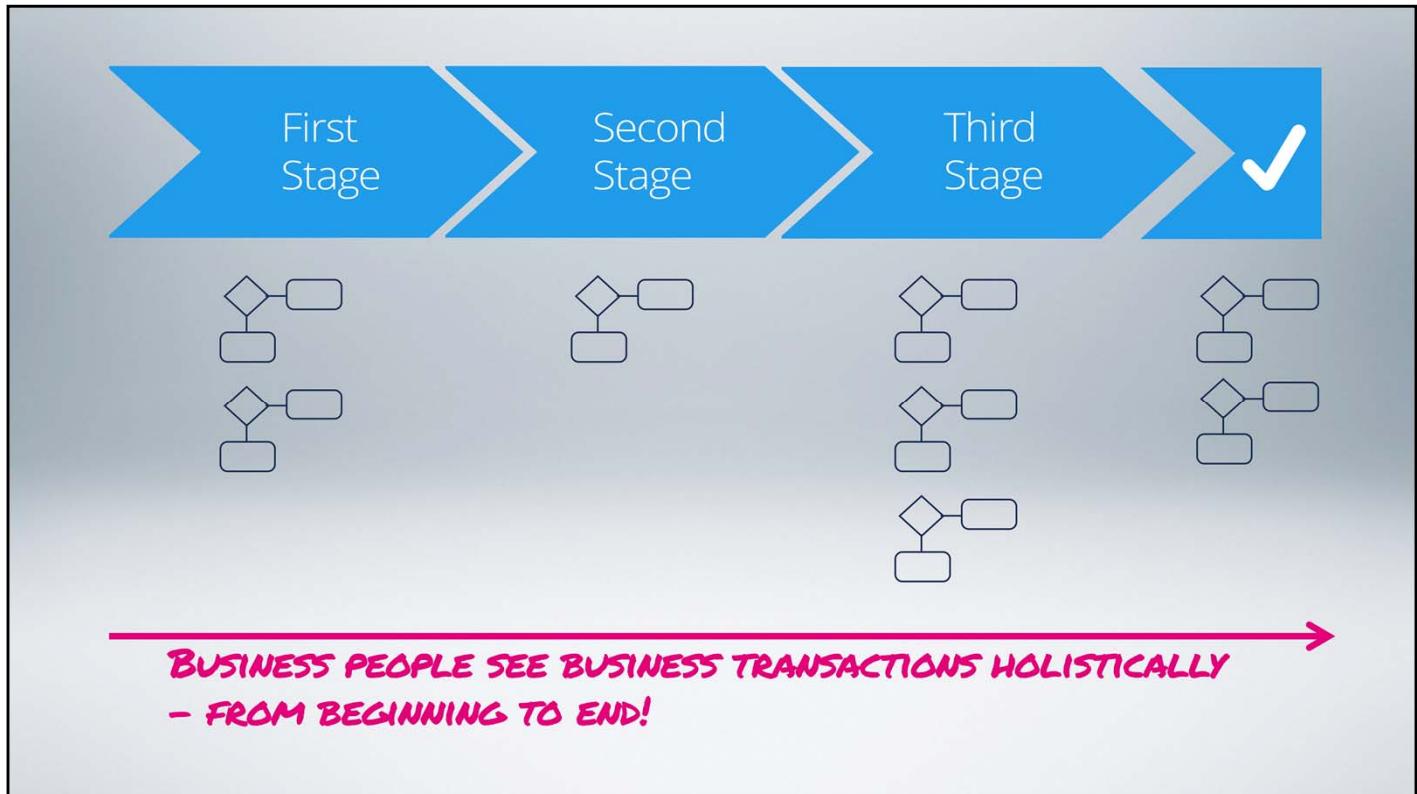
**TRADITIONALLY, A BUSINESS TRANSACTION IS A MYOPIC VIEW OF A SERIES OF FORMS TO COMPLETE...**



Rather than a myopic view of a business transaction as a series of forms to complete

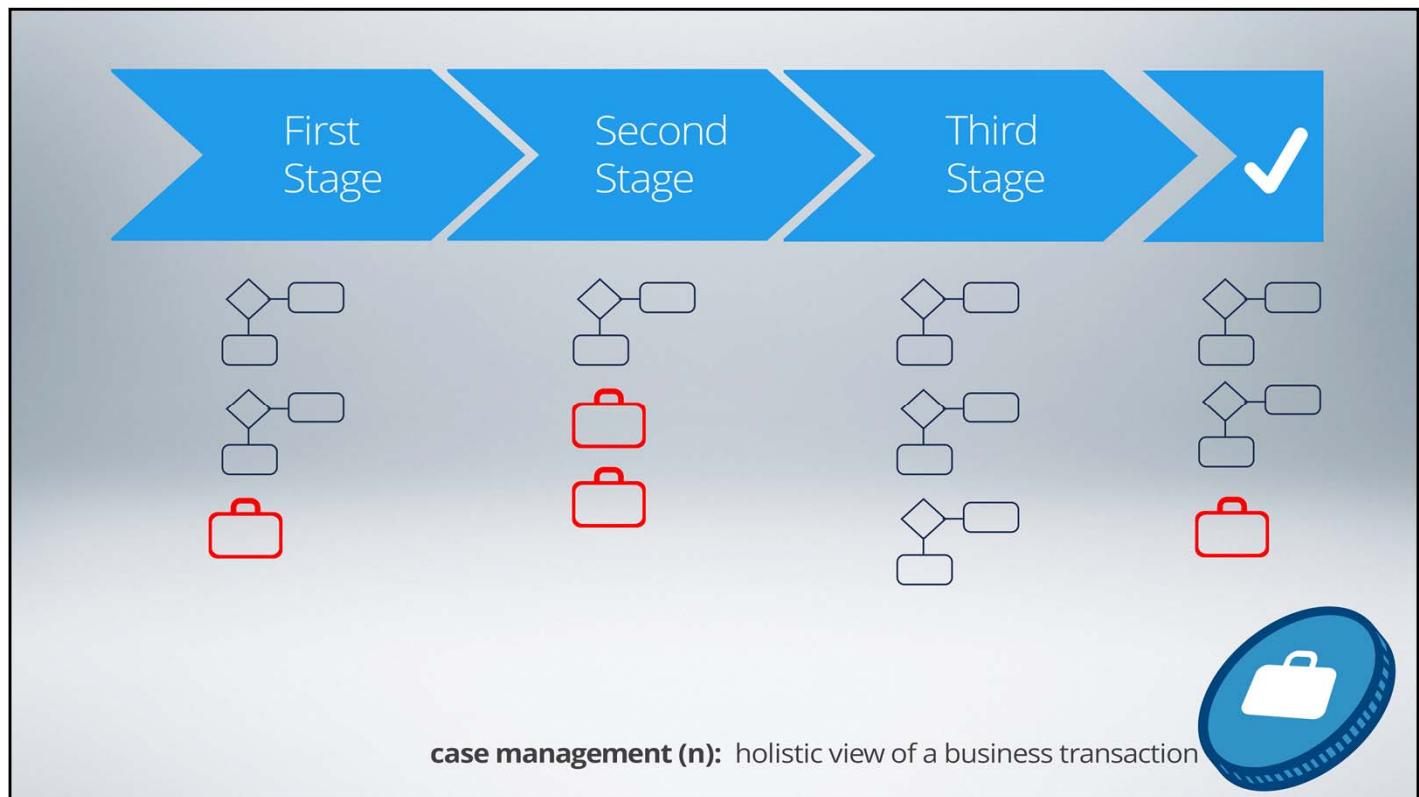


and individual, “atomic” processes controlled by business rules to follow – the more traditional view of “process management”



they see the business transaction holistically - from beginning to end.

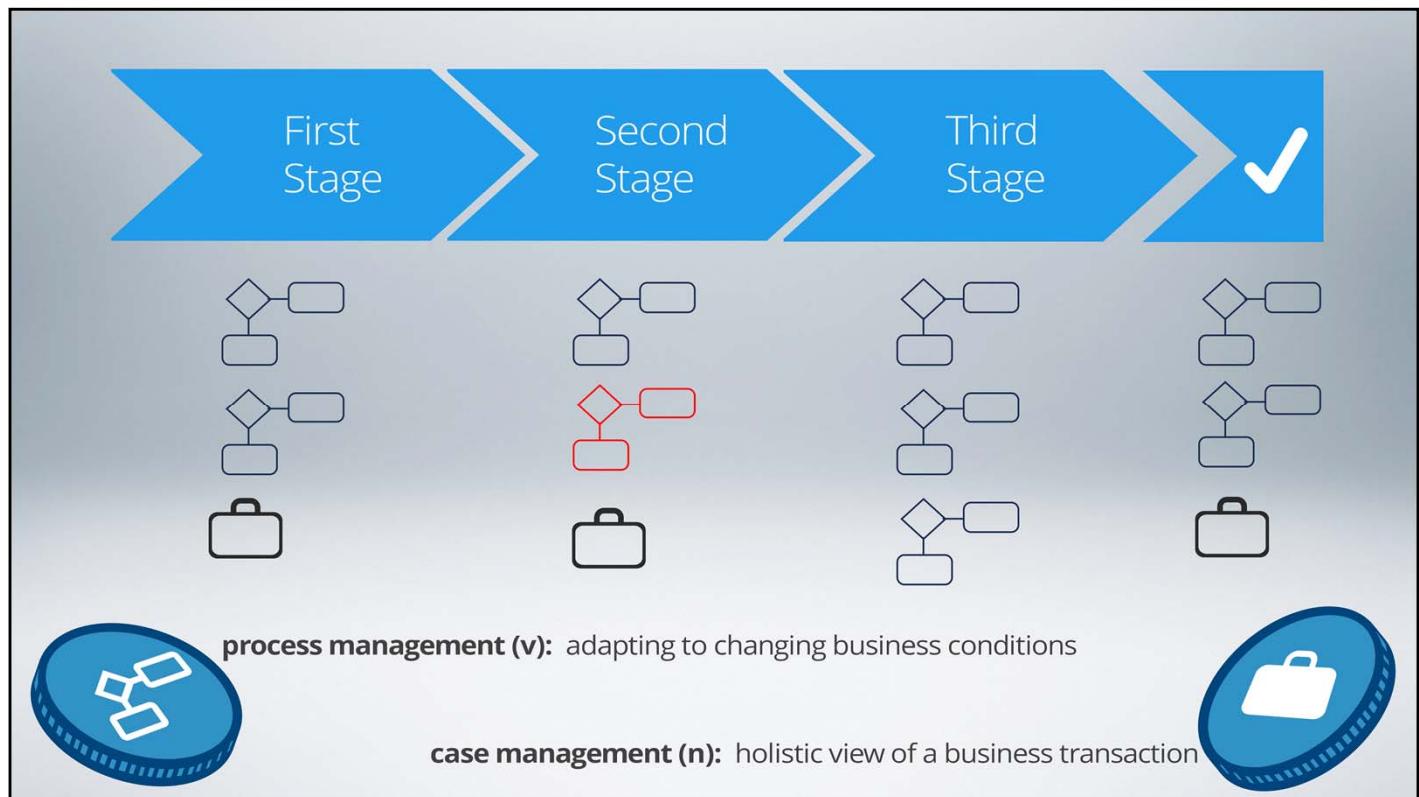
There is “work to be done” – the case - and “the process” is how they do that work. They have a “process” for resolving any given case.



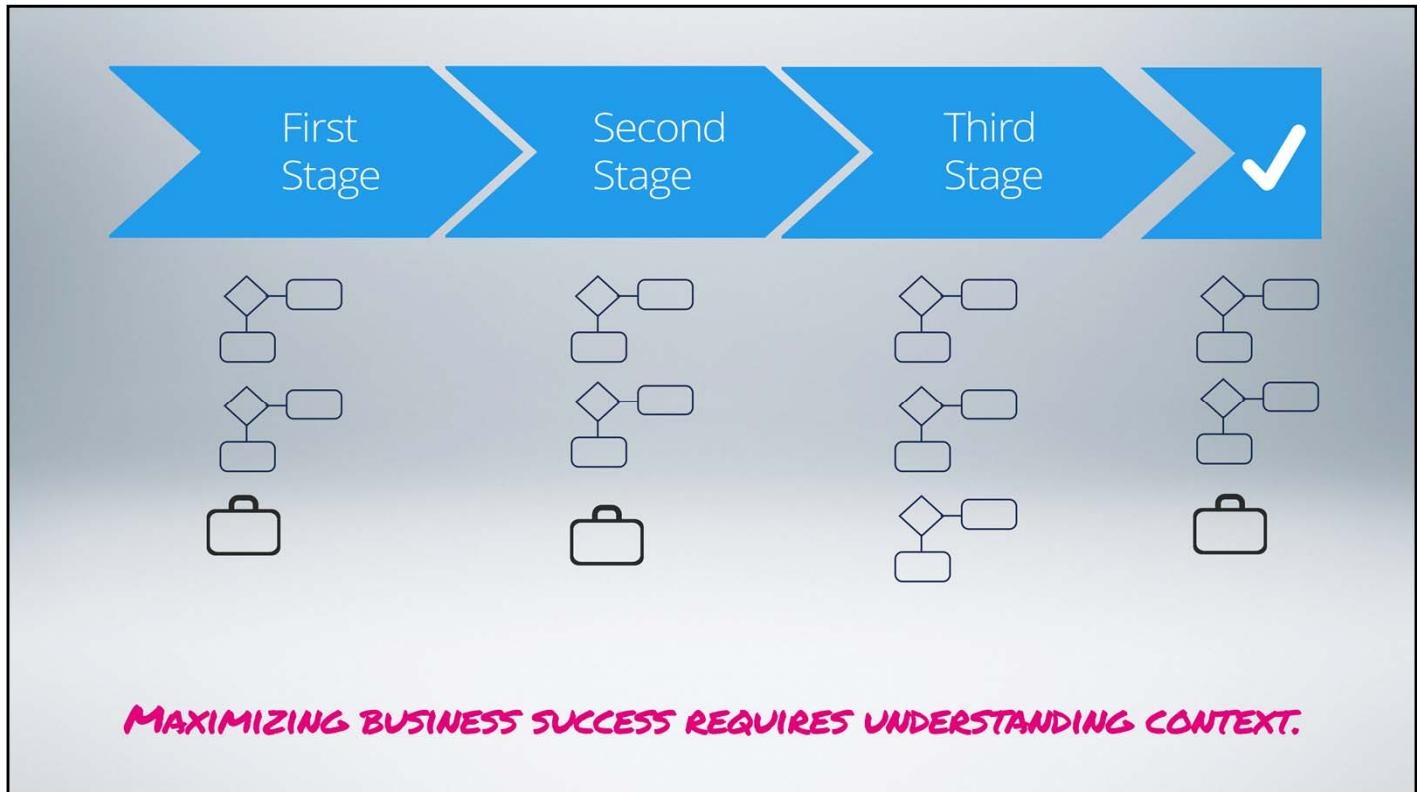
So, as with “case” and “process,” case management and process management are two sides of the same coin.

Case management provides for a more holistic view of a business transaction. It allows the business to see, and interact with, the business transaction the way it moves through the organization.

This becomes especially useful when you consider that a “case” may include not only processes, but other cases - or, sub cases - as well.



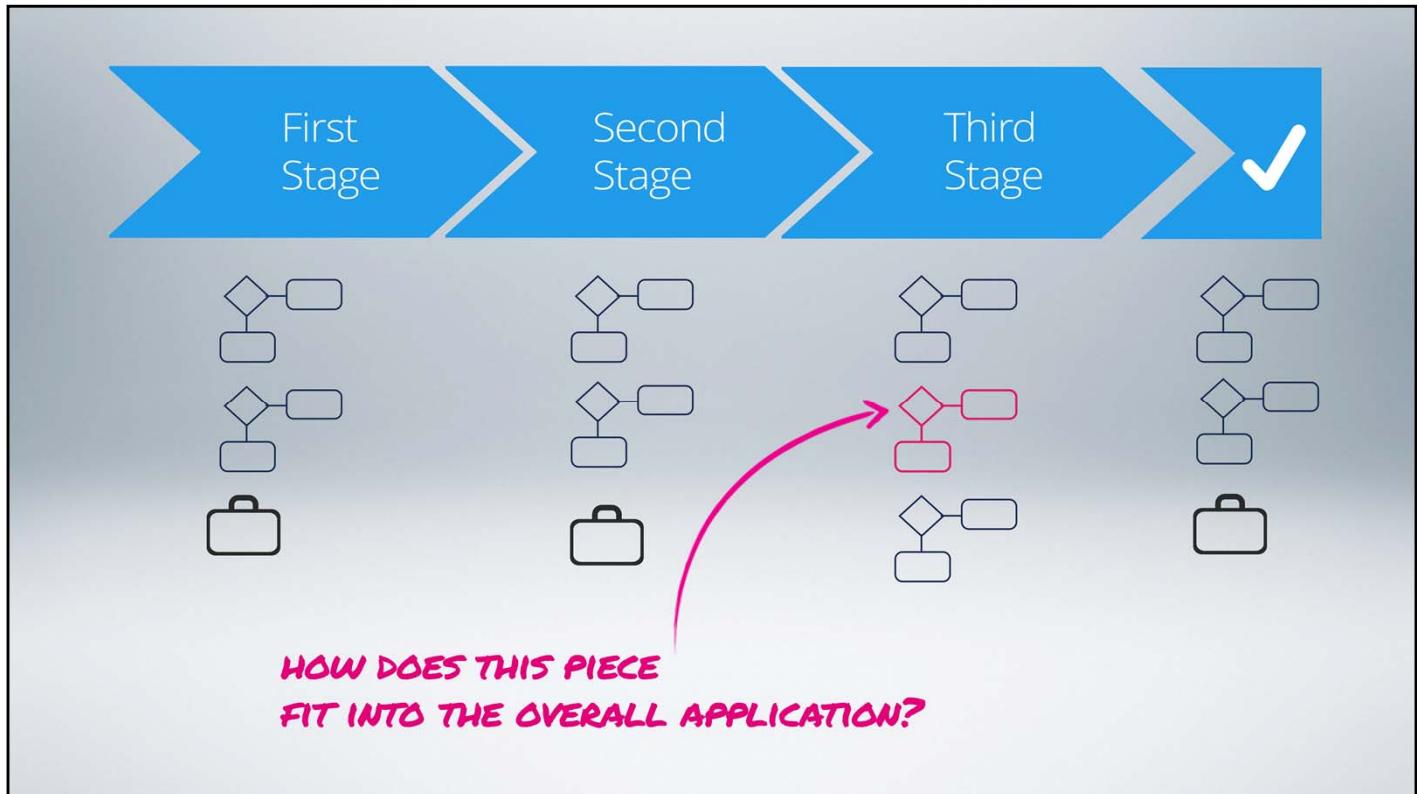
Process management is simply a way of ensuring a business transaction can be adapted to frequently changing business conditions.



Let's explore a few of the motivating reasons for adopting case management design.

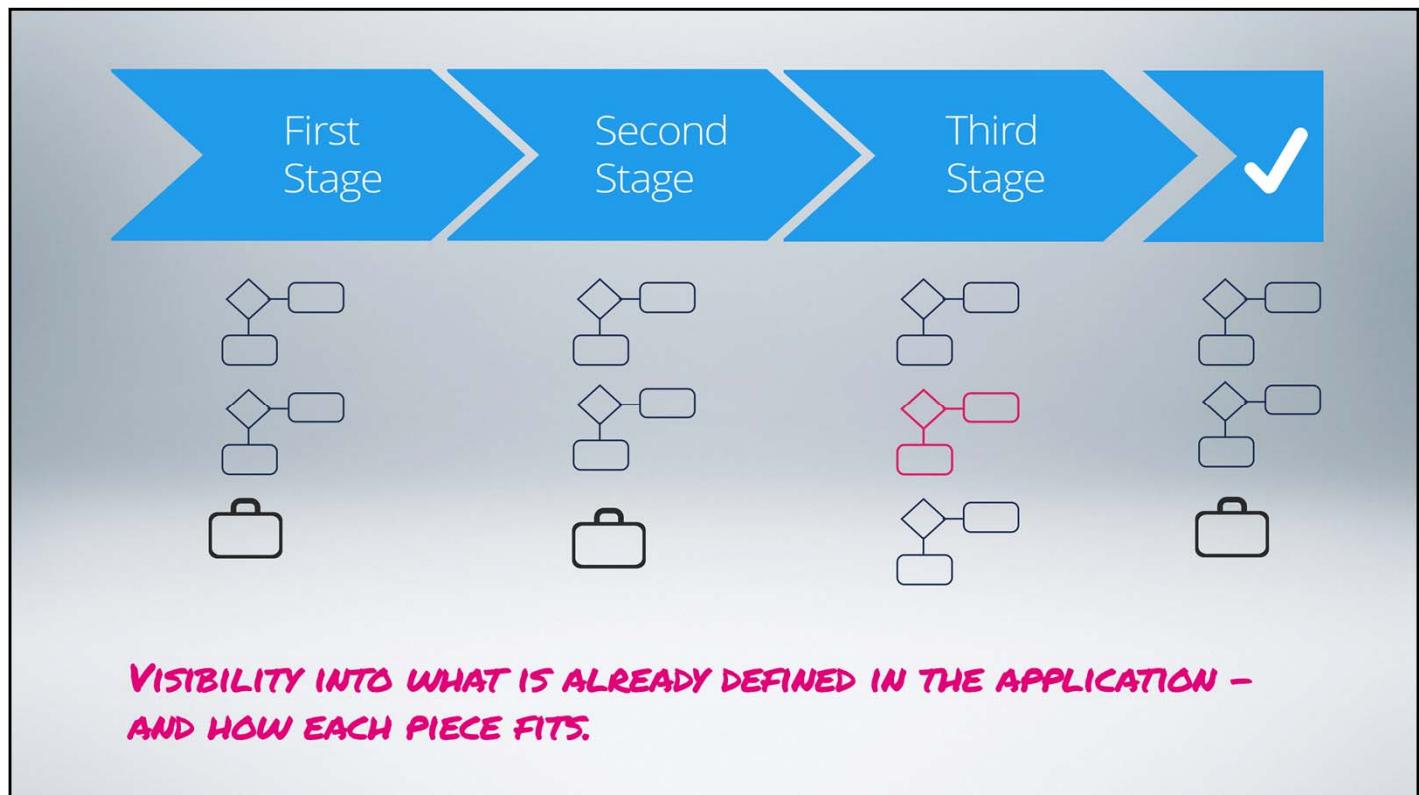
At the risk of repeating myself, case management design principles allow you to define the business application based on the work that moves through your organization.

This allows the business to work holistically and in context. It is this context that allows them to maximize their business success. But... it does not stop there – with the business.



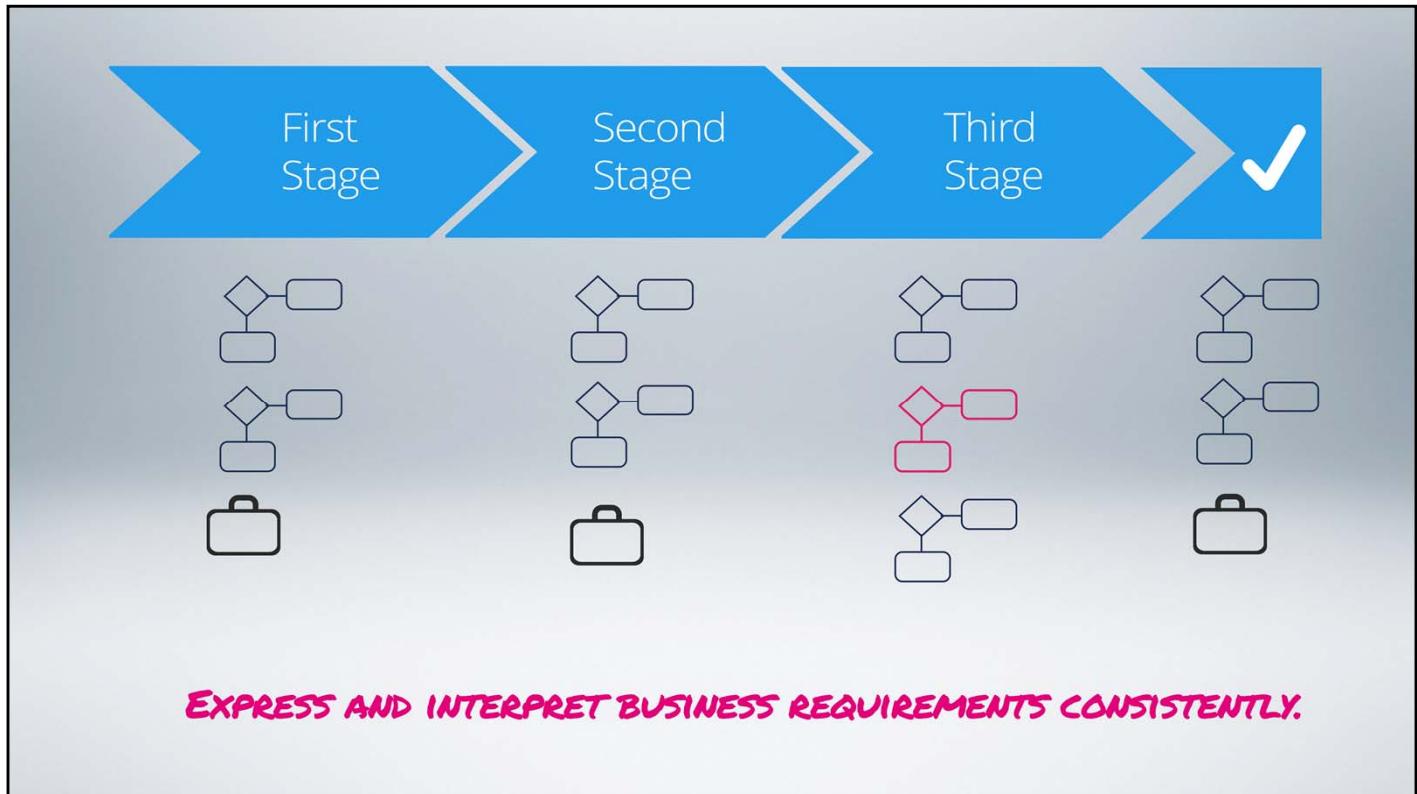
This holistic view of an application not only lets the business understand the full context of a business transaction, it also allows the technologists – you – to treat your work holistically as well.

How many times, on any given project, have you found yourself wondering: “how does this piece I am building fit into the application? How does it fit into the overall project?”



Using a case management design approach, you have visibility into what is already defined in the application.

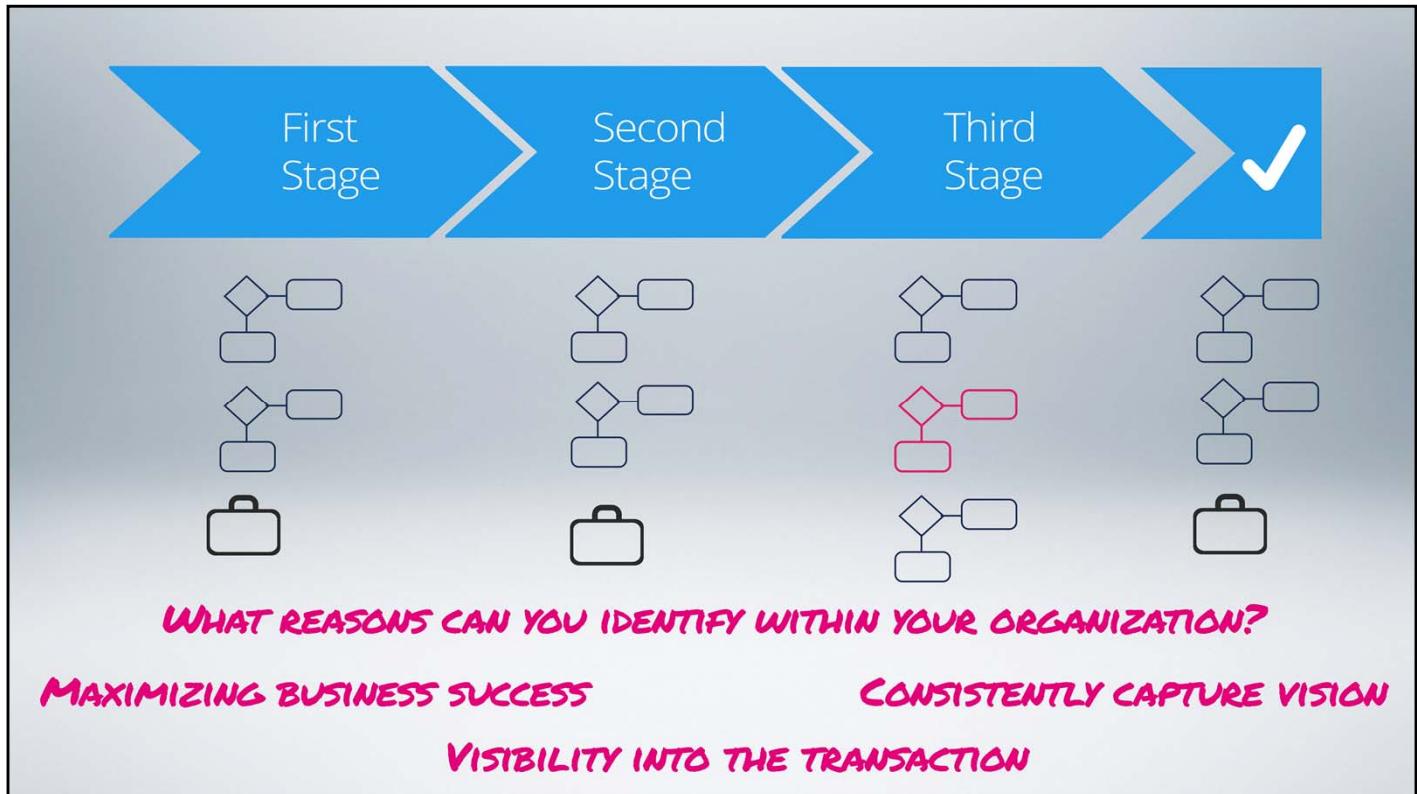
When building an individual piece, you can see how that individual piece works within the context of the business transaction, and how it fits into the overall project.



If this holistic view is how business people think about a business transaction, it should be how they can describe the business transaction to technologists.

And it should be how technologists interpret the description of that business transaction into a model easily understood by both the business and technologists.

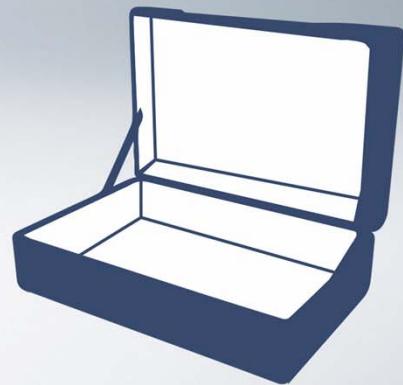
If we, the technologists define the business transaction the way business people discuss – and understand – that transaction, it makes it much easier to capture their vision.



These are three very common reasons for adopting case management design, however, your organization may have additional, or very different motivating factors.

As we close this topic, think about reasons your organization might have for adopting case management design. Knowing, and understanding these reasons will help you make the most of your case management design with Pega.

## Case Management Best Practices



As you will discover, the challenges of designing enterprise applications can be easily met by adopting a holistic view of that very application.

However, applying a few best practices will always go a long way in ensuring what the business describes is what the technologists will define, or build.



First, and probably most important, engage, and empower the business.

When business people are engaged in the project, it is much easier for everyone to align towards the outcomes they are looking to achieve.

PRPC 7 has integrated DCO management, enabling development teams to enter and review DCO information where and when it is needed.

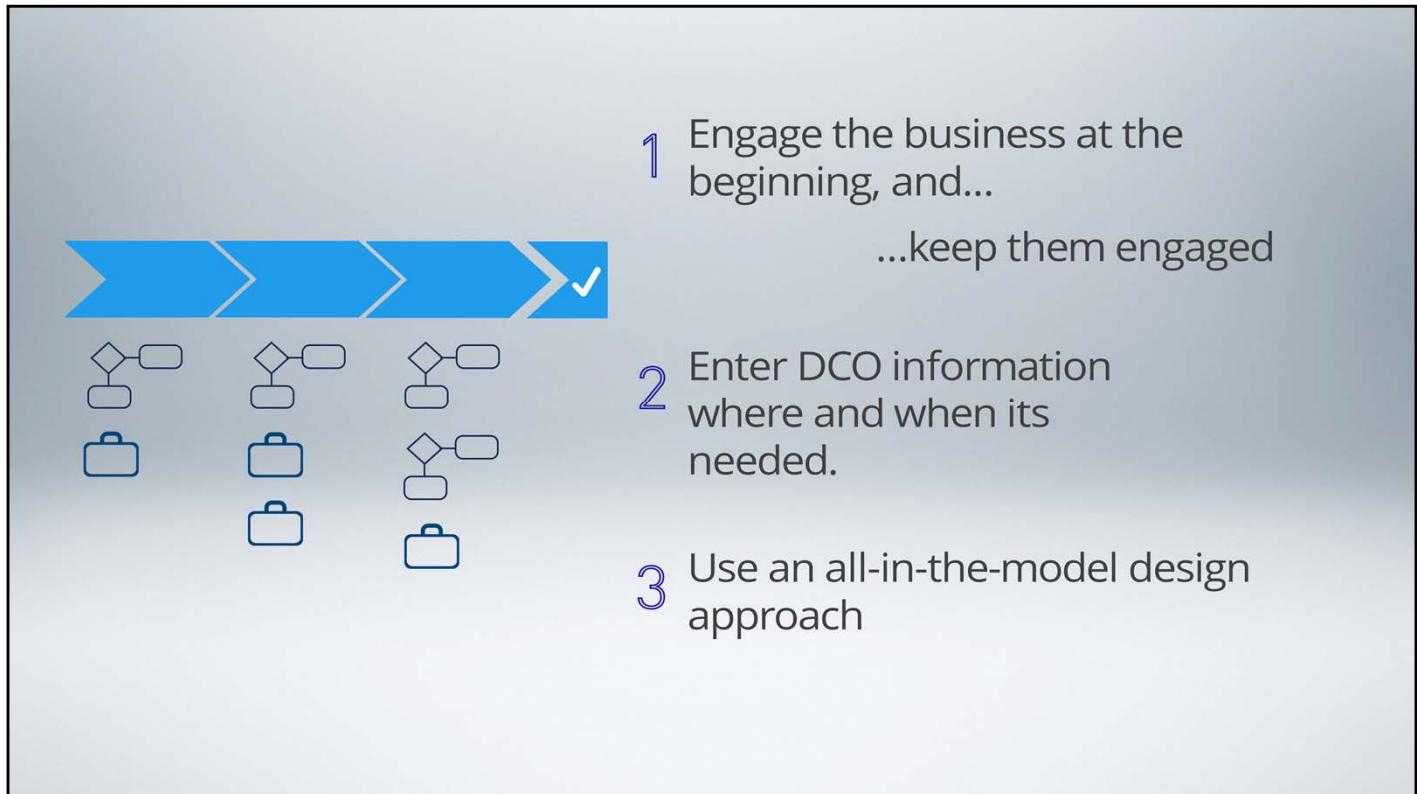
The goal is to always tie business and project goals, objectives, processes, specifications, and requirements to implementations.



- 1 Engage the business at the beginning, and...  
...keep them engaged
- 2 Enter DCO information where and when its needed.

We will see more of, and work directly with, DCO throughout this course.

Using an all-in-the-model shared approach dramatically cuts time to value.



When business people and technologists share an understanding of the business transaction, errors due to misinterpreted requirements are greatly reduced. Remember, if the “big picture” view is how business people think about a business transaction, it should be how technologists interpret and define that business transaction into an easily understood model.



- 1 Engage the business at the beginning, and...  
...keep them engaged
- 2 Enter DCO information where and when its needed.
- 3 Use an all-in-the-model design approach

You had to know this was coming.

Build for Change®.

Your goal should always be to build your solutions in such a way that future changes can easily be accommodated. This effort includes identifying, and delegating responsibility for updating, selected important parts of each application to business managers.

As you make your way through the rest of this course, “Build for Change®” will be a central focus when deciding how best to build a solution for any given business need.

BPM

business transaction

case

subcase

visibility = success

milestones  
stages

DCO

case management  
process

all-in-the-model

changing business conditions

OK, this is a lot of data points: cases, stages, DCO, transactions...

Let's see if I can summarize these data points into a single, cohesive picture.

## In summary...



**case (n):** a business transaction



moves through **stages**



**visibility** into the transaction



express and interpret requirements **consistently**



A case is a business transaction to resolve.

It is seen as work moving through a series of stages

This holistic view provides visibility into the transaction for both the business and technologists

A shared model allows business people and technologists to express and interpret business requirements consistently.

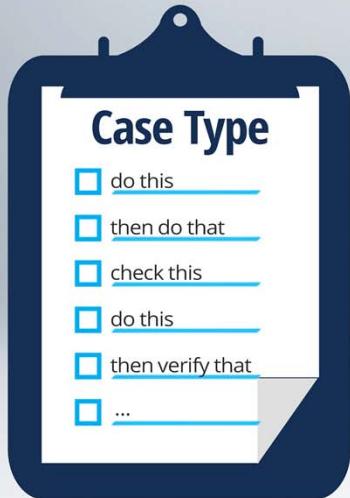
And, finally, the goal should always be to build solutions in such a way that future changes could easily be accommodated.

# Managing Enterprise Apps using Stage-Based Case Design

In this lesson you will learn to define the behavior of an application using case types and stages.

At the end of this lesson, you should be able to:

- Confirm your understanding of case types and stages
- State at least three best practices for defining case types
- State at least three best practices for defining stages
- State the difference between primary stages, alternate stages and resolution stages
- Define the behavior of a business transaction using case types and stages in the Case Designer



***Case Type:*** tasks needed to automate a business transaction (case)

Learning and adopting best practices for stage-based case design - whether they are defined in this course or in your company's center of excellence - will go a long way in making you an effective Pega system architect.

Effective case management requires that individual contributors complete tasks in a coordinated manner so that they can resolve a case efficiently. These tasks can be seen as entries on a checklist, representing objectives to be completed to resolve the case.

In its simplest form, a case type defines the tasks and decisions needed to complete a business transaction.



**Case Type:** tasks needed to automate a business transaction (case)

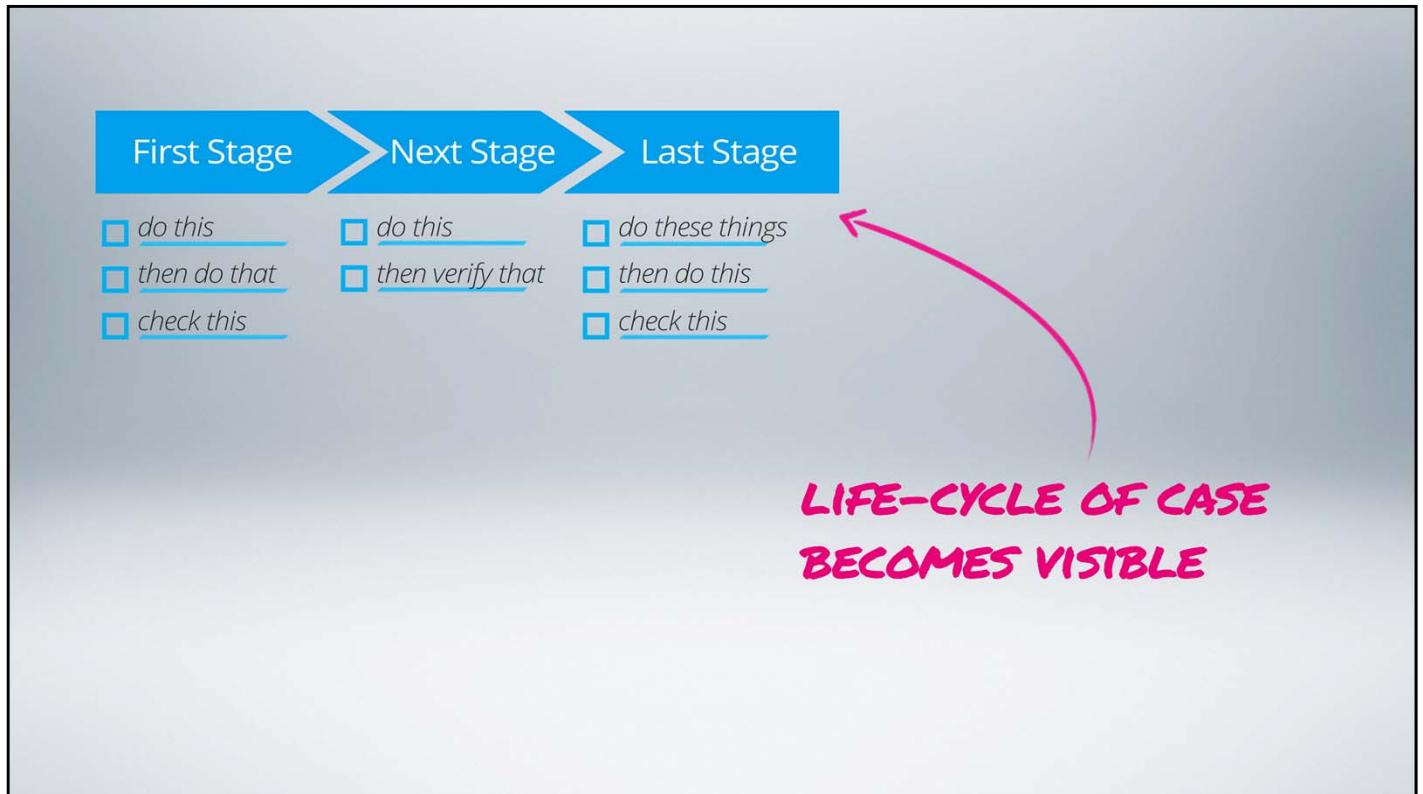
In Pega 7, a case type is a collection of Pega-related artifacts such as data elements, UI screens, processes and sub cases, decisions and integrations used to implement the tasks for a specific case.



**case type:** tasks needed to automate a business transaction (case)

**stage:** first-level grouping for related tasks

Without any context, however, the relationships and dependencies between these objectives are invisible to both application designers and the people working on the case.



A “stage” is a first level of organizing the different tasks required to complete work associated with a case.

Now, these sound like easy enough definitions, but... how do we know where to draw the line?

What, if any, guidelines can we follow to help us ensure we know a case type and a stage when we see one?

Application = Home Construction  
= Construction

## Applications

- Agile solutions for automating work
- Contain related case types
- Named after the solution it provides

Learning and adopting best practices for stage-based case design - whether they are defined in this course or in your company's center of excellence - will go a long way in making you an effective Pega system architect.

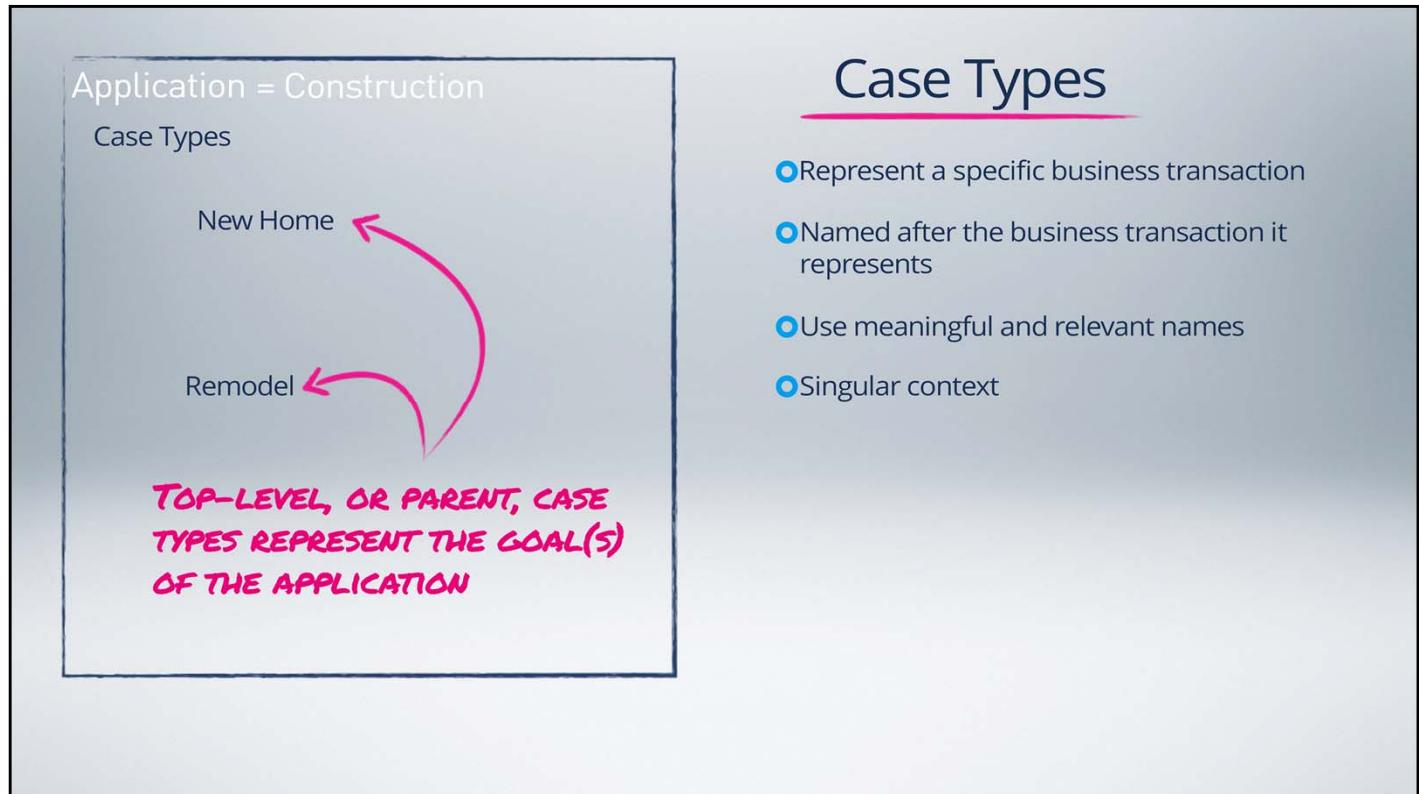
Case types are contained in an application - we will let this box represent the application.

It is highly likely the application you will be working with on your first couple of projects is already defined - as well as the case types for that matter. However, let's go over a few key points about applications and case types.

In Pega, applications are agile solutions for automating work.

Most case management applications handle few to several case types, so the application is usually given a name that is representative of the case types it contains.

Let's consider a rather non-traditional example - a home construction company which wants to automate the management of their construction projects. The business solution is to automate how they build a home so our application would probably be named something along the lines of "Home Construction." or, maybe, simply, Construction.



## Case Types

- Represent a specific business transaction
- Named after the business transaction it represents
- Use meaningful and relevant names
- Singular context

Now, before we start identifying case types, let's set some ground rules.

Case types represent a business transaction we want to automate, so, case types are generally named after the business transaction they represent.

Use names that are relevant and meaningful to the business users.

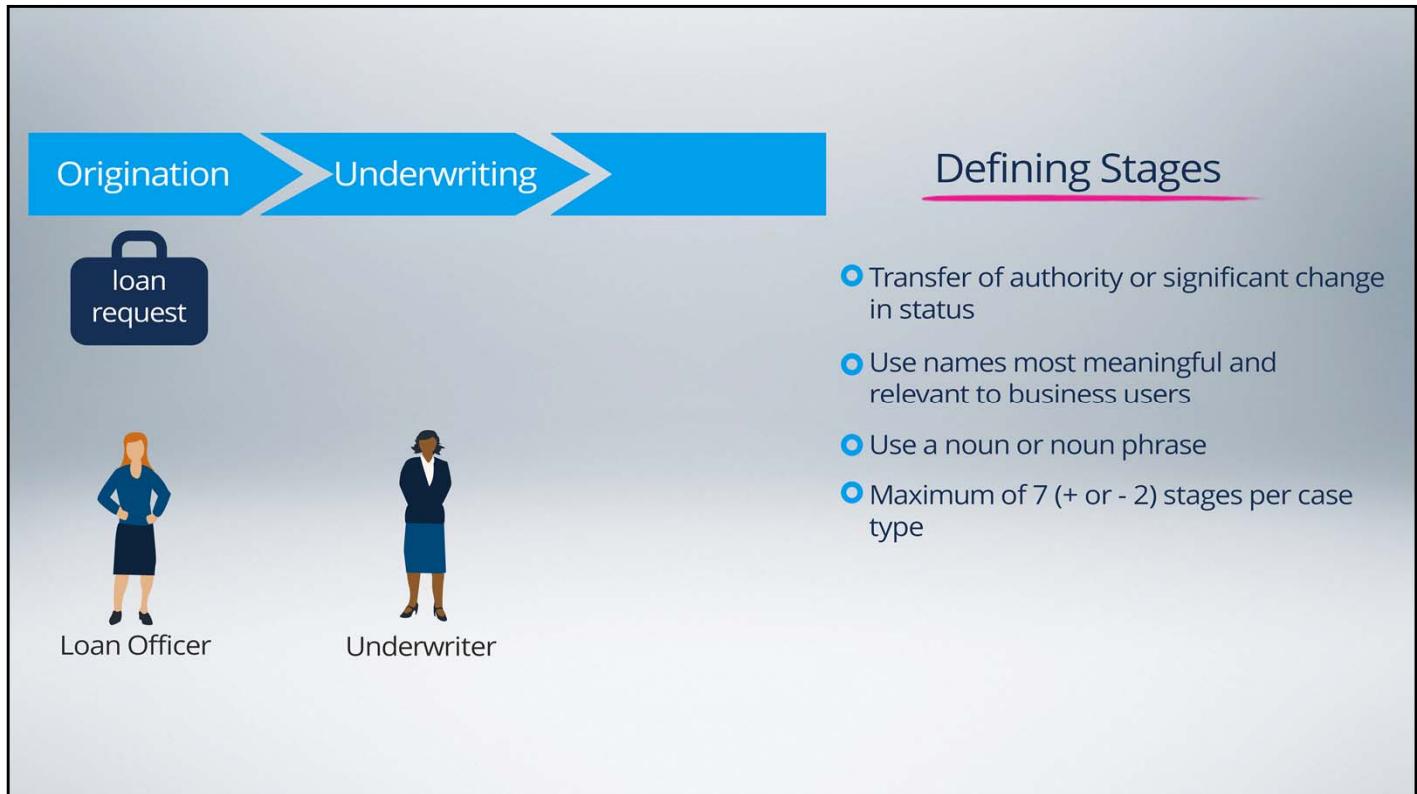
If the application has a meaningful and relevant name, the name given to the case type can be short - usually only one or two words - as the context is provided by the application name.

Also, given that a case type represents a singular business transaction, we typically use a singular context.

When working to identify case types for an application, start with the top-level case types - which should represent the goal, or goals, of the application.

In our construction company, we want to automate the projects for how we build new homes and remodel existing homes. These become the top-level, or parent, case types.

Do not worry about subcases at this point in the discovery effort. It is best to start with the goals of the application - the top-level cases – and then as you gain an understanding of how the application should work, the subcases are much easier to identify.



Before we work to identify the stages in our case type, let's establish some guidelines for doing so.

When it comes to identifying and naming stages,

business stake-holders usually already have these identified and named.

Do not be surprised - for sure don't be offended - if your effort comes down to just typing in what they tell you. But... it is important to understand how stages are identified and named so you can participate in the conversation if necessary.

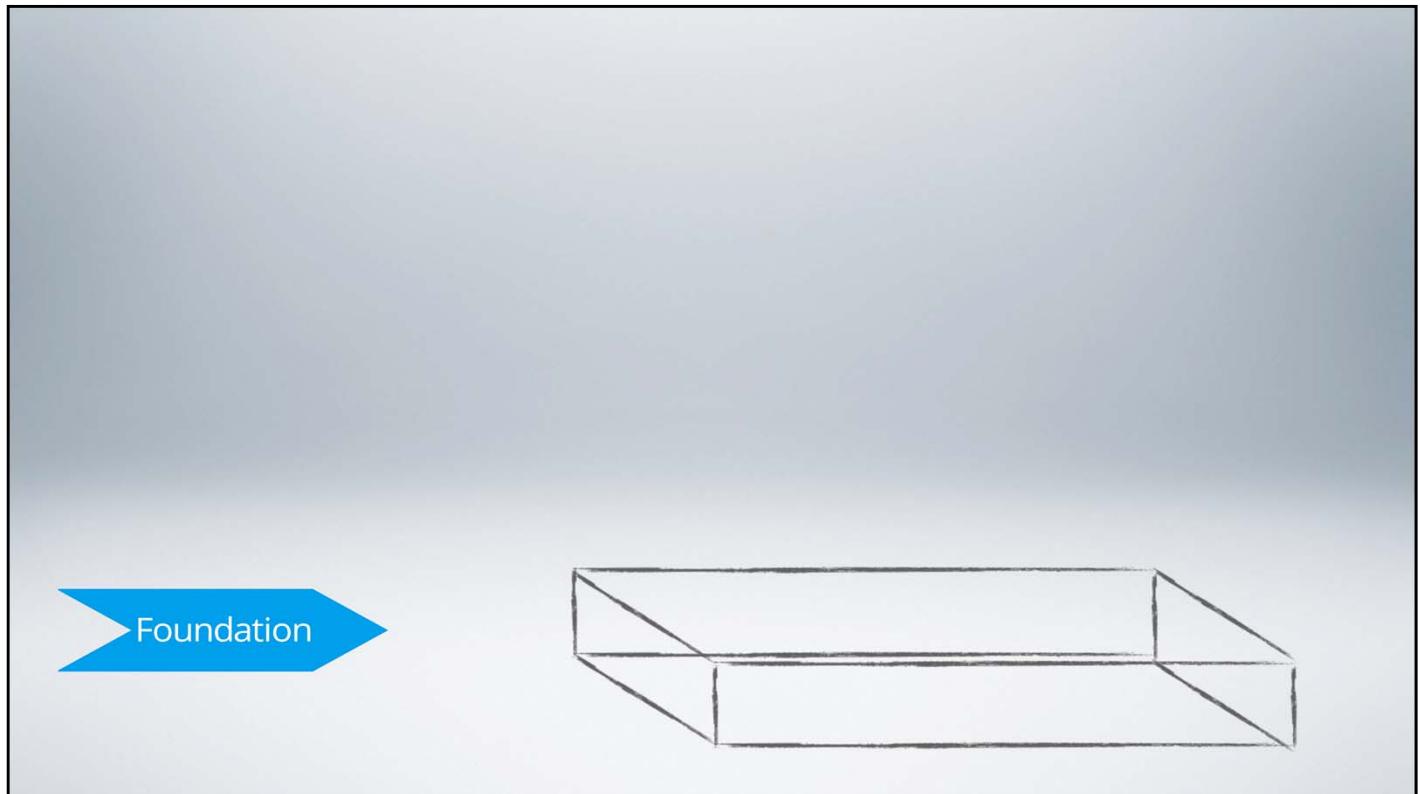
Stages typically represent the transfer of the case from one authority to another, or from one part of the organization to another; or maybe a significant change in the status of the case.

Here, in particular, use names that are most meaningful and relevant to the business users.

Use a noun, or noun phrase, to describe the context of the stage. As much as possible, try to limit the stage name to no more than two words.

Consider limiting the number of stages in any given case type to 7 (plus or minus 2).

If you find yourself needing more than 10 or so stages, it is possible you have not decomposed the case type into a small enough chunk. On the minimum side, do not be concerned if a case has only one or two stages. Focus on maintaining a maximum number of stages in any given case type and the minimum will work itself out.

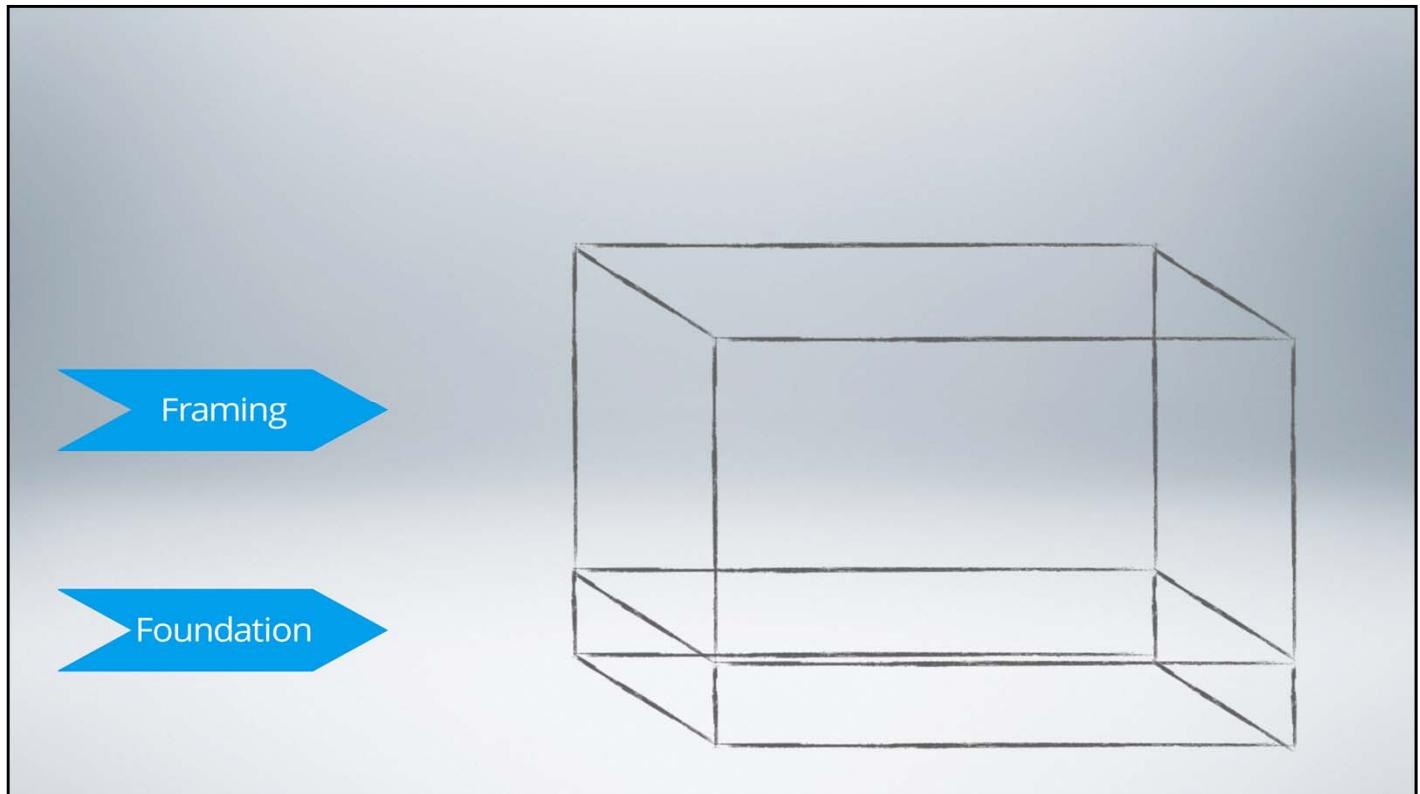


Let's explore how to identify the stages in the building of our new home.

Remember, stages are a high-level grouping of tasks - the first level of case decomposition. To keep this example simple, let's focus specifically on the new construction of our house.

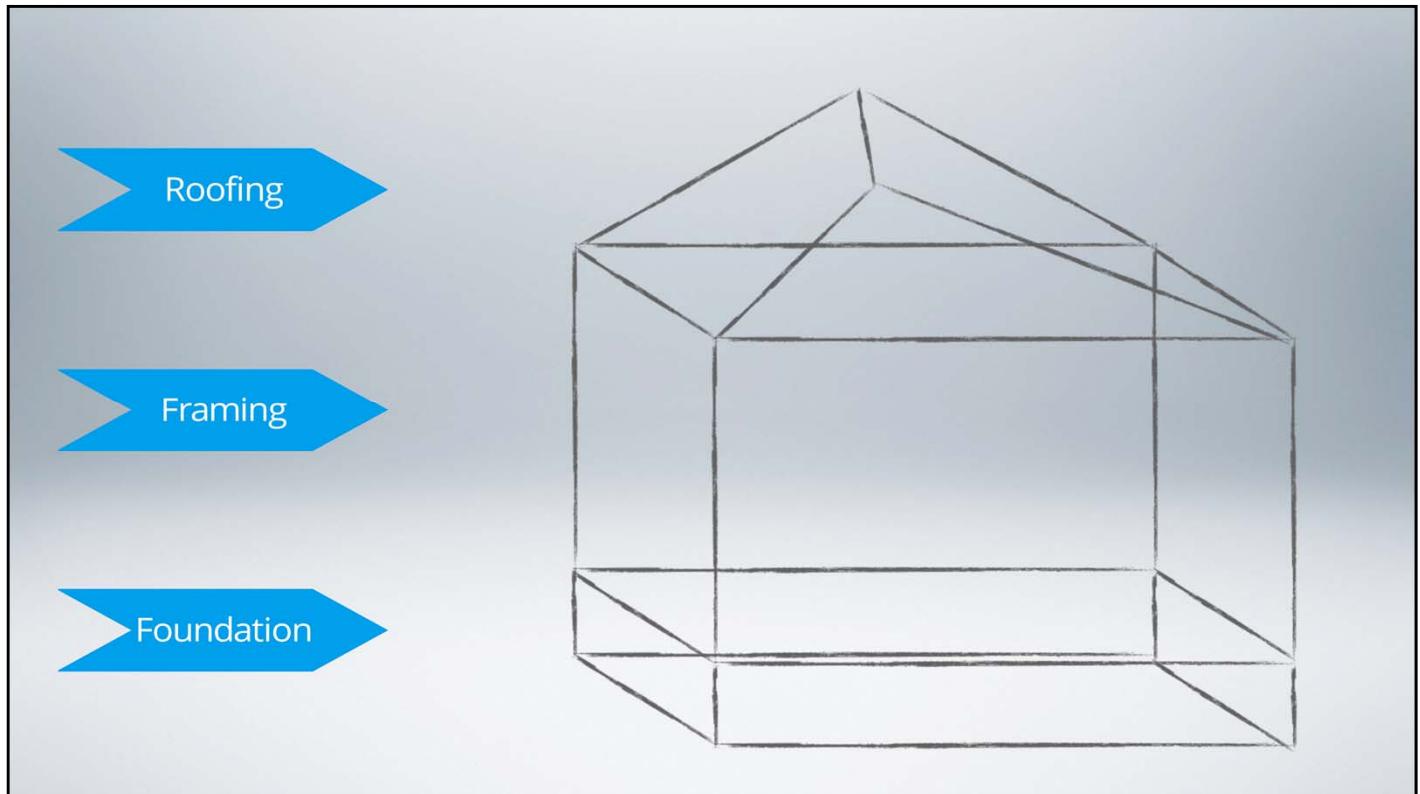
Because we are building our house from the ground up, it seems the first major goal would be to build the foundation.

There might be a lot of little things - the tasks, or steps - we would need to complete to make that happen, but building the foundation is definitely the first major goal we would need to complete to build our house correctly.



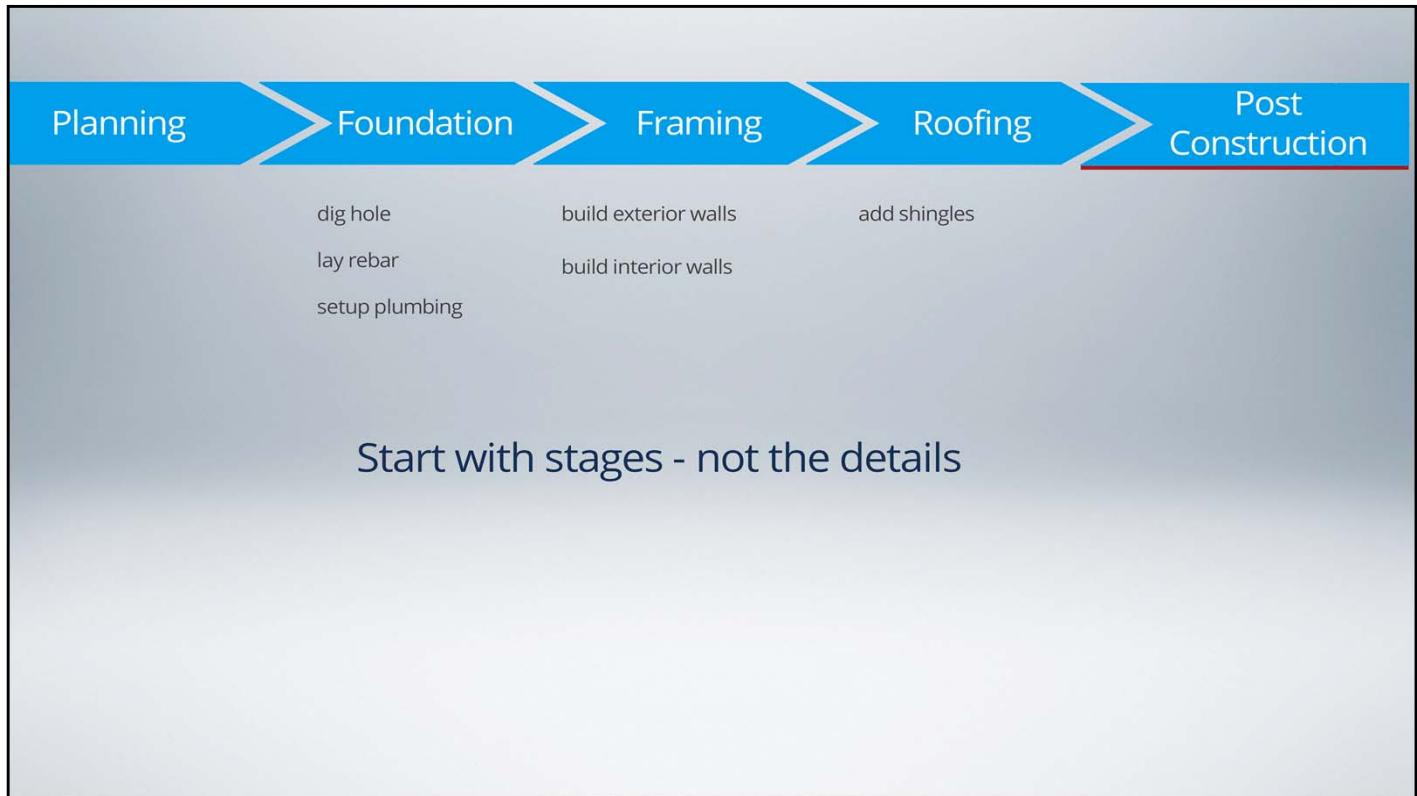
Once the foundation is complete, we would then most probably build the frame of the house.

Here again, there are a good number of tasks that must be completed to construct the frame of the house but, when defining stages, our goal is to simply outline the construction case at a high-level - get it? - "outline?"



At this point in our discovery we are not concerned with identifying the individual tasks necessary to construct the frame, or the foundation for that matter - we are simply after a high-level understanding of what it takes to build our house.

Finally, we would need to build the roof.



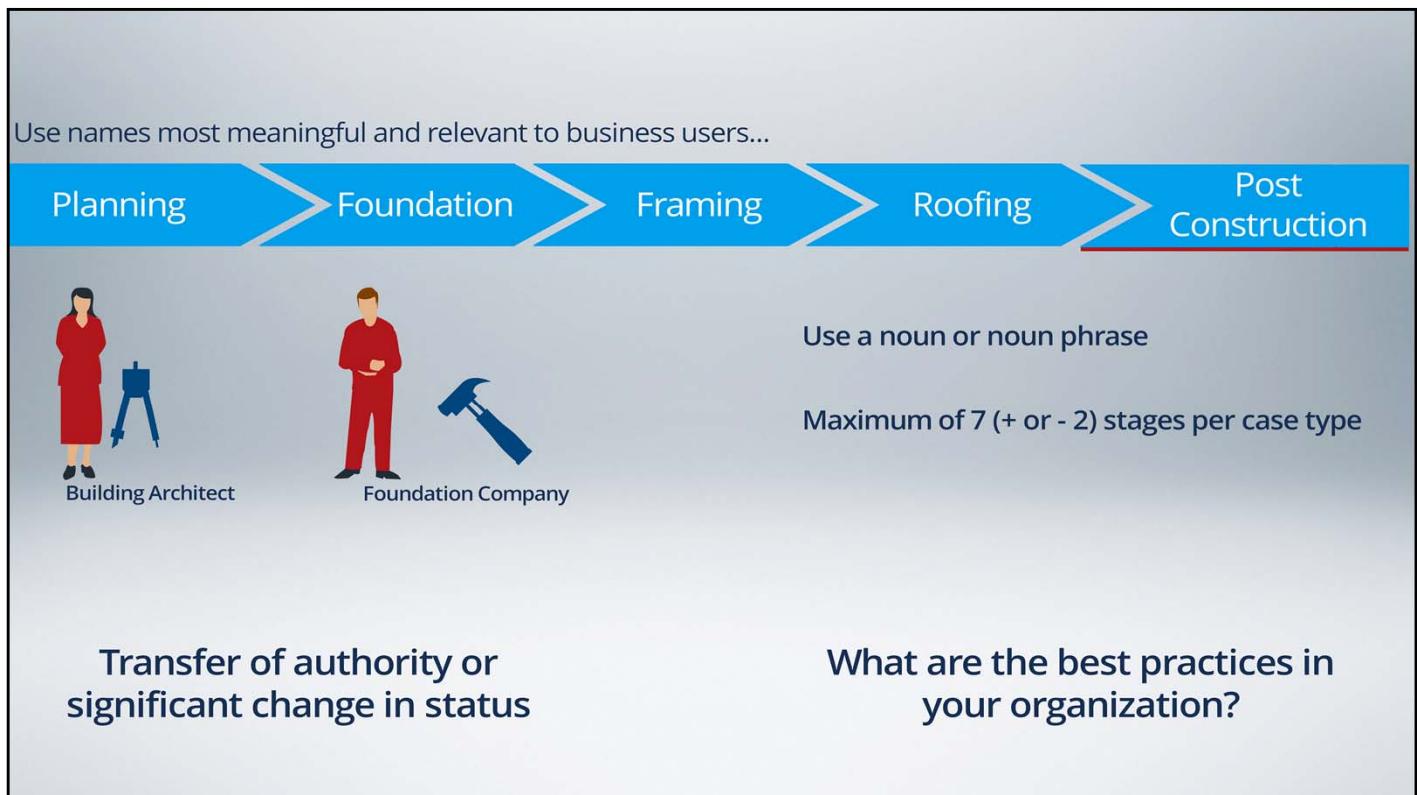
There you have it - the primary stages for building a new house.

Allow me to purposefully point out what is missing. At this point of our discovery, we might not know any details about how the house is actually built - and that is OK.

All we know is, in order to build a house, we would complete the foundation, then the frame, and only then could we add the roof. This high-level abstraction can prove very useful in helping bring clarity to a business transaction cluttered with years of work-arounds and personal preferences.

As we close this example, allow me to recognize that, yes, there are some things that must be completed before the construction of our home begins, and there are certainly some things to do after the construction of the house is complete.

And..., do not be surprised if it happens this way – that you seem to define stages in a non-sequential order. It may take a few passes to define the right stages in the right order.



One last point – and probably the most important point of all. There is no real right or wrong way to define stages. You may define a very different model to represent the stages for building a house. The goal should be to establish best practices for doing so, then follow those best practices.

Let's review our best practices for stages. Stages typically represent a transfer of authority or significant change in status

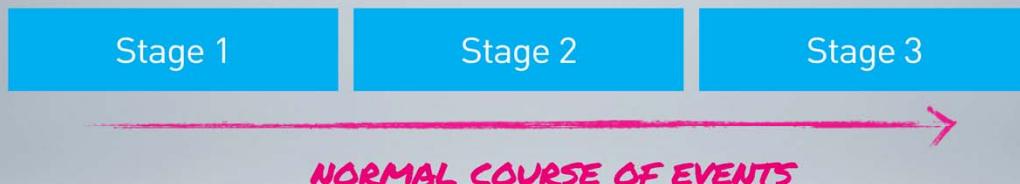
Try to name the stages so the names are meaningful and relevant to business users.

Use a noun or noun phrase – limiting the stage name to no more than two words

And, as much as possible, set a limit to the number of stages for any given case type. We suggest using the “Rule of 7.”

Finally, it is important to ask: What best practices are defined in your organization? Knowing, and following these best practices will help you make the most of your case management design efforts.

## Primary path stages



When defining the stages for any given case type, the goal is to define primary path stages, alternate stages and resolution stages.

“Primary path” stages are stages the case would pass through in a “normal course of events” and are generally free of exceptions.

Let’s look at another example case type – that of an expense report. I suspect this may be a case we are all very familiar with.

## Expense Report Stages

Submission

Review

Payment

NORMAL COURSE OF EVENTS

OR  
HAPPY PATH

Oh! And....as always, let's keep this example simple so we can focus on the concept of "primary path" stages and not get side tracked on how to define an expense report case.

An expense report might have as its primary path stages a "Submission" stage where the person submits their reimbursable expenses, a "Review" stage where a department manager would review the expense report for accuracy and adherence to expense policies, and a "Payment" stage where an approved expense report would be processed for payment.

In our business example, these are the stages we know an expense report will always pass through. This "normal course of events" is sometimes referred to as the "happy path."

But..., what happens if the expense report does not get approved? Or...maybe an audit is necessary because some business expense rule is violated.

## Alternate Stages

Stage 1

Stage 2

Stage 3

**SPECIAL CIRCUMSTANCES OR EXCEPTIONS**

This is where “Alternate” stages come in.

“Alternate stages” are those stages that are not part of the “normal course of events” but must be available under certain circumstances – or exceptions to the normal course of events.

## Expense Report Stages

Submission

Review

Payment

AUDIT REQUIRED AFTER REVIEW IF  
"THESE CIRCUMSTANCES" ARE MET

Let's go back to our expense report.

Let's say our business policies dictate that every expense report is reviewed by at least the department manager, but an audit is required after the review when certain spending thresholds are crossed or the expense report is customer billable.

## Expense Report Stages

Submission

Review

Audit

Payment

SEEMS TO BE PART OF THE  
"NORMAL COURSE OF EVENTS" -  
BUT IT IS NOT!

We could place this "Audit" stage in the primary path and use a "Skip stage when" condition, but that presents a number of challenges not the least of which is we lose visibility, and with that comes loss of context.



A more accurate representation of the case would be to display the “Audit” stage as an alternate stage.

Now, it becomes very clear that an “Audit” – because it is defined as an alternate stage – is not part of the normal course of events and only happens under certain circumstances. We may not know what those circumstances are – and that’s OK. The point of the discovery map is to represent the big picture view of a business transaction – not the details.

## Alternate Stages

A Stage

A Stage

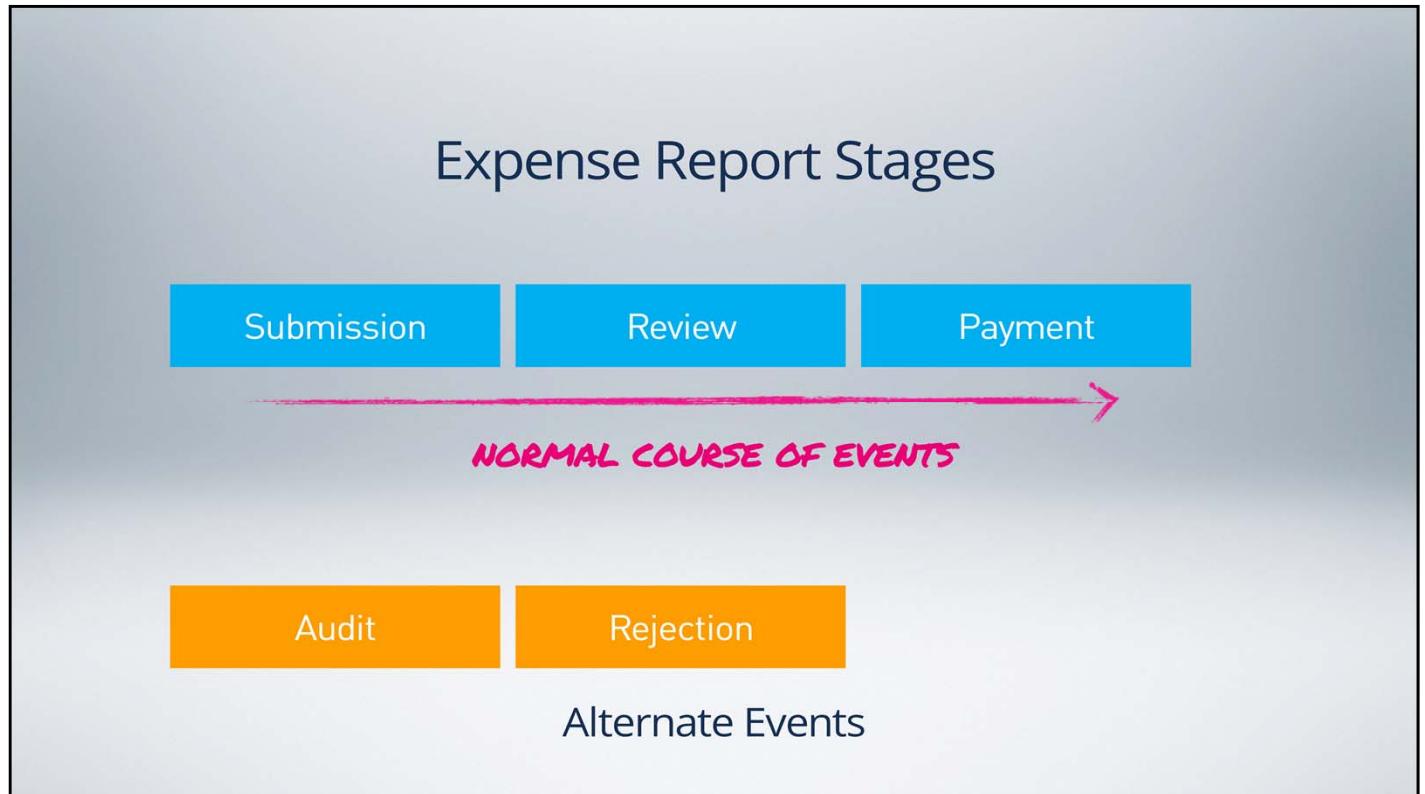
A Stage

Special circumstances or exceptions

Must be manually called

Cannot be sequenced

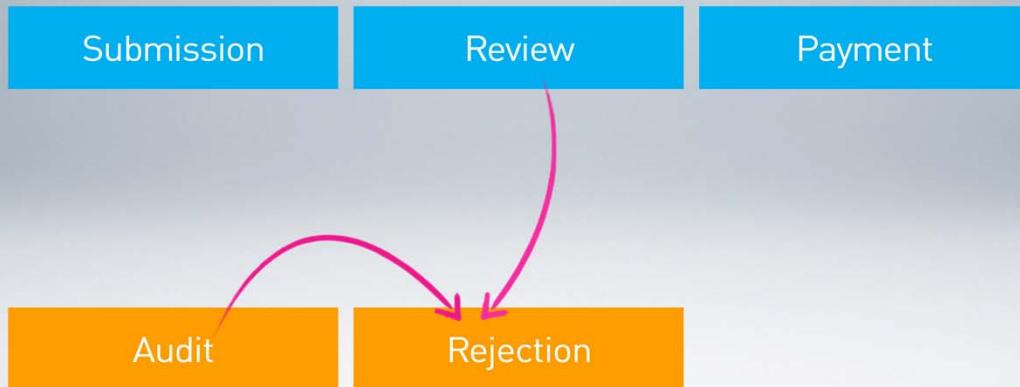
Now, I need to make one very important point about alternate stages. Alternate stages can only be accessed manually, and therefore have no sense of sequence relative to other stages – there is no sense of being 'next'.



And, if you have more than one alternate stage, there is no sense of order in the way they are presented. Let's add a "Rejection" stage to our expense report case type to handle what happens when an expense report is rejected.

Again, an expense report getting rejected for any reason would not be the "normal course of events" so we would add it as an alternate stage

## Expense Report Stages



Let's say our business policies allow that the expense report can be rejected in either the "Review" stage or the "Audit" stage.

The order in which we present the two alternate stages does not matter. We can list the alternate stages in the order I have presented here

## Expense Report Stages

Submission

Review

Payment

Rejection

Audit

or we can present them in this order. It simply does not matter. Alternate stages never imply a sense of order.

## Resolution Stages

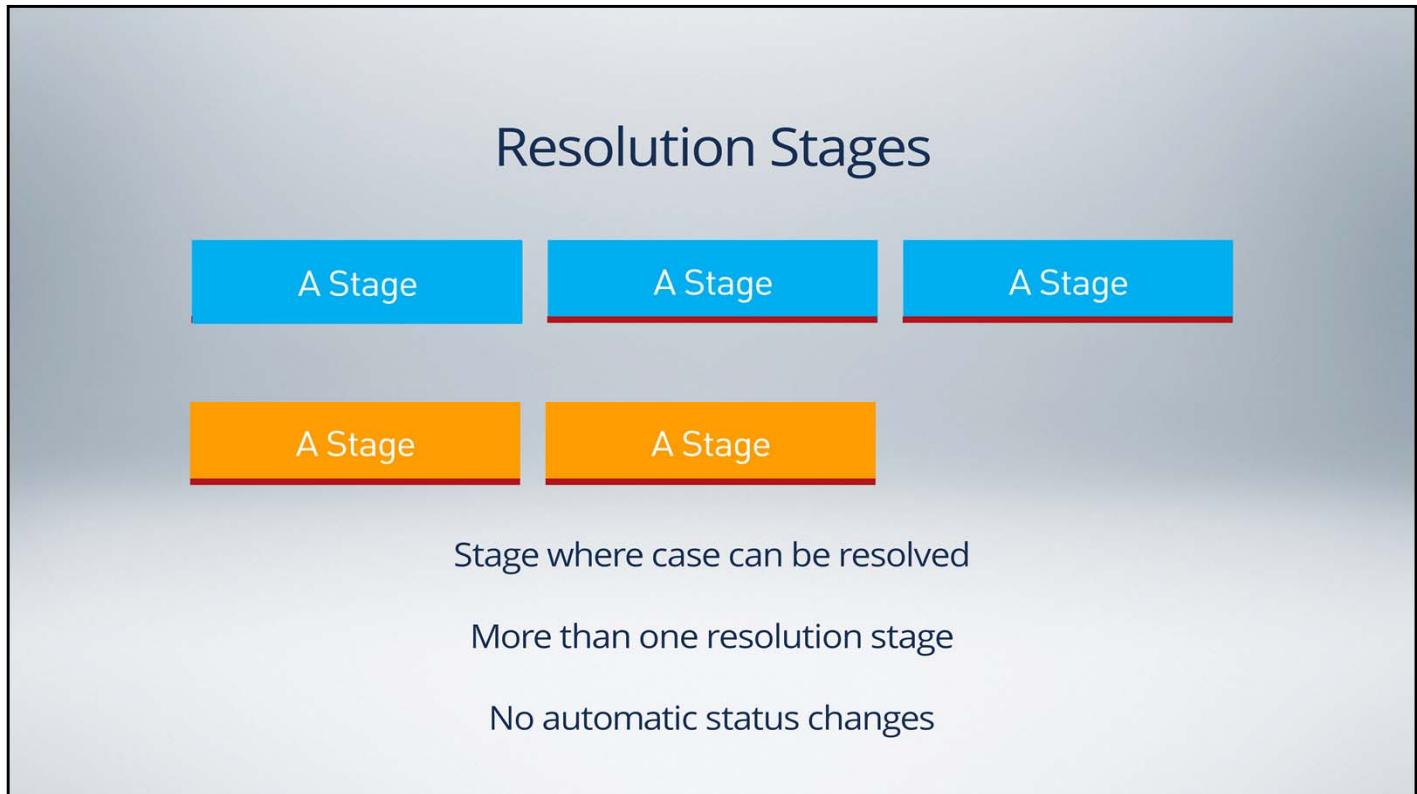
A Stage

A Stage

Stage where case can be resolved

This brings us to “Resolution” stages – which...are not a stage “type” but a stage “behavior,” or “classification.”

As you can see, both primary path stages and alternate stages can be defined as a “resolution” stage. Classifying a stage as a “resolution stage” provides a visual indicator that a case can be resolved, or closed, in that particular stage. This can be useful – someone who doesn’t know the process very well can see, and understand that is where the case can come to a close.



You can also have more than one resolution stage in the primary path or alternate stages. This may get a bit confusing, so use multiple resolution stages very carefully. The measurement should always be: “Does the discovery map provide clarity and understanding of the case?”

It is important to note that a resolution stage has no functional behavior – other than setting the stage to a manual transition. Changing the status of a case, or closing the case completely must still be specifically called for in a step in the stage.



Let's go back to the expense report case type.

Our business policies indicate there are two places the expense report case may come to a close: the "Payment" stage and the "Rejection" stage.

Classifying these two stages as "Resolution" stages makes it readily apparent where the case may come to a close.



One last point to make about primary path, alternate, and resolution stages.

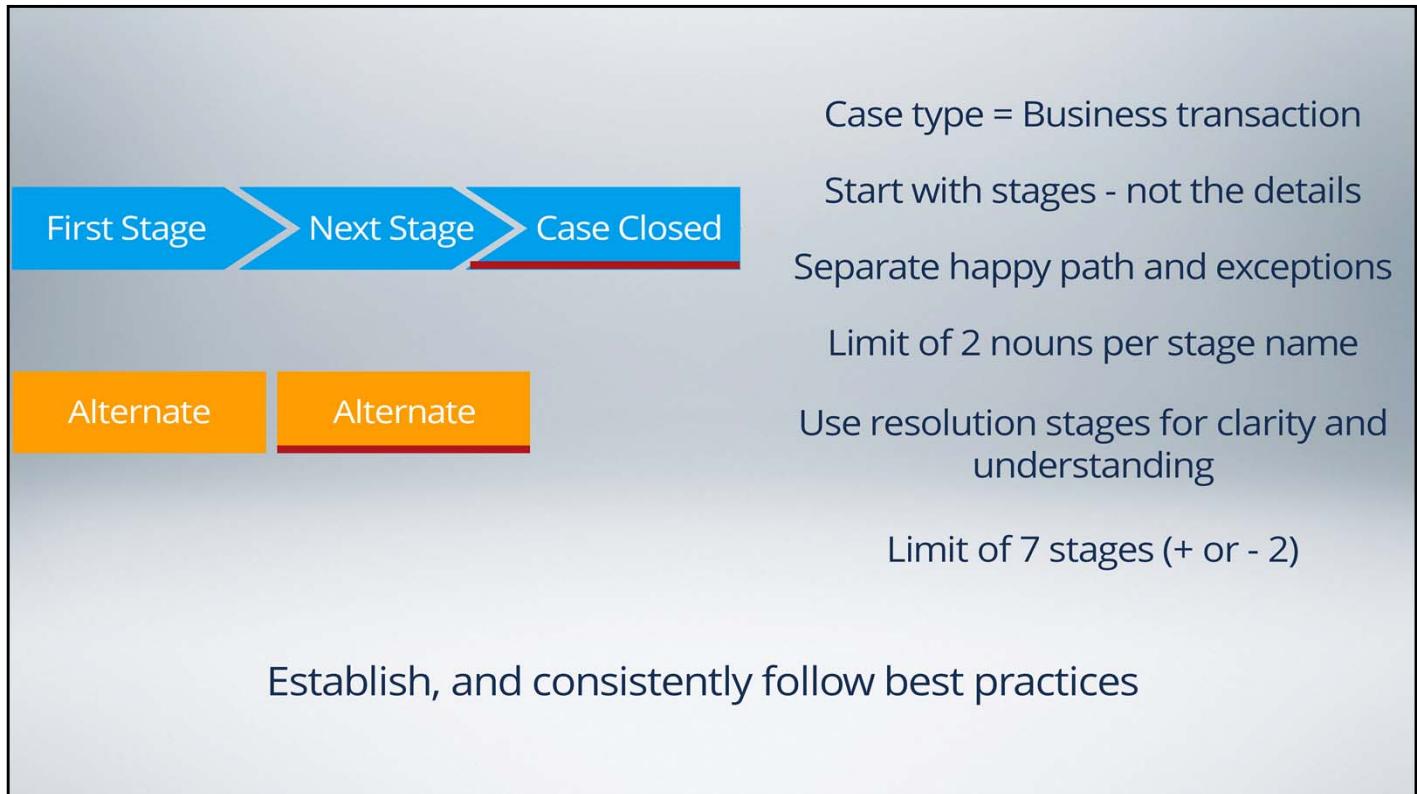
A primary path stage can be configured to automatically transition to the next primary stage when all steps in that stage are complete or, set to manually transition to another stage by an end user or as defined in a process model.

Alternate and resolution stages can only ever use a manual transition to a next stage.

Now, I recognize this is a lot to take in all at once. The good news is there is a “Legend” available. You can access this legend from the “Actions” menu in the upper right corner of the Case Designer.

Click “Actions” and then “Legend” from the drop down menu.

The types of stages and their icon are presented on the left side.



Let's see if I can summarize this lesson into a nice, easy to read picture.

Case types represent, and are named after, a specific business transaction.

Start with identifying the stages – not the details of the case.

Remember to separate the “happy path” from the exceptions.

Use nouns, or noun phrases, when naming stages – and try to avoid using more than two words.

Classify a stage as a “resolution” stage to indicate the case can be closed in that stage.

Consider limiting the total number of primary stages to seven, plus or minus two.

Now, these guidelines are not chiseled in stone. Your organization may adopt, or adapt, these guidelines. It is also quite possible your organization may have very different guidelines. The important take-away is this:

Establish best practices. This keeps everybody on the project working and thinking in the same direction. Most importantly, when building a case management application, be consistent. Regardless of the guidelines, be consistent.

## **Exercise: Define the behavior of a business transaction using case types and stages**

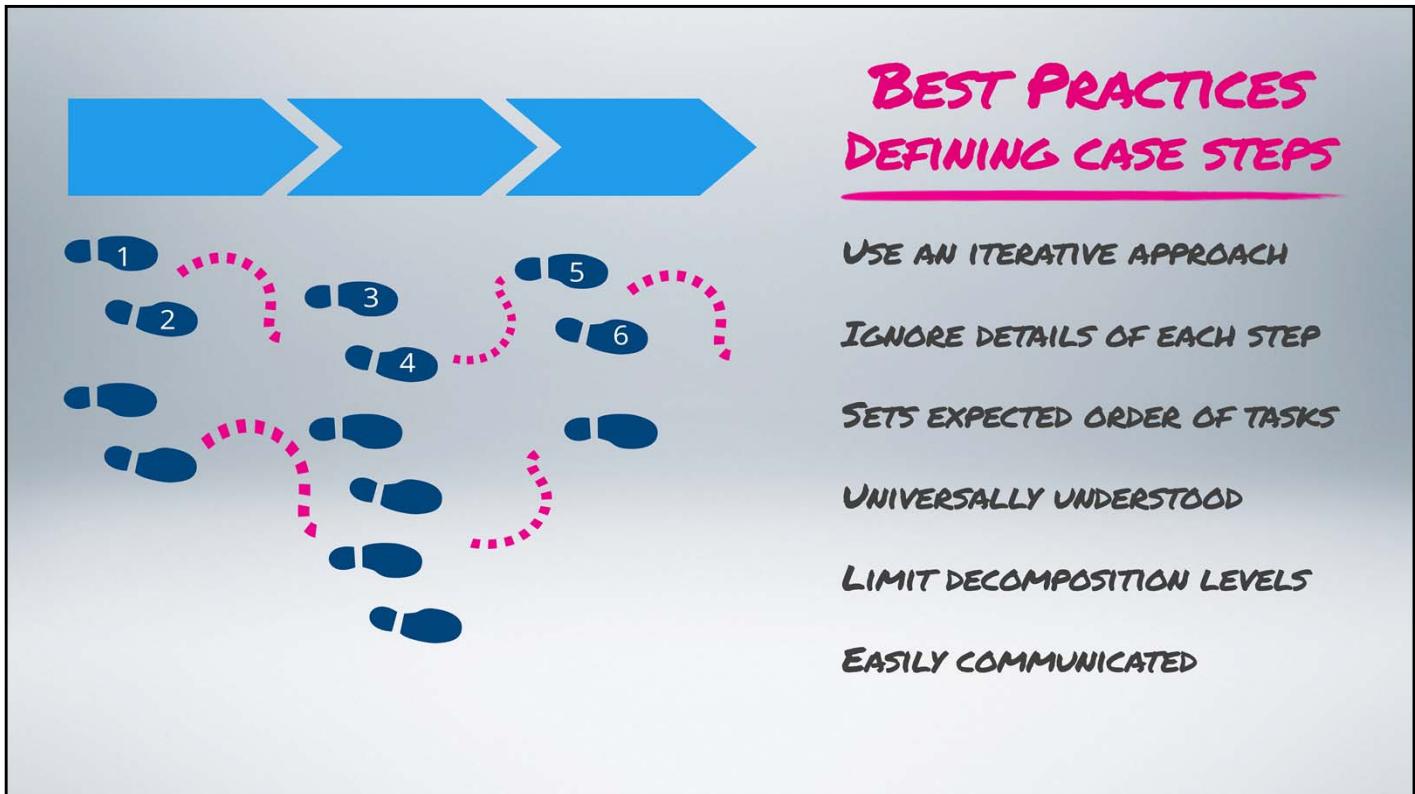


# Best Practices for Effective Case Decomposition

In this lesson, we explore best practices for applying criteria to determine when a given business use case would be best defined as a case, process or subcase.

At the end of this lesson, you should be able to:

- State at least three best practices for defining steps in a case
- State the use cases for using a Single Step Assignment, a Multi-Step Process and a Subcase when decomposing a case
- Decompose a business scenario into the appropriate steps and step types



Now that the stages for our case are in place, we can turn our attention to a further level of case decomposition.

It is time to identify the tasks, or “steps,” needed to complete, or resolve, the case.

But...as always, let’s set some ground rules for doing so. What “best practices” can we use to ensure we are consistent in defining the steps necessary to resolve a case?

Consider using an iterative approach when defining the steps of the case.

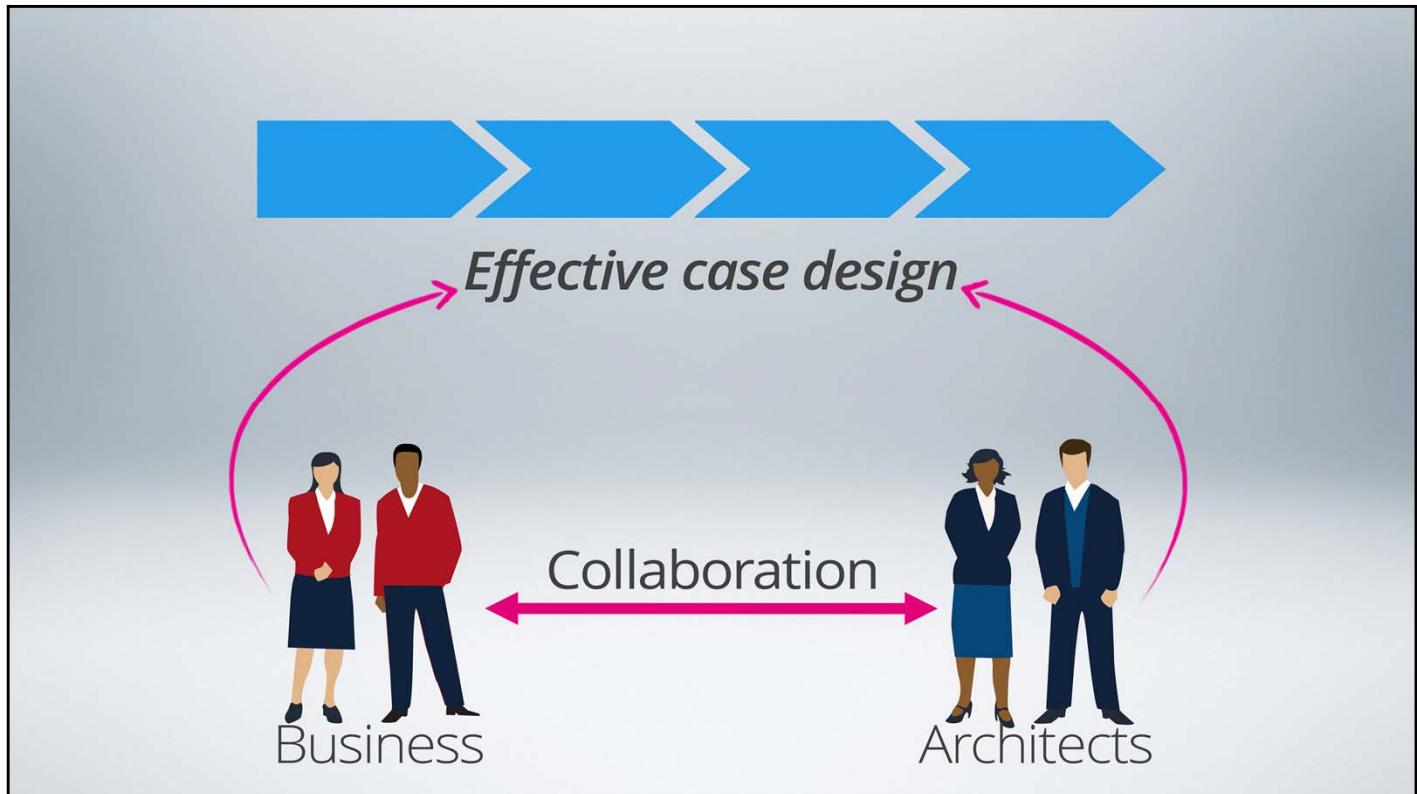
The first iteration should be to simply identify the steps needed – the checklist of things to accomplish – to resolve the case. We should not be worried about specific details such as what data to collect or what decisions, if any, are needed.

This initial discovery effort allows us to build a case map that communicates the expected order of the tasks throughout the life cycle of the case.

Both business people and technologists should universally understand this high-level map.

Keep an eye on the number of steps in any given stage. While there is no specific number of steps for any given stage, consider using the “Rule of Seven” here as well. This guideline suggests that no dimension of a case should be sub-divided more than seven times, plus or minus two.

Finally, at any level of granularity, the case should be easily communicated. If we follow best practices our case model should be explainable in five minutes or less.



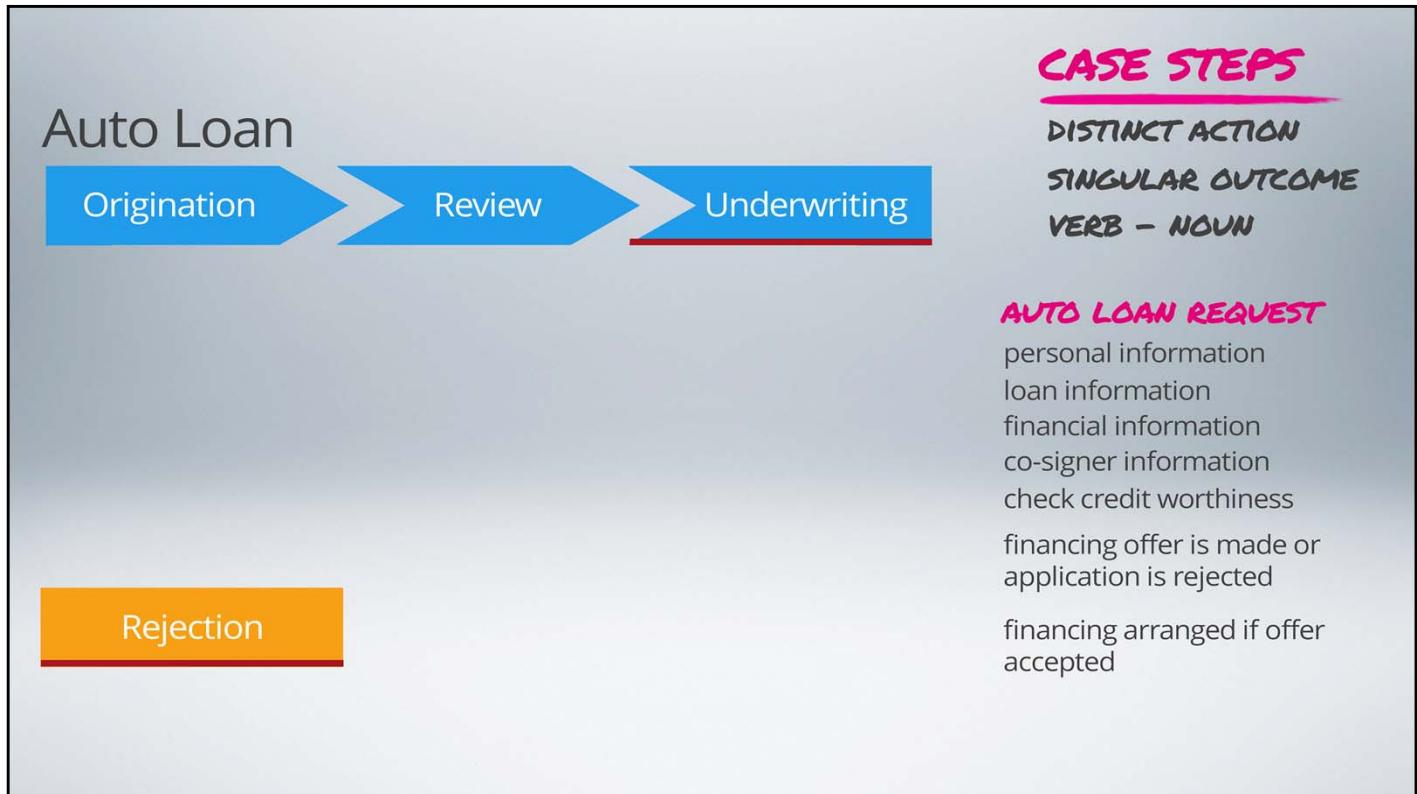
Oh, and one more thing.

As with any case design effort, defining the steps in a case is not something you would do on your own, in an information vacuum.

Engaging the business users, and other stake-holders, directly in the development process is critical.

Effective case management design is best accomplished when both the business stakeholders and the development team contribute.

Collaboration must be considered a best practice.



Let's turn our attention to understanding how to know a "step" when we see one.

Consider a step to be a distinct action taken to help resolve a case

and should have a goal that can be expressed as a singular outcome.

When naming steps, use a "verb plus noun" naming convention - perform "this action" on "this object."

OK, those guidelines sound simple enough. Let's look at an auto loan request and see these guidelines applied.

And, let's keep the example simple so we can focus on learning how to identify steps at the right level of granularity and not how to actually define an auto loan case. We have already defined three primary stages, and one alternate stage, for the auto loan case.

An "Origination" stage where the loan application is filled out,

A "Review" stage where a loan officer reviews the loan,

and an "Underwriting" stage where an approved loan application is processed for financing.

There is also an alternate "Rejection" stage in case the borrower doesn't get approved for the loan.

Our goal is to identify the steps necessary to transact the auto loan and place those steps in the right order in the right stages.

The business users tell us that, in order to transact an auto loan request, the borrower must complete a loan application, which contains the borrowers personal information,

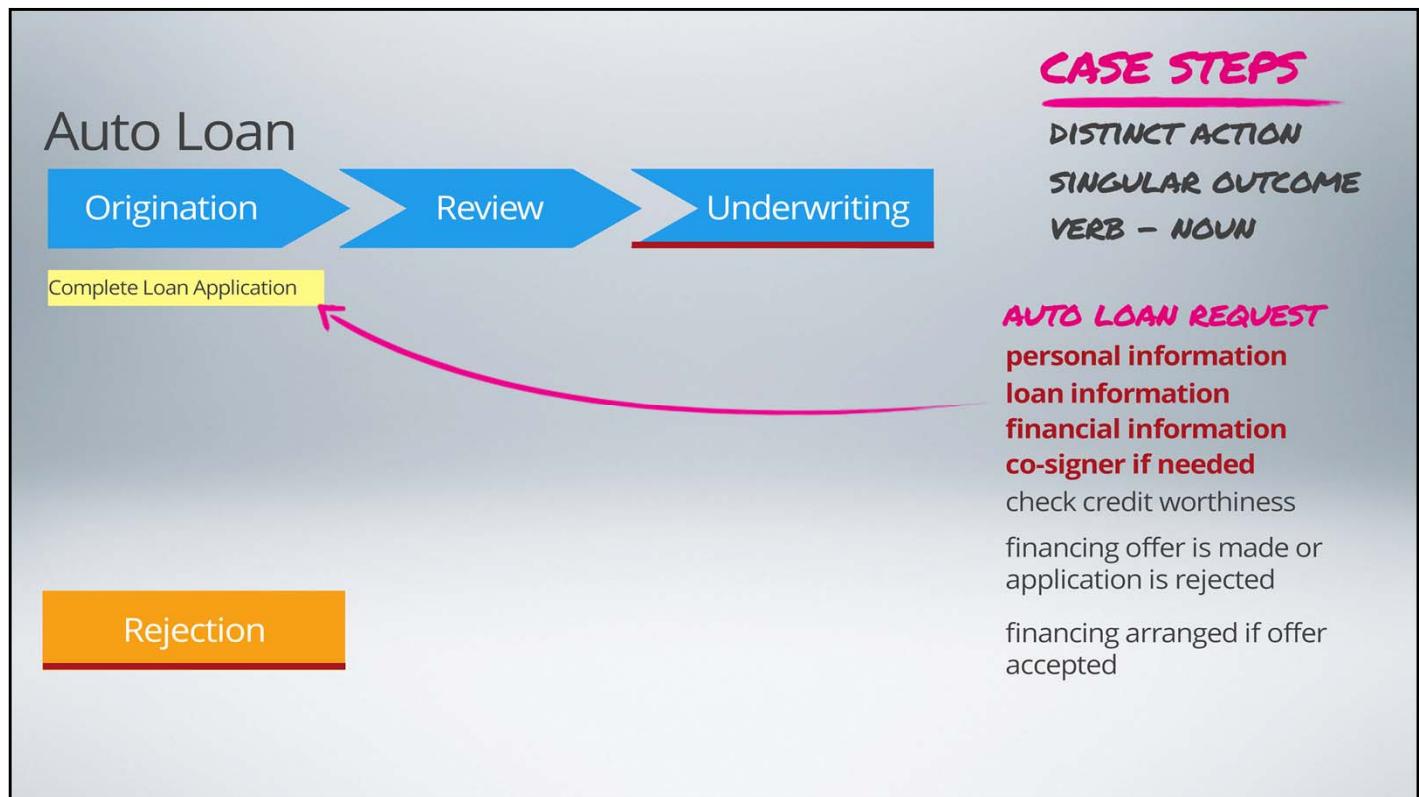
information about the vehicle such as year, make and model, and the cost,

the borrower's financial information such as assets and liabilities, and co-signer information if required.

The borrower's financial information is then evaluated by a loan officer, along with a check of their credit score. If a co-signer is required, then the loan officer also evaluates the co-signer's financial information and credit score.

If the borrower's credit worthiness is considered acceptable, a financing offer is made. Otherwise, the loan application is rejected and the case is closed. If a co-signer is required, we will also want to evaluate their credit worthiness against the same criteria.

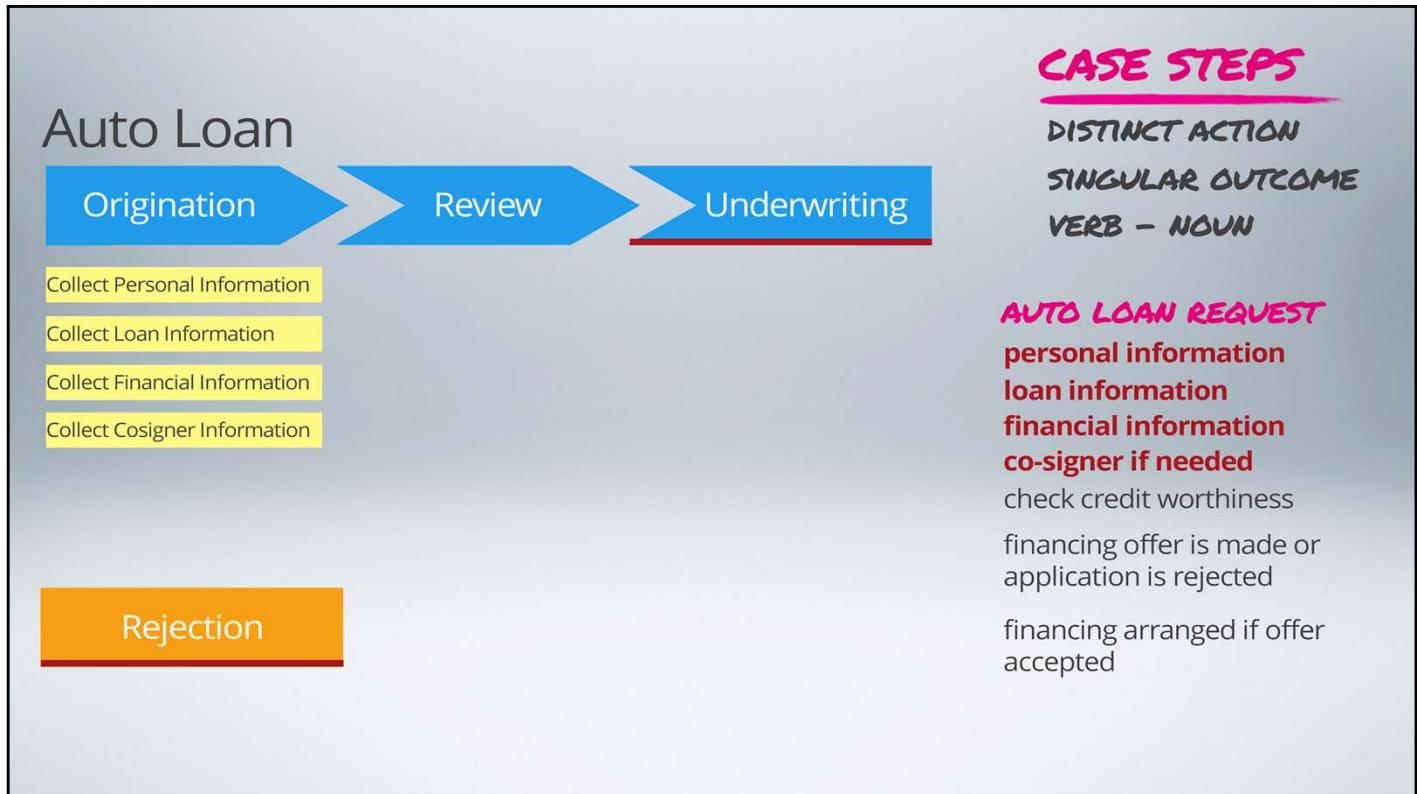
If the borrower accepts the financing offer, the loan is financed. However, the borrower can reject the financing offer and, if they do, the loan application case is closed.



Let's start the case decomposition with the loan application itself.

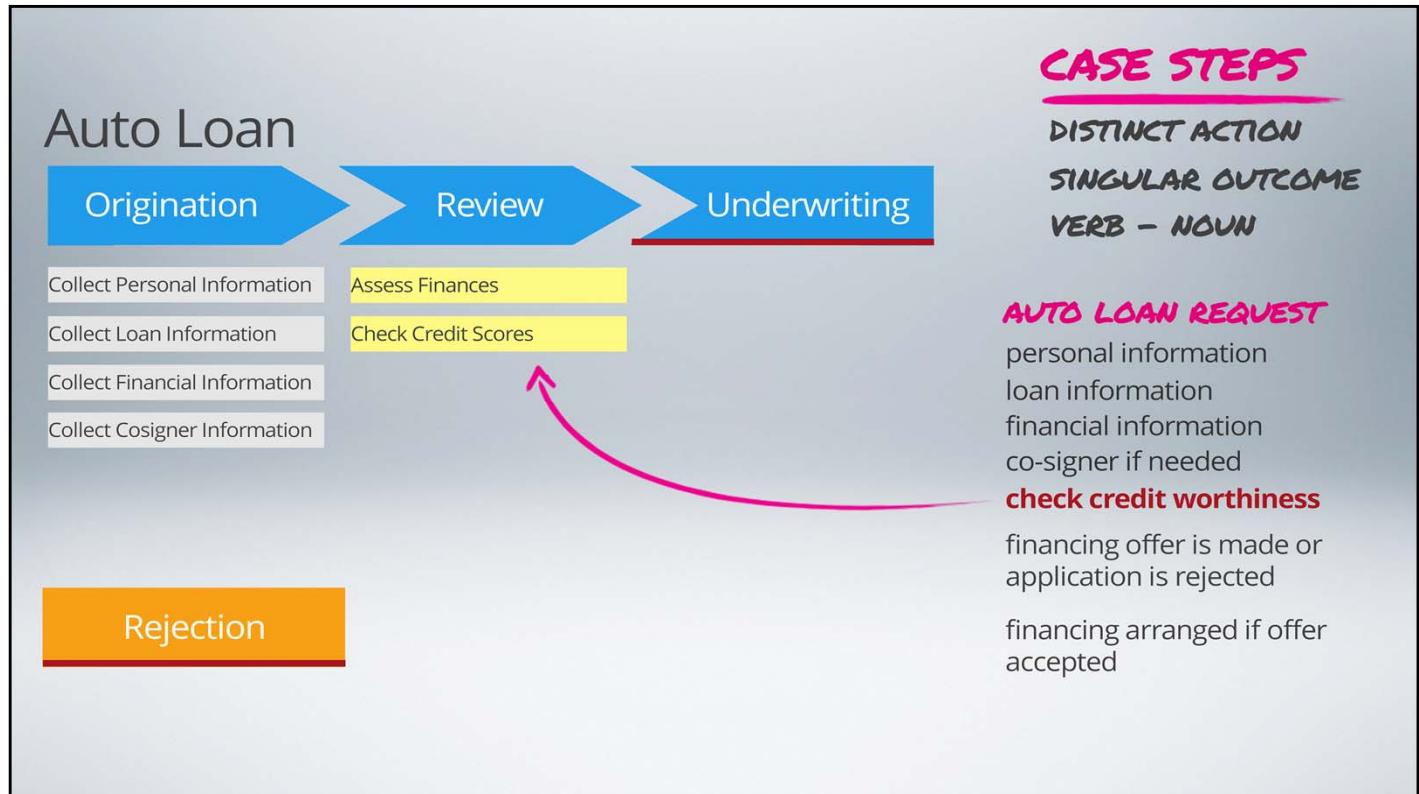
Remember, a "step" is a distinct action that produces a singular outcome.

We could consider completing the loan application to be the first, and possibly only, step in the "Origination" stage. This is clearly a distinct action with a singular outcome – a completed loan application.



However, if the desire is for more visibility into how, exactly, we complete a loan application, we could use separate steps for each part of the loan application.

Both designs work, and both have trade-offs – which we will discuss shortly. But for now...let's continue the decomposition of the auto loan case.

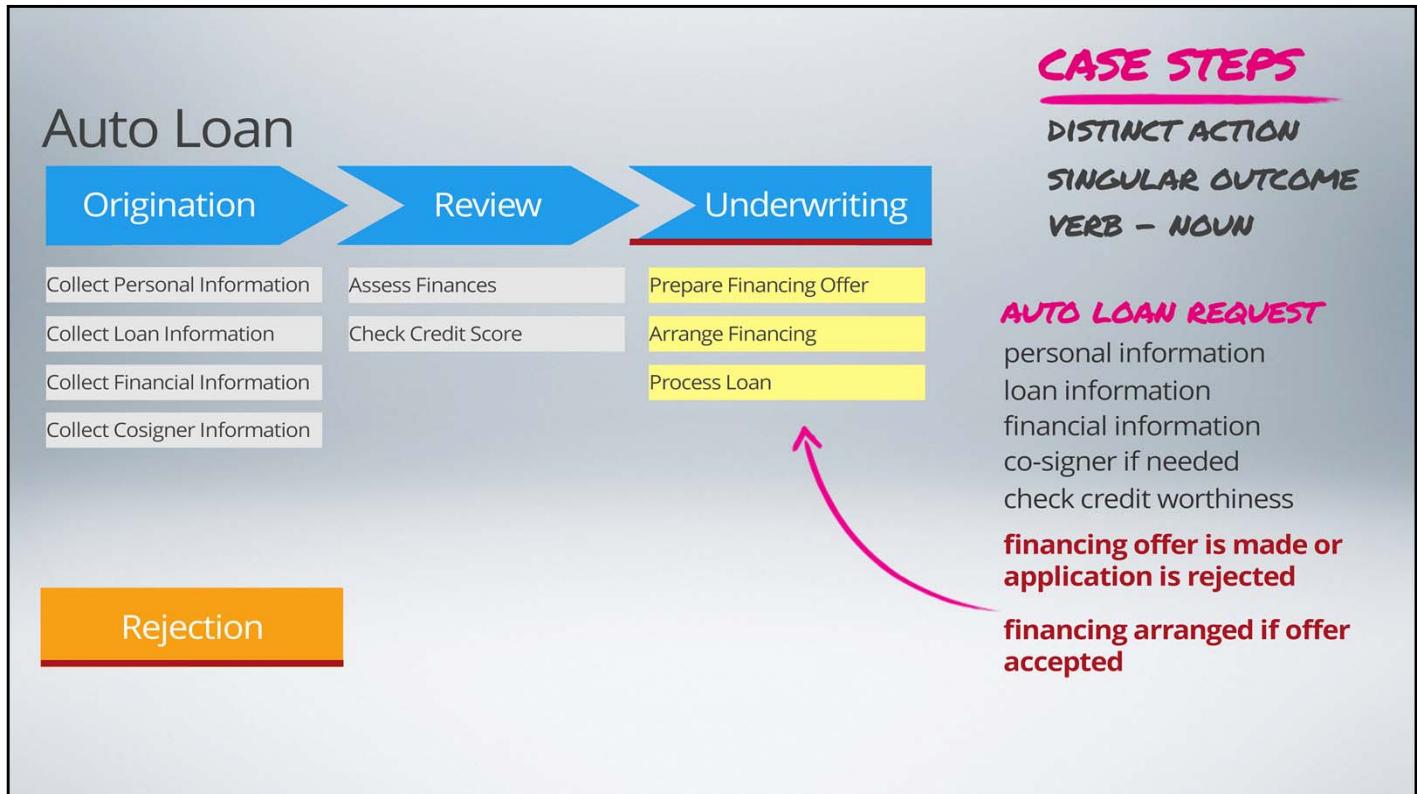


Once the application is completed, the borrower's financial information is evaluated

This consists of assessing the borrower's risk,  
and checking their credit score.

Now..., before we move on. Notice we placed these steps in the "Review" stage. Why?

Remember, a stage usually indicates a transfer of authority or significant change in status. In our user story it seems we meet both of these requirements so this "next stage" seems appropriate. The business tells us the completed loan application is sent to a loan officer who then conducts these checks. So, it seems the status of the loan application has changed from "new" to "pending review," or something like that. There is also a definite transfer of authority – from the loan applicant or agent to the loan officer.

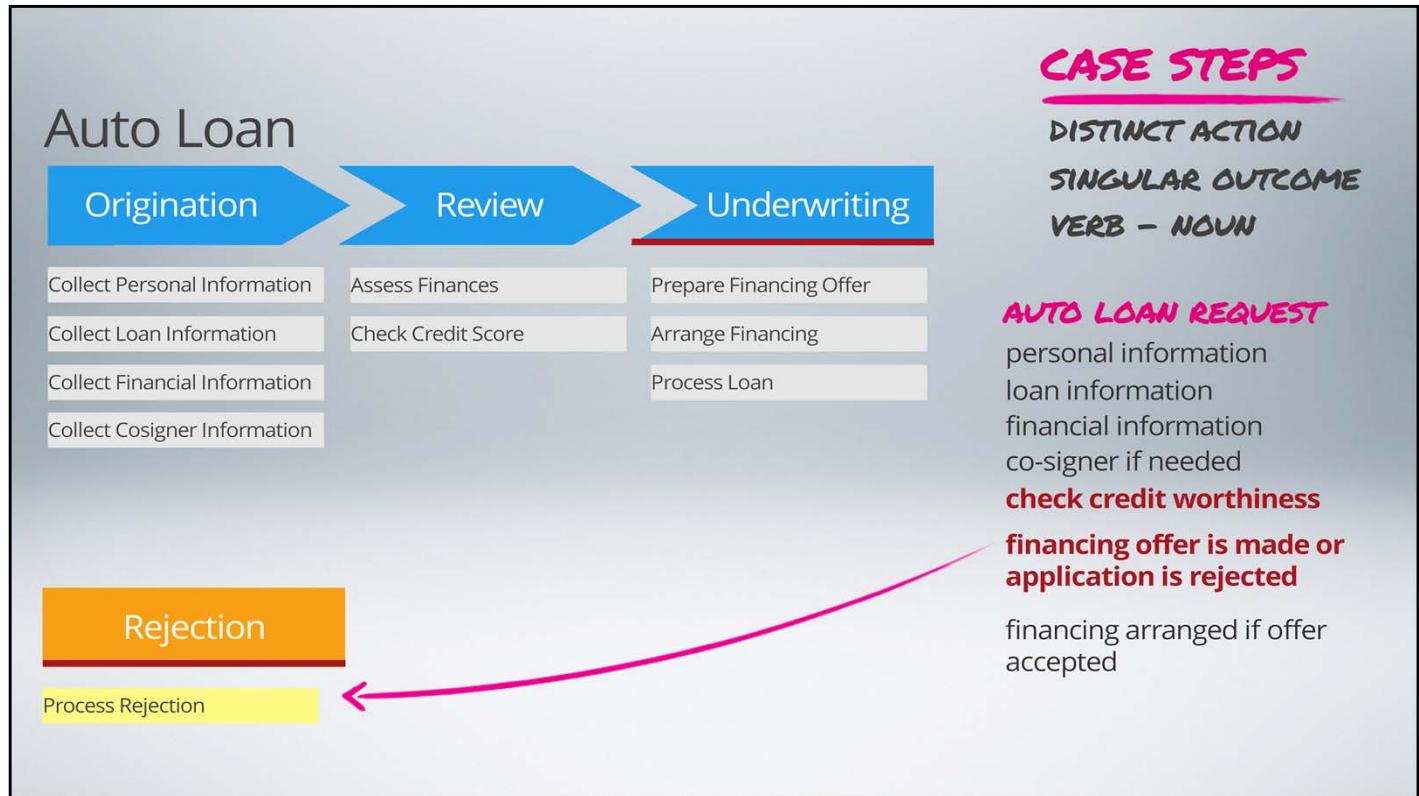


OK, let's continue identifying the steps. If the borrower passes the credit worthiness tests they have made it through the review cycle so we have another change in status – which means we can move to the next stage.

The user story tells us a financing offer is prepared and made.

Then, if the borrower accepts the offer, financing is arranged.

Let's add a step where we actually process the loan – signing paperwork, transferring money, things like that.



Finally, we need to recognize that there are a few places where the loan application can be rejected – either by the loan company or the borrower.

Let's account for that in the alternate “Rejection” stage.



There you have it - an initial draft of an auto loan request case map.

There are more ways in which we could map the auto loan case. The measure of success should be: did we adhere to our best practices? Let's check.

Are we using an iterative approach? This is our initial draft of the auto loan case, so it seems we are.

Did we focus on simply identifying the steps needed to transact an auto loan, while ignoring the details? For the most part, it seems we have.

Do we get a sense of order in how an auto loan request is handled? It seems we do.

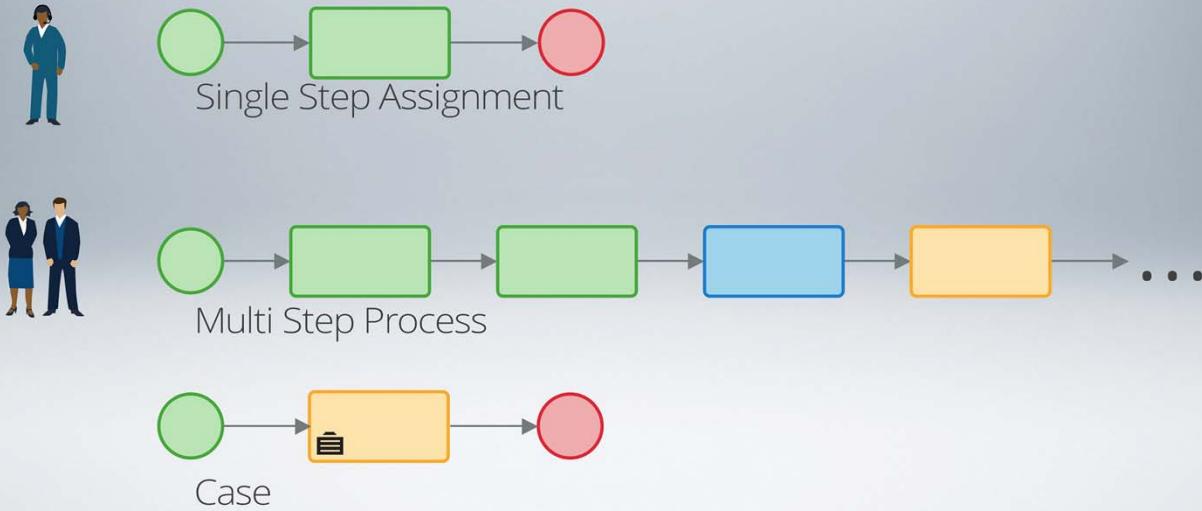
Is the case map universally understood? Is it meaningful to both the business users and technologists?

Did we limit the levels of decomposition? If we are following the "Rule of Seven," does any stage contain more than seven steps – plus or minus two? We have only four stages and no more than 4 steps in any given stage so it seems we can claim success here as well.

And, finally, can the auto loan case be explained – just using the steps we have defined in the case map – in 5 minutes or less? Read through the case map while timing yourself.

It seems we have hit all of these targets? Would you agree?

# Case Configuration Options



Let's begin this topic with confirming our understanding of the options we have for configuring a step in a case. Then, we will study a few use cases to help us understand when to use the available configuration options.

A step in a case can be configured in one of three ways:

a "Single Step Assignment" -

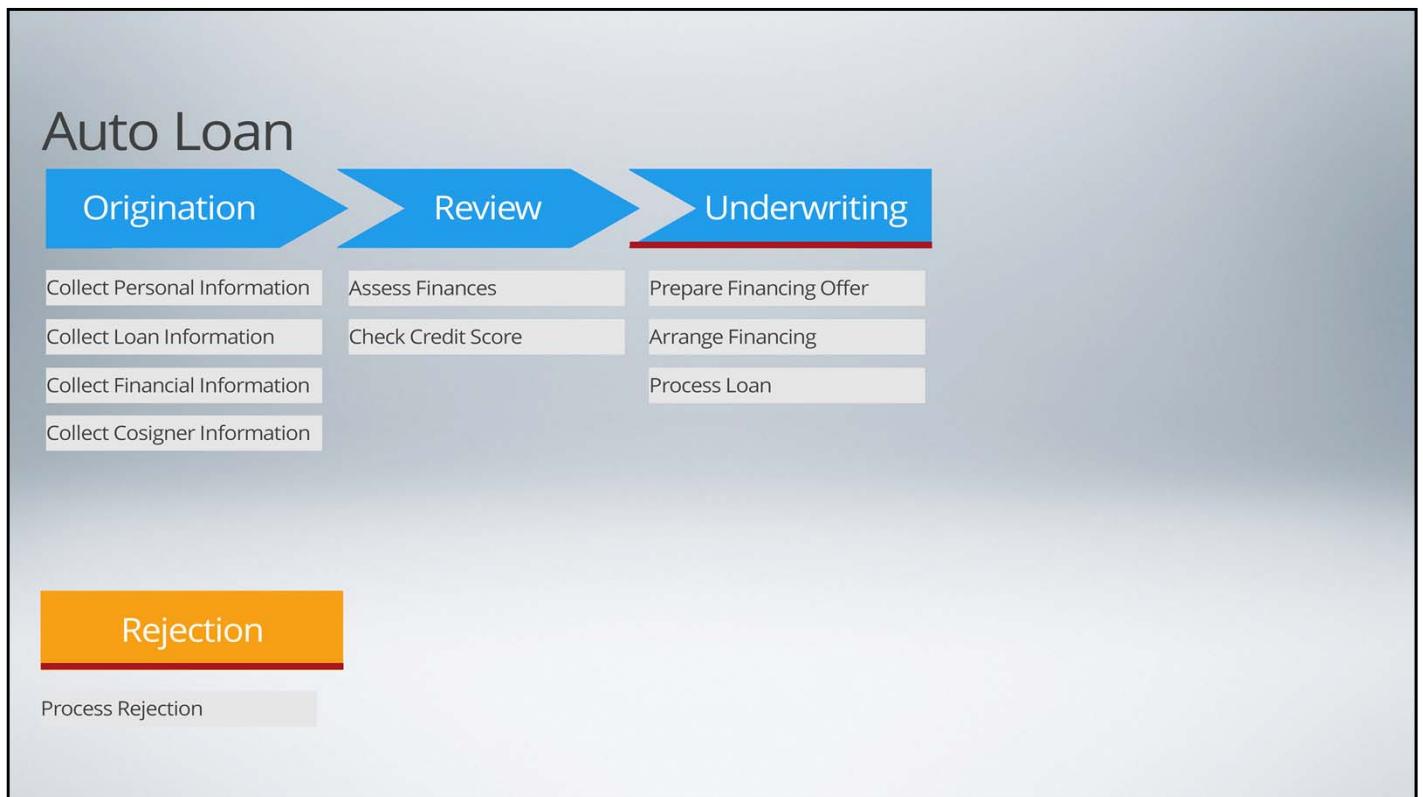
which is just that – a single action performed by a single human actor.

A "Single Step Assignment" can only be a UI screen and additional shapes cannot be added.

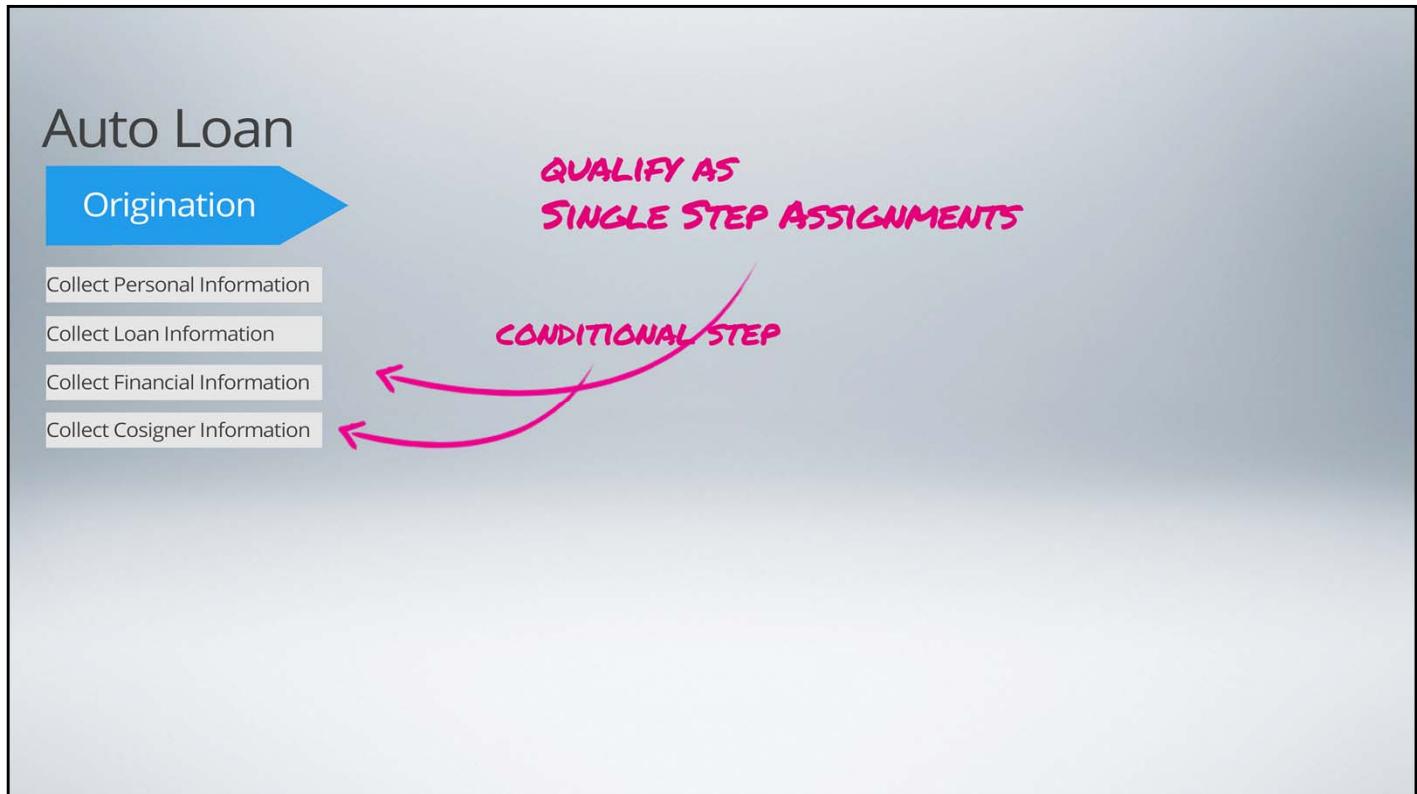
A "Multi Step Process" is any combination of actions – represented with different shapes depending on the action.

A "Multi Step Process" may involve multiple users, including PRPC itself or other systems.

The last option is a "Case." Cases are automated single steps, performed by PRPC, that create other cases. This starts the cycle of stages and steps all over again.



So..., how do we know when to use which option? Let's go back to our auto loan case example,

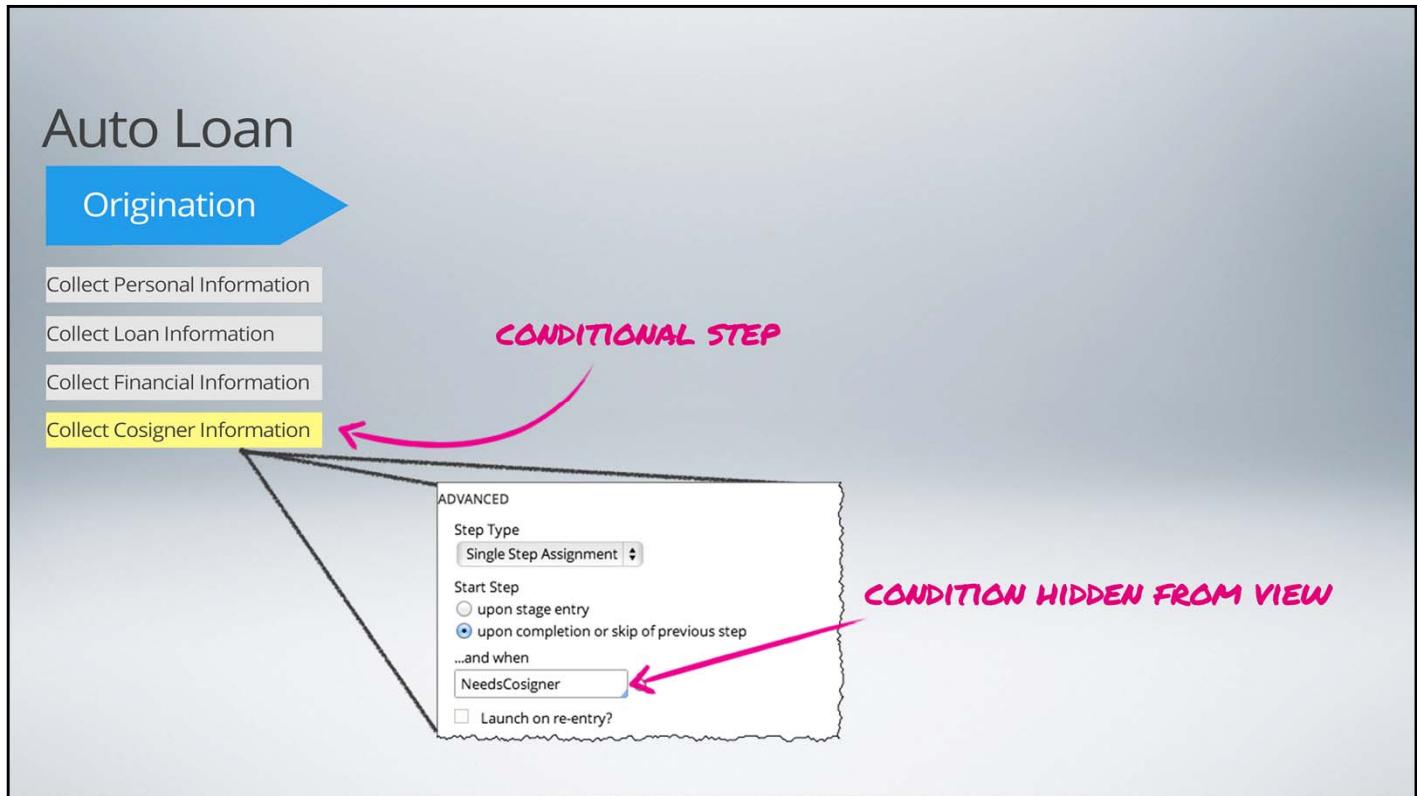


and, let's focus our attention on the steps in the "Origination" stage

Each of these steps qualify as a "Single Step Assignment" because a single actor has a single UI screen to interact with. And..., remember, this is the default configuration when you add a step to a stage.

However, the "Collect Cosigner Information" step is conditional – on some as yet unknown factors – but we do know it is conditional.

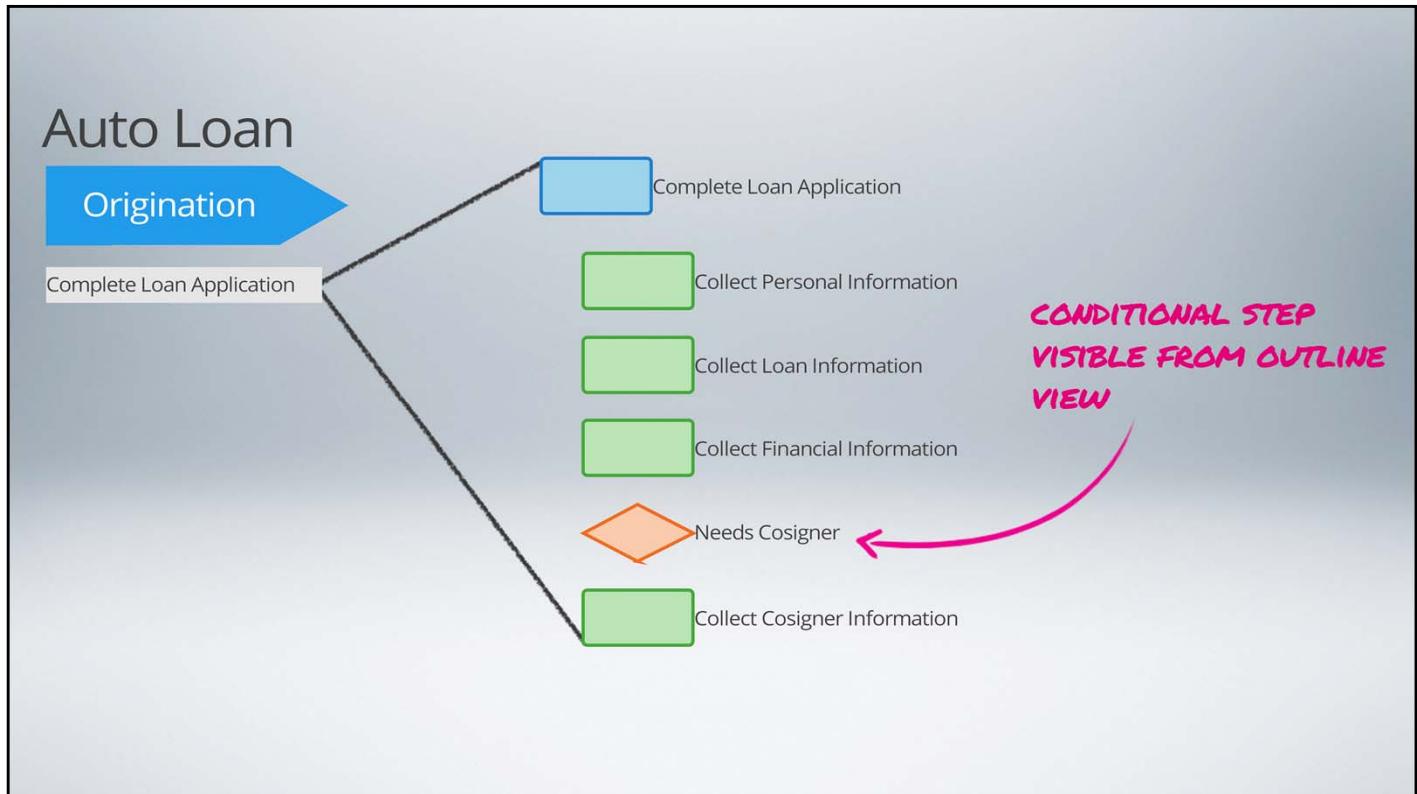
We could add an "...and when" condition to start the step when the conditions are met, However, the condition is buried in the step configuration – hiding it from view in the case map.



The only way we would know this step is conditional is by looking at the details of the step configuration.

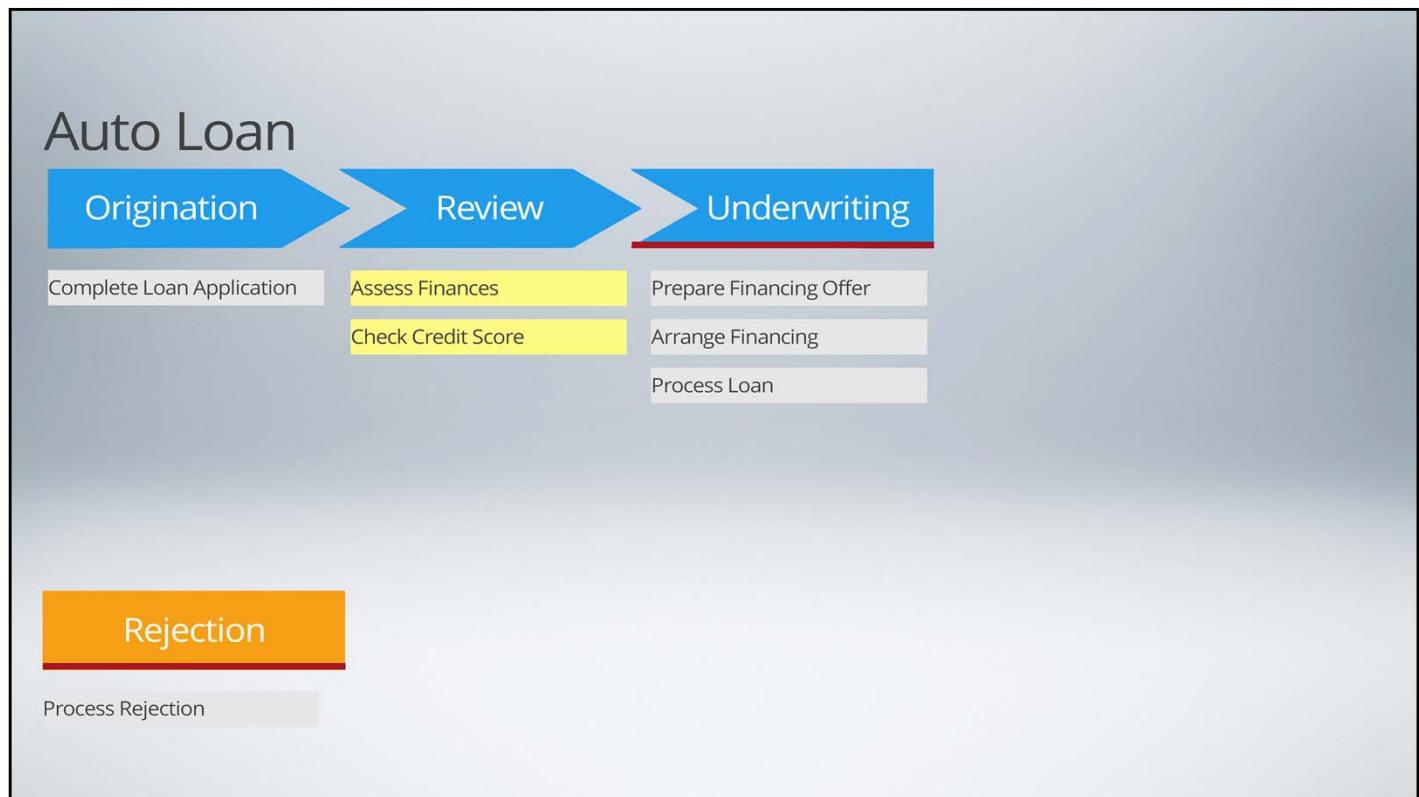
Now..., before we move on, let's be clear. This is an acceptable design – it will work. It may not be as clear as some believe it should be, but it will work.

This is where a “Multi Step Process” becomes very useful.

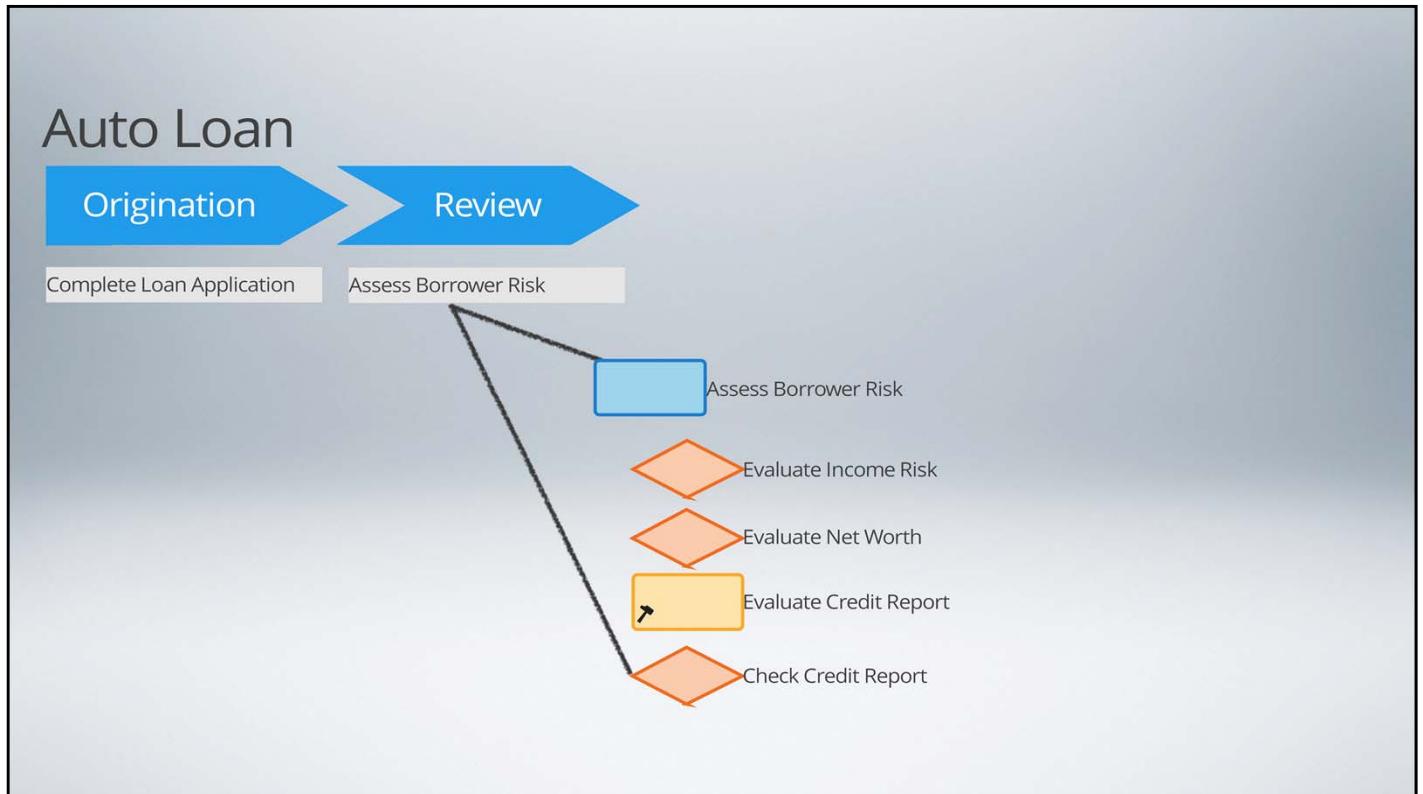


Remember, a multi step process is a collection of different shapes including not just assignment shapes but decision shapes.

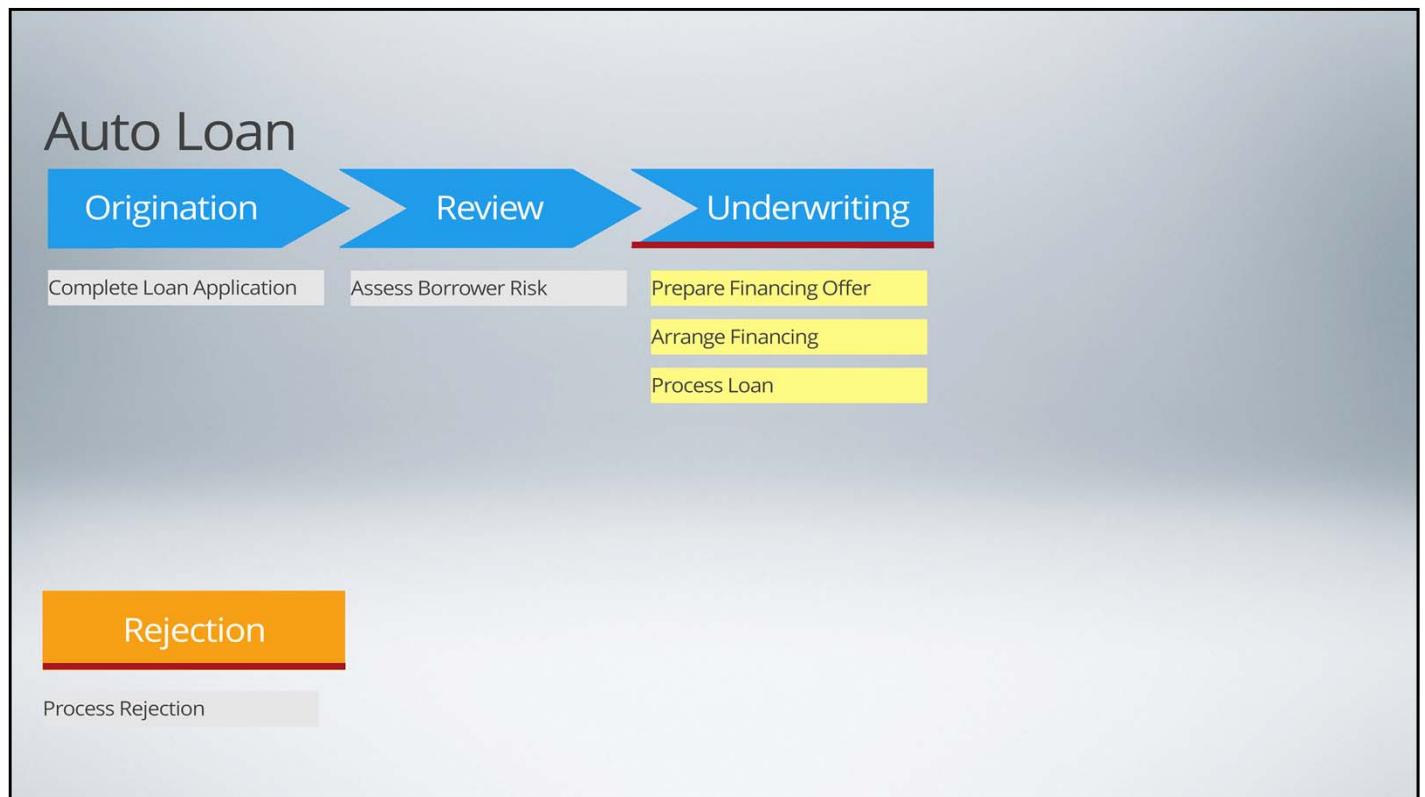
We can take advantage of the “Multi Step Process” configuration to add visibility to that conditional step.



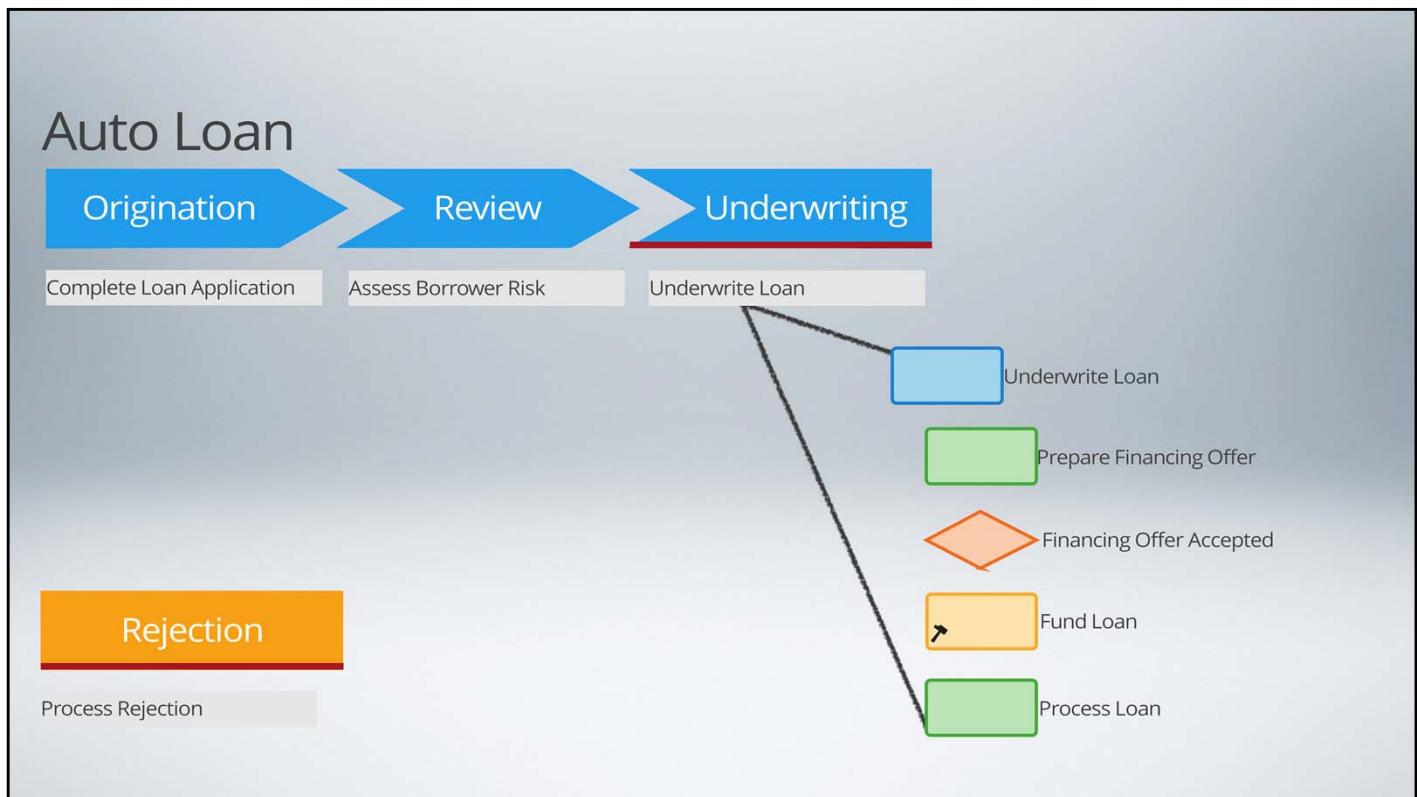
If we apply these same guidelines to what happens in the “Review” stage



we could use a multi step process to model how we evaluate a borrower's credit risk.

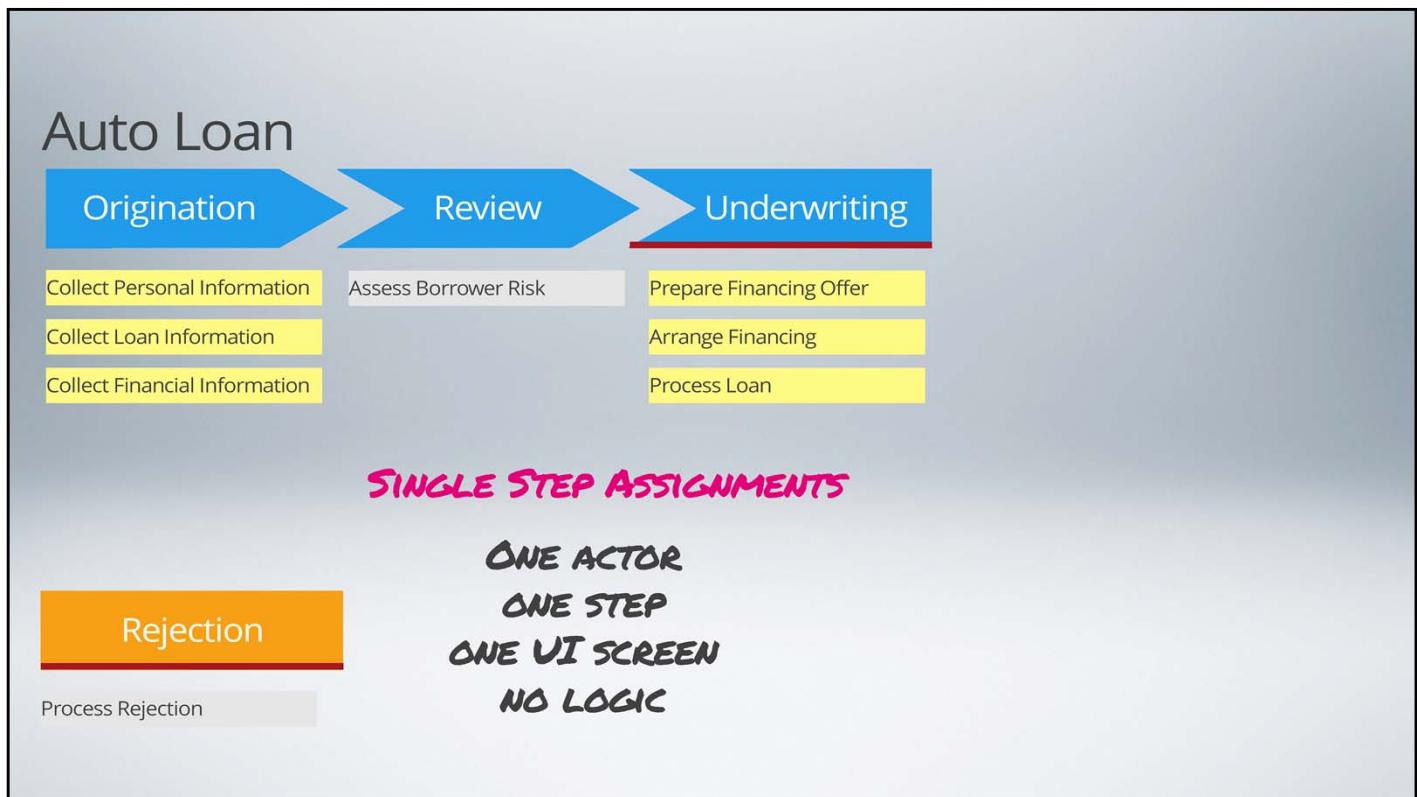


And...finally, in the “Underwriting” stage, we could collapse the three single steps

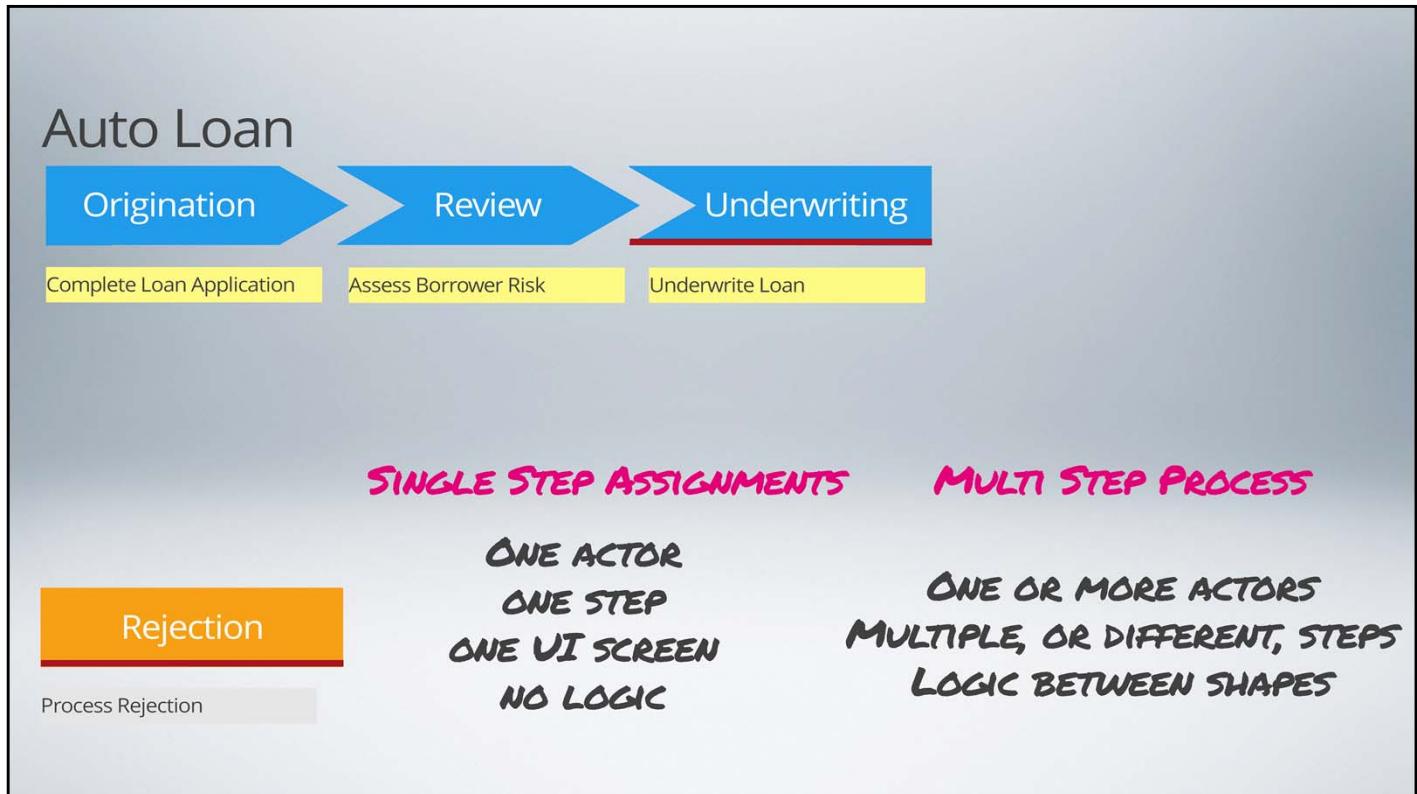


into a multi process step

and use the outline view to provide visibility into how a loan is processed.



Now, we do recognize this is a rather simplified view of an auto loan request case.



However, remember, the goal of this lesson is not to model an auto loan request case but rather knowing when to use a single step assignment

and when to use a multi step process.

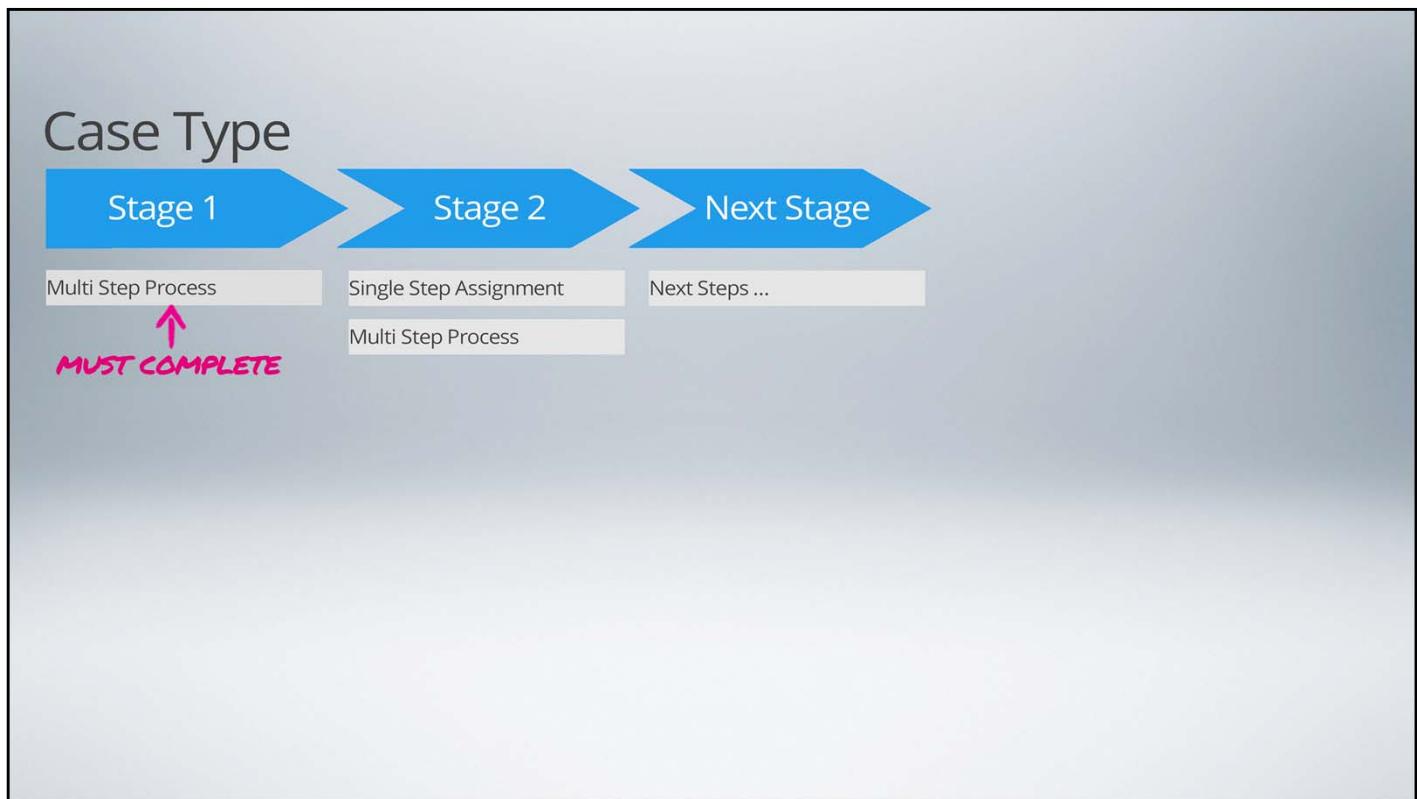
That leaves us with the third option we have for configuring steps – a “case.” To avoid talking in circles, let’s move this discussion to its own topic.

# Working with Subcases

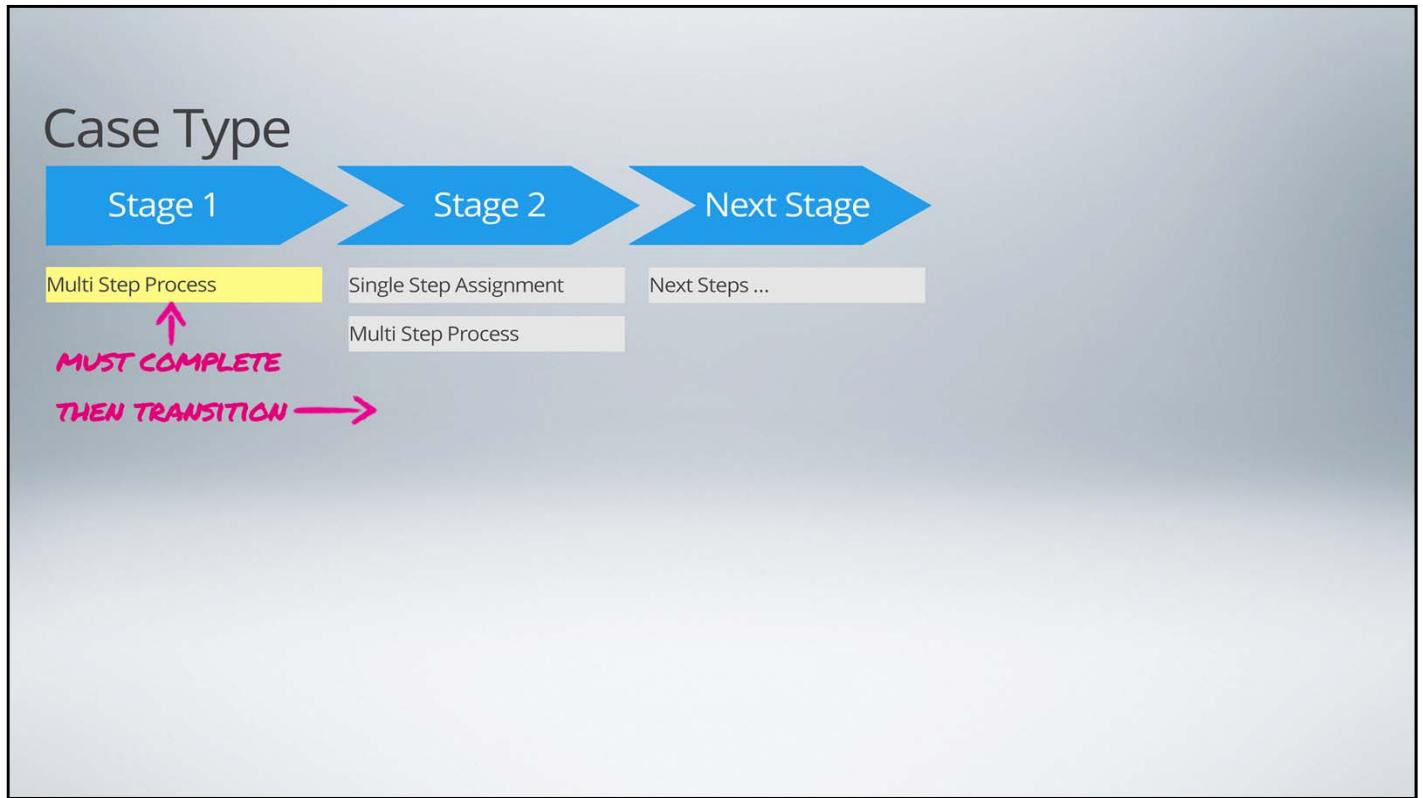
PEGA ACADEMY™

The last option we can use to configure a step in a case is a “Case” itself – and, in particular, as a subcase.

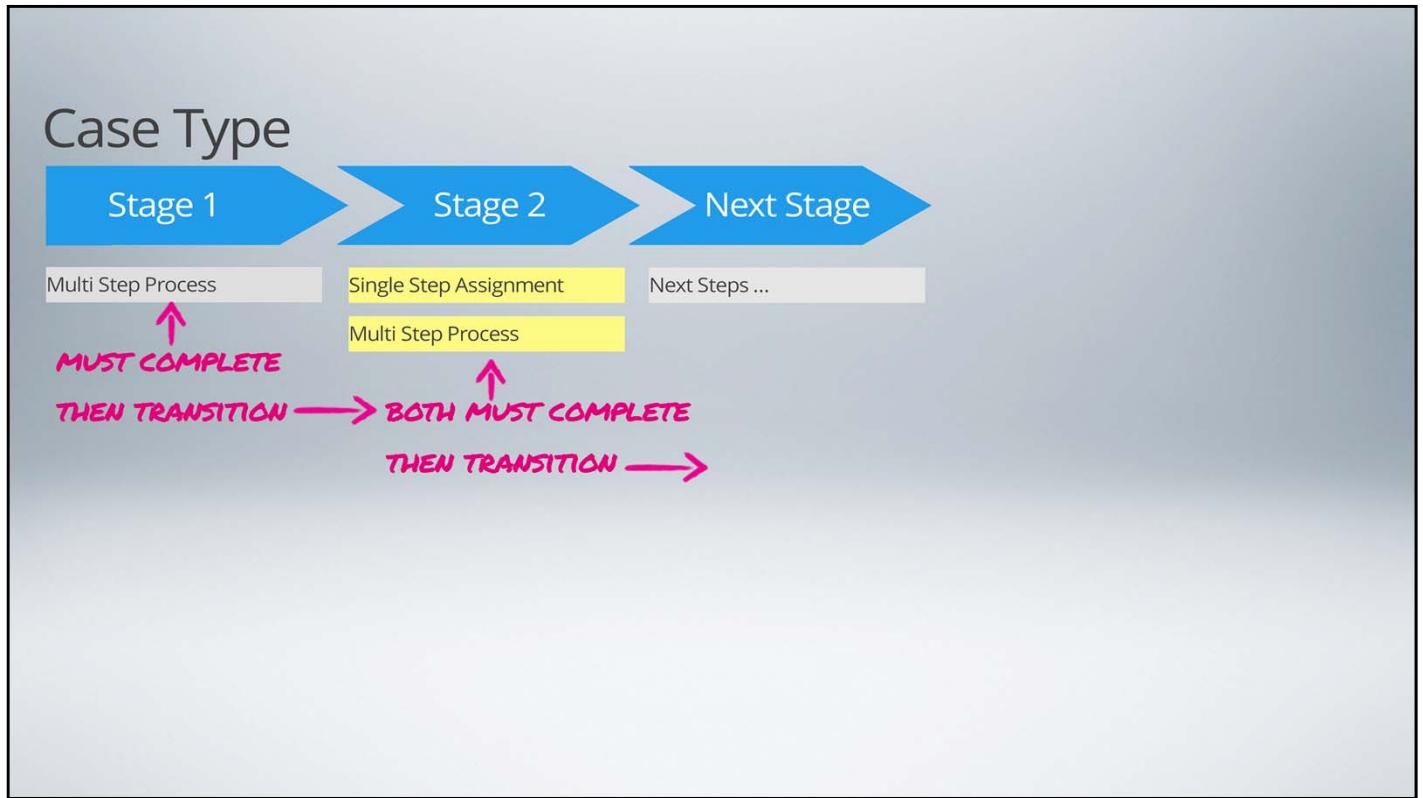
Let’s begin this topic with confirming our understanding of how a case type executes at runtime.



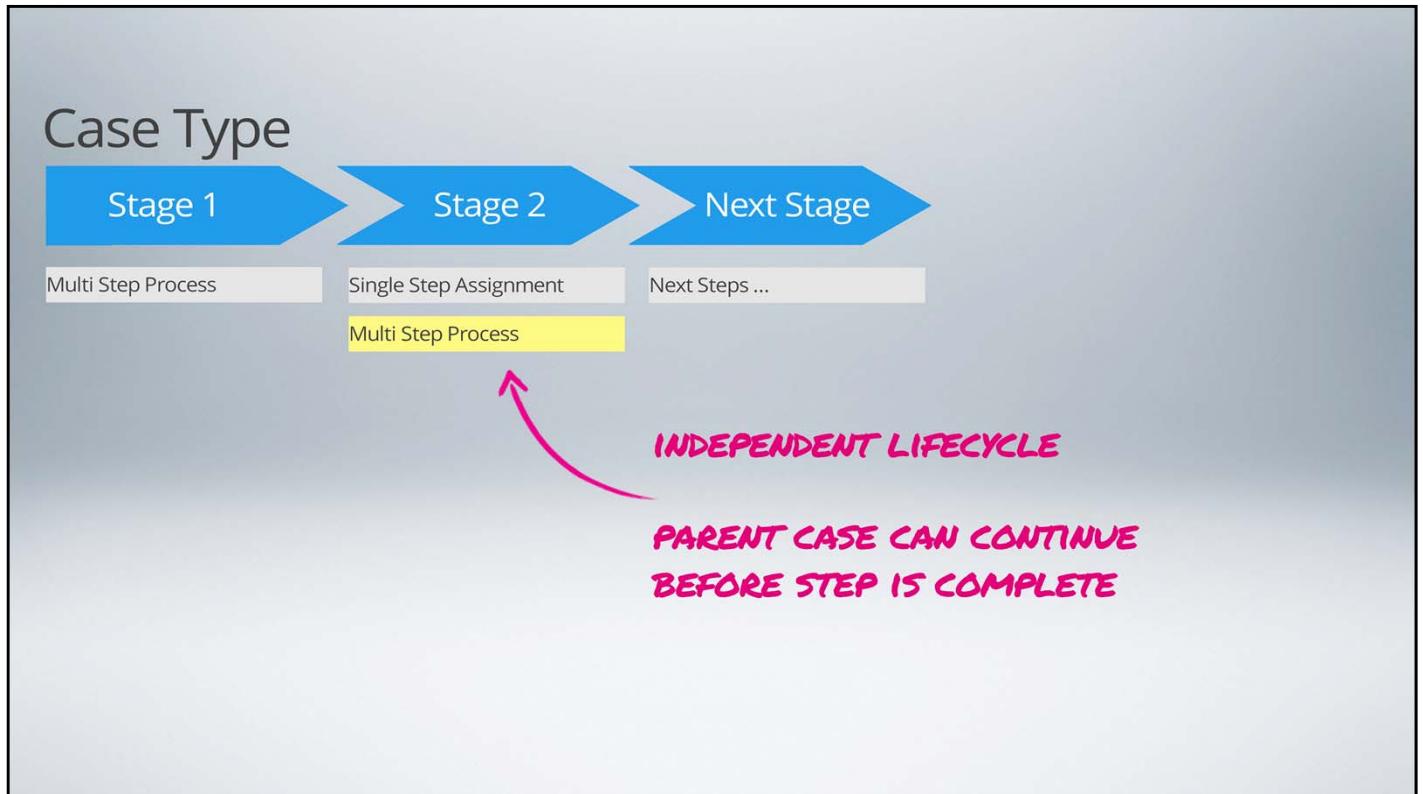
When using “Single Step Assignments” or “Multi Step Processes,” each of the steps in any given stage must complete before the case can advance to the next stage. There are options for skipping steps, but all steps in a given stage must be resolved somehow before the case advances to the next stage.



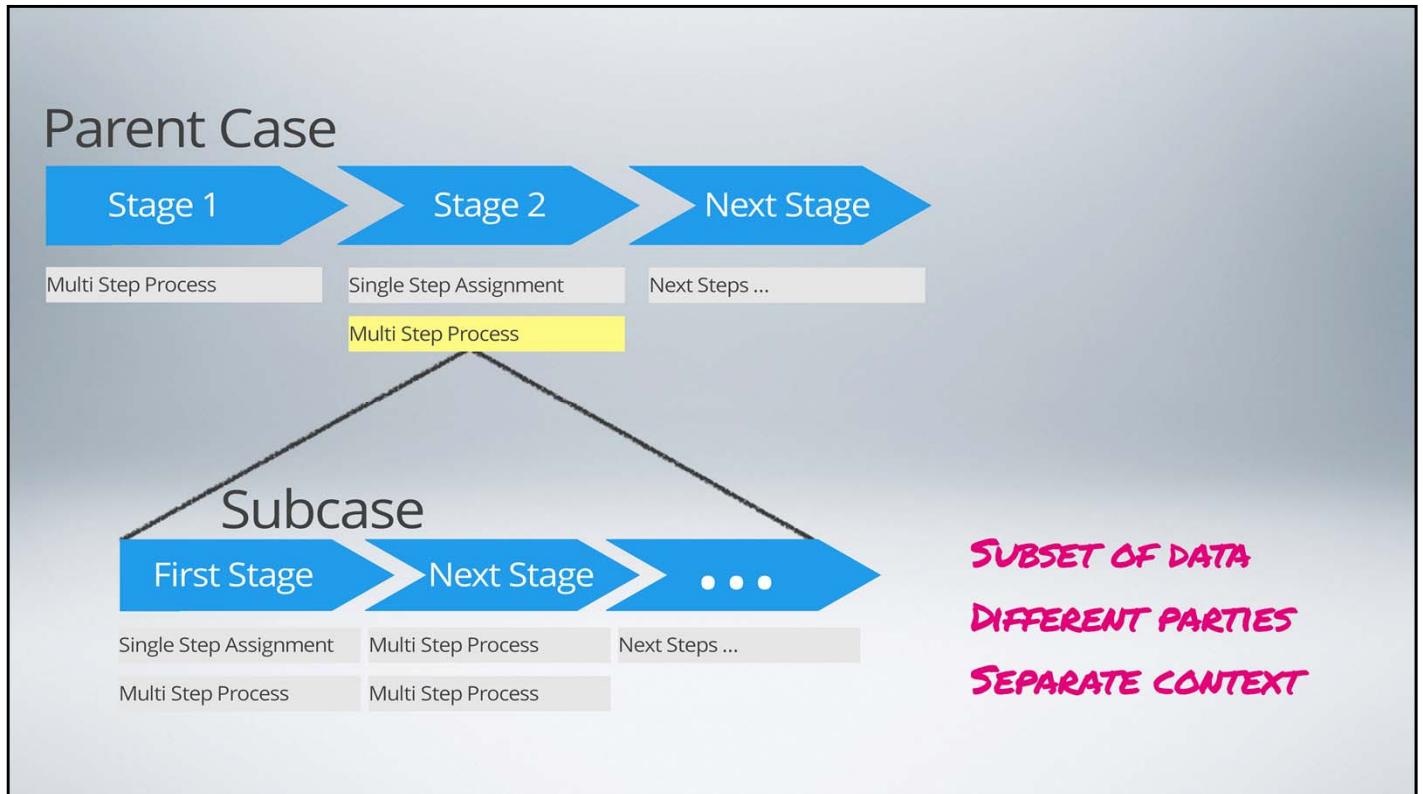
However, a very common case design pattern requires some “things to do” that are independent of the main, or parent, case.



These “things to do” have their own life cycle

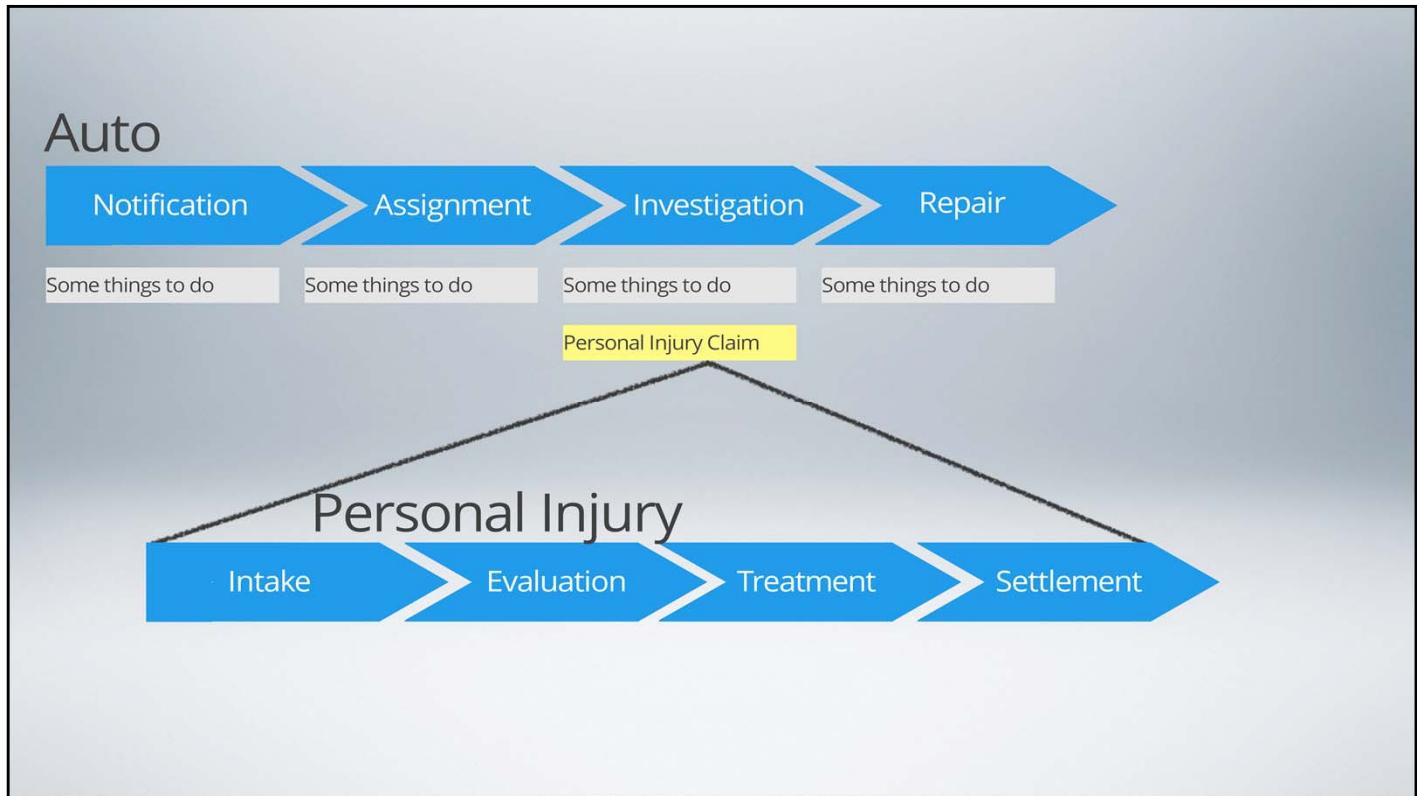


- but are still very much related to the main, or parent, case. And..., the work in the parent case can continue without waiting for those things to complete. This is where a subcase becomes useful.



A subcase allows us to subdivide a main case into individual transactions that can be handled separately.

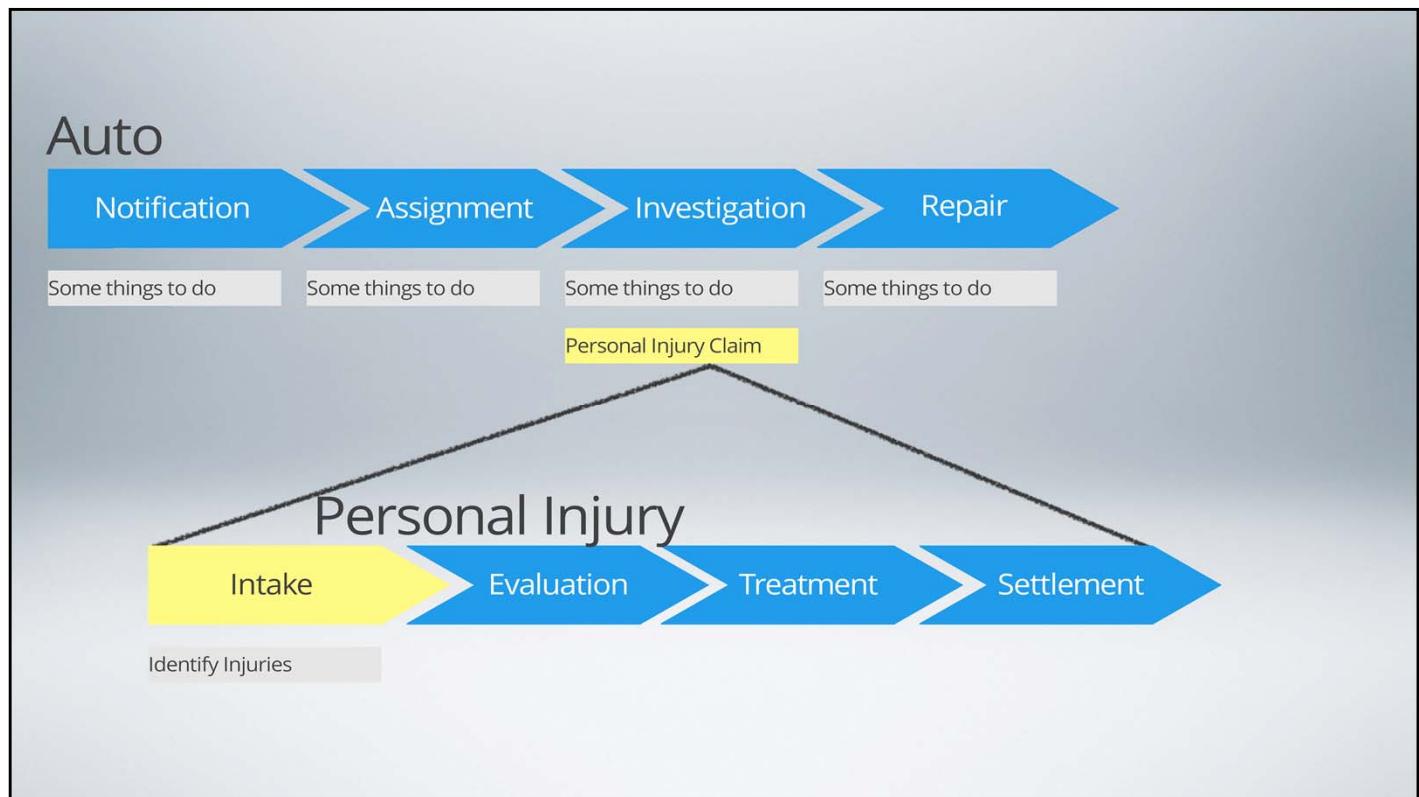
Subcases typically focus on a subset of the data relative to the parent case, usually involve different parties than the parent case, and may be executed outside the context of the parent case.



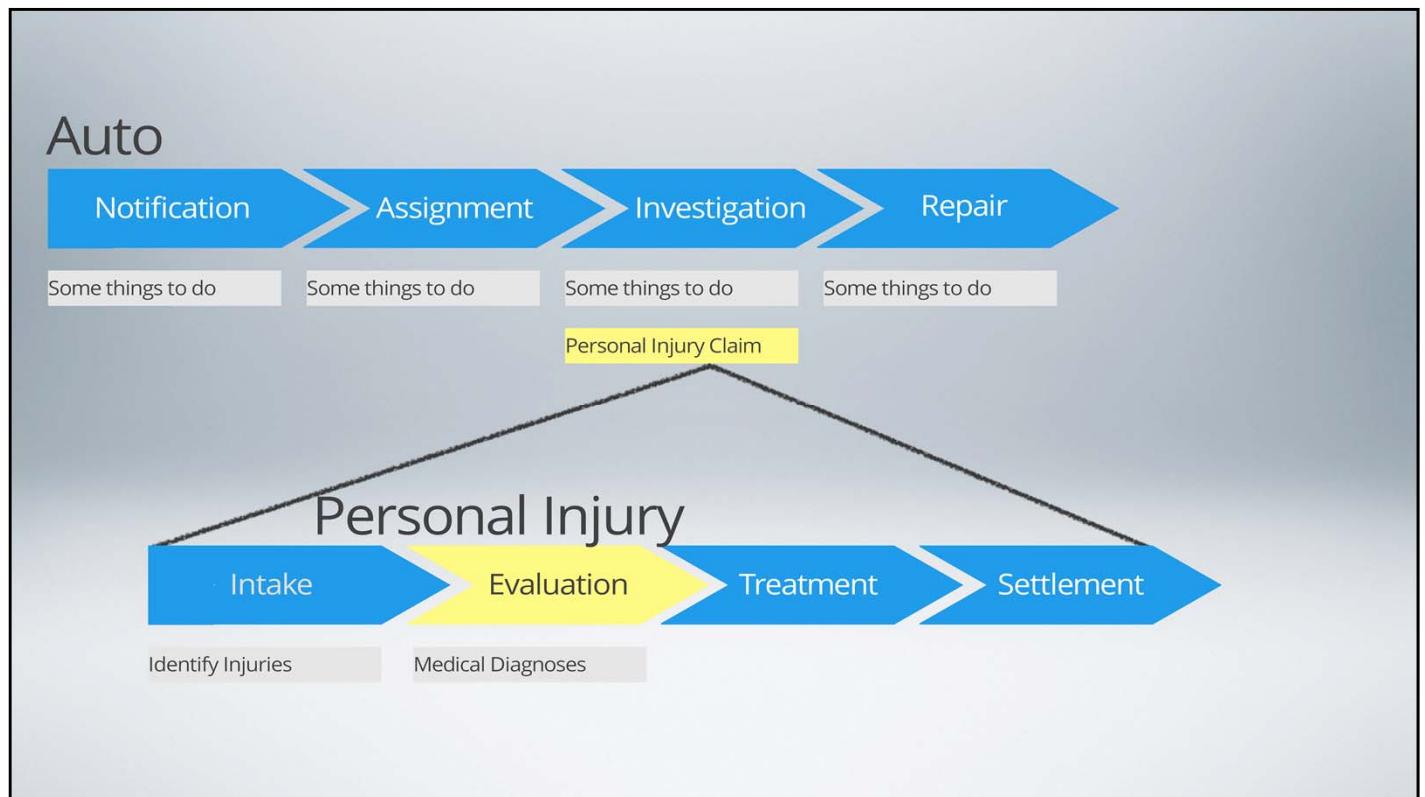
Let's look at an example – that of an auto accident claim; it seems we wrecked that fancy new car we just financed.

And...again – let's keep the actual claims process simple and focused. Our use case will be that when an auto accident claim is filed, we do some things. Among those “things to do” is identifying if anyone was injured and, if so, manage the personal injury claim – or claims - if necessary.

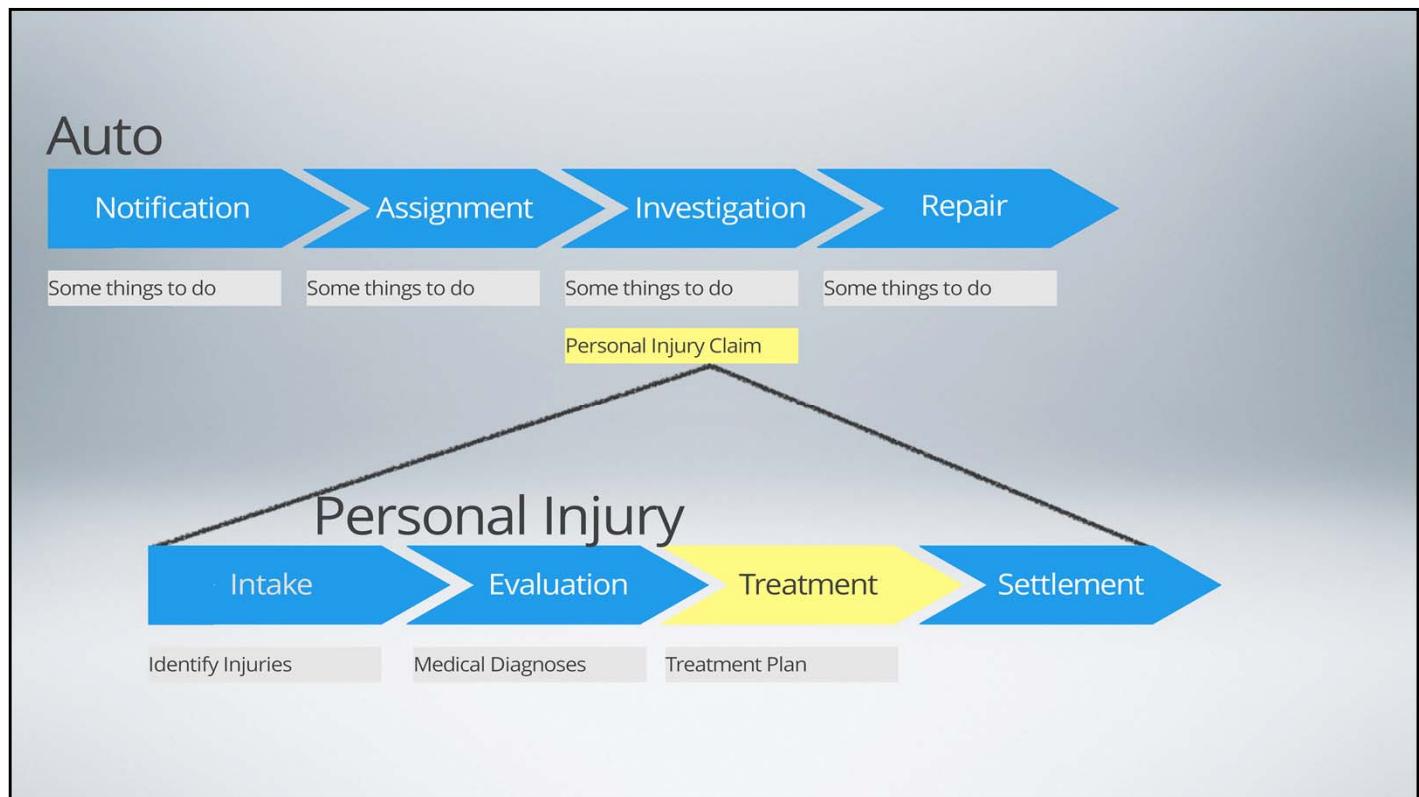
The lifecycle of the Personal Injury Claim is undoubtedly different than the auto claim’s lifecycle. And we are definitely dealing with a subset of the data.



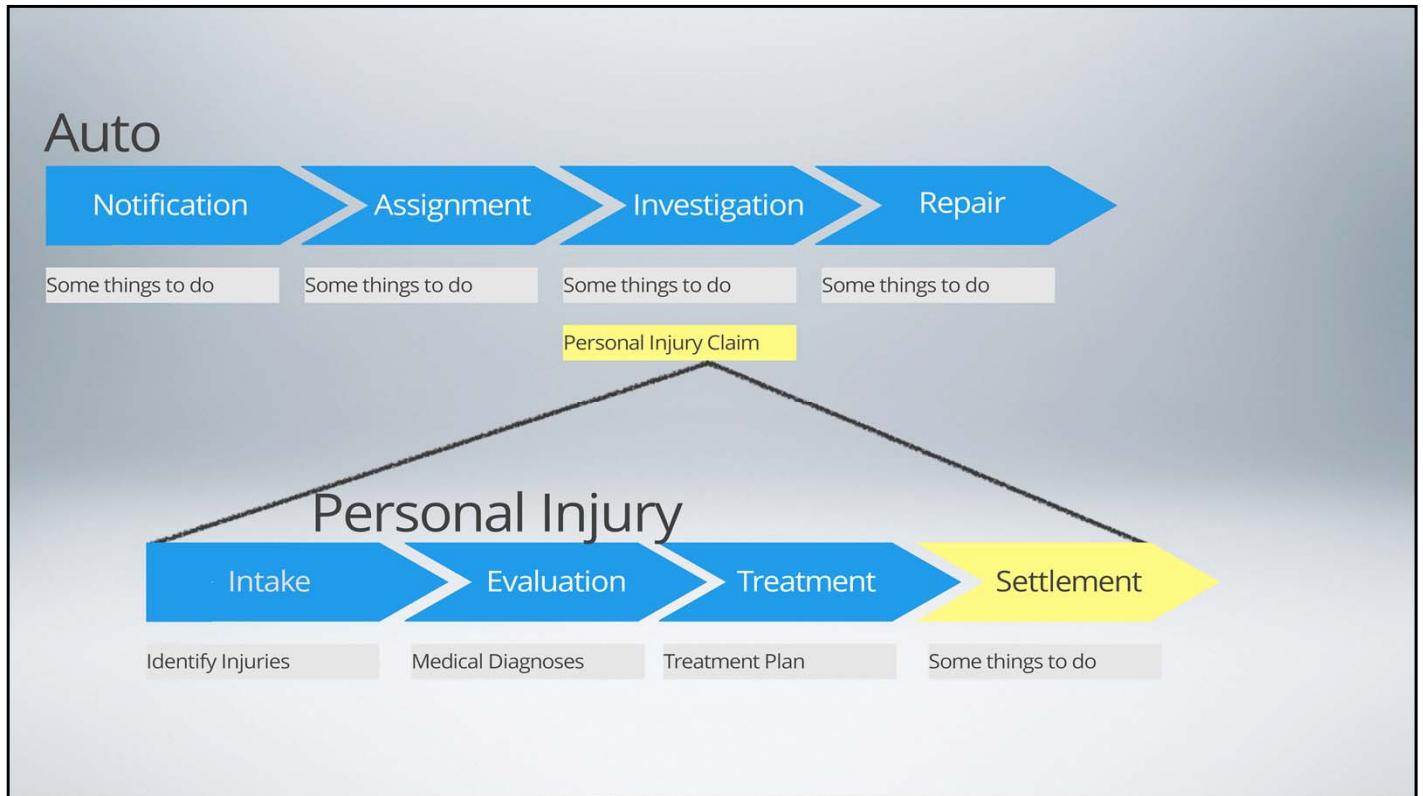
Instead of damage to the vehicle and repair estimates, the personal injury case requires information regarding the type of injuries,



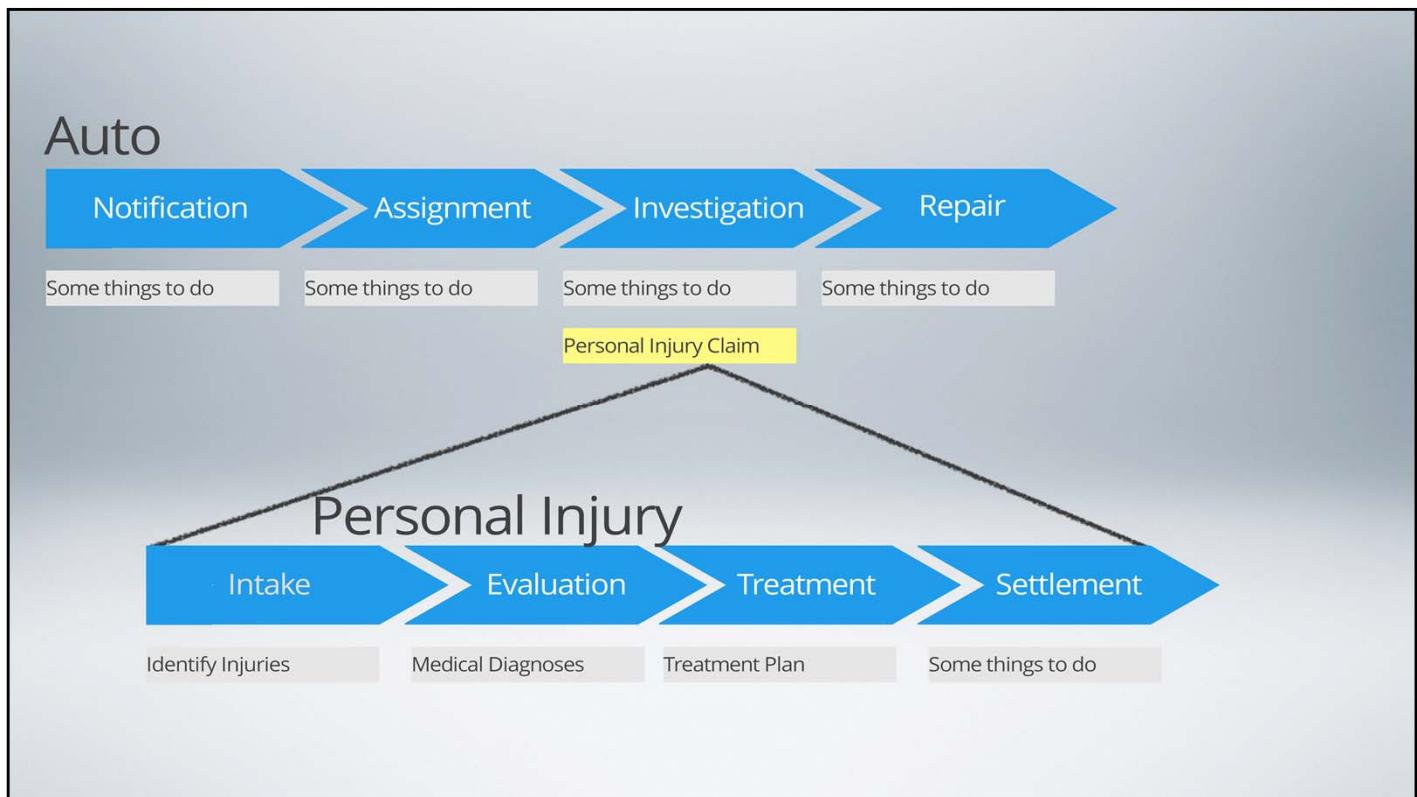
a medical report from the emergency responders, the hospital or the doctors,



some data regarding how the injured parties are to be treated – and the cost. While we are here, it may be worth pointing out that the “Treatment Plan” may very well be yet another subcase.

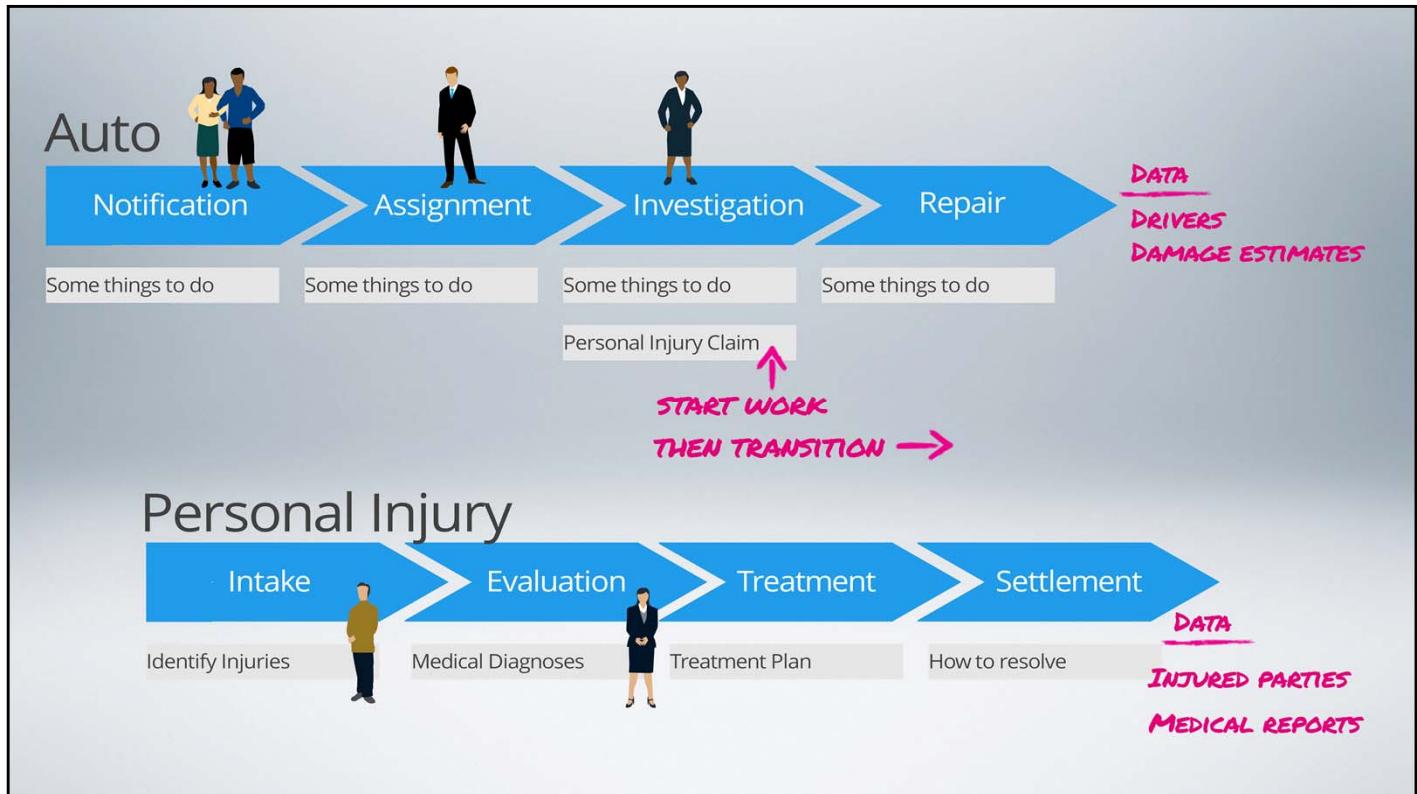


Finally, there are undoubtedly some things to do to settle the personal injury claim.



This example is purposefully simple – and definitely incomplete – but we just want to focus on how to know a subcase when we see one.

So, let's review the guidelines for using subcases and compare the auto claim with the personal injury claim case types and see if our sample case design meets the criteria.



The lifecycles are indeed different.

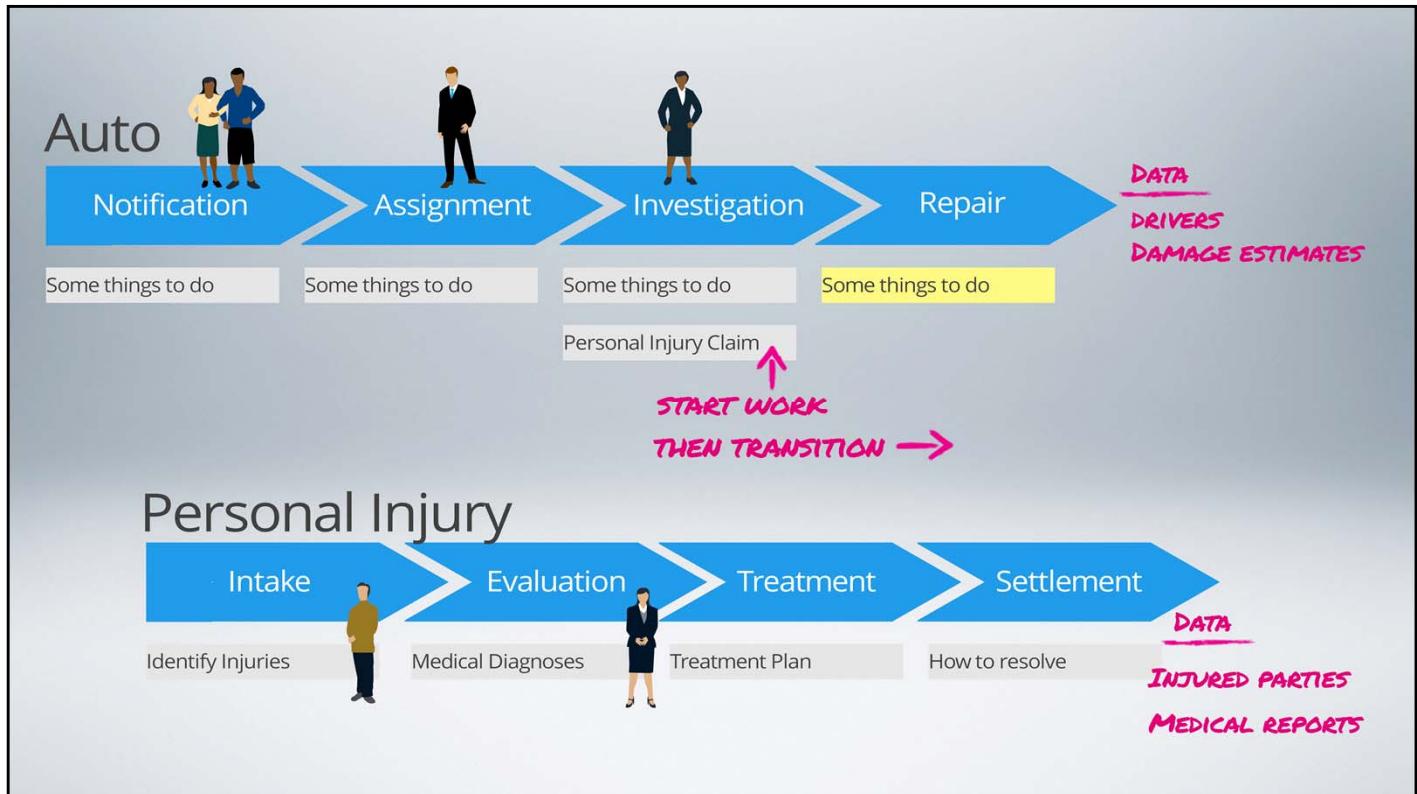
Remember, subcases typically focus on a subset of data relative to the parent case. In the auto claim case we have information about the insured and the other parties insurance coverage, vehicle damage and damage estimates, things like that.

In the “Personal Injury Claim” however, we need to know who is injured and the nature of the injuries,

medical information such as medical diagnoses and any treatment plans – and the costs associated with the medical attention.

The actors are different as well. For the auto claim there is the insured, the insurance agent, and probably the adjustor.

For the personal injury claim, there are the people injured and the case worker assigned to the personal injury case.



Finally, the personal injury claim does, indeed, execute outside the context of the auto claim case. If personal injuries are claimed,

the auto accident claim itself can continue being processed.

For example, damage estimates and repairs can take place while the personal injury claim is being handled separately.

## Demo



### Creating and Using Subcases

Let's see how we would go about adding a subcase to a parent case.

When we discover the need to add a step that will be configured as a subcase, we need to create the subcase before we can add it as a step in a parent case if it does not already exist. To create the subcase, select "Add a case type" from the Case Explorer drop down menu. In the "Create Case Type" dialog, we'll provide a meaningful name for the case type – in our example, "Personal Injury" and..., because this case type will be a subcase of the Auto claim, we must specify the parent – in our example, we select Auto.

Now, we click OK to create the case type. The case type is created and opened in the Case Designer. The case type is created and available for use – there is no need to flesh out the case type at this point; we can use it as is. However, we can make some modifications now – such as naming a few of the stages. Remember to save any changes.

Now, let's go back to the Auto case type and make use of the newly created case type. In our example, we want to add the Personal Injury as a subcase in the "Investigation" stage, so we "Add a step" to that stage.

We'll name the step "Personal Injury," then click "Configure step behaviors" from the step options menu. We want to configure this step as a "case" so, in the "Step Configuration" dialog, we'll select "Case" as the step type,

then click OK. There are a number of options for configuring a step as a case, but let's focus on creating a subcase for now. So, we'll select that option. Notice the options change yet again. From the "Case type" drop down menu, we'll select the "Personal Injury" case type. Notice a starting process is automatically created for us. Finally, we click OK to commit the changes in the Step Configuration dialog.

Let's save our changes to the Auto case type.

## Pega 7.1.6 Update Notes

### Updates for Pega 7.1.6:

- Dialog for adding a new case has been updated for ease-of-use

The screenshot shows the 'Add' dialog box for creating a new case type. It includes fields for Name (Purchase Order), Description (Purchase Order), and a checked checkbox for 'add this as a subcase'. A dropdown menu for 'Select the parent case' is shown. Below this is a section titled 'ADVANCED SETTINGS' containing fields for Derives From (Directed) set to 'Work-Cover-', Derives From (Pattern) set to 'MainCo-EmpReimb-Work', RuleSet set to 'EmpReimb', Version set to '01-01-01', and Remote Case type with a checked checkbox. At the bottom are 'OK' and 'Cancel' buttons.

- Two new step types have been added
  - Single-step assignment — Flow contains one Assignment shape
  - Multi-step process — Flow contains two or more shapes
  - Case — Flow contains a Create a Case Smart Shape
  - Approval — Flow contains the Approval Flow subprocess (new)
  - Attachment — Flow contains the Attach File assignment (new)

## Review



Case Type



Stage

Step

Step

We've covered quite a bit of ground so far. Let's end this lesson with a short review of how case types, stages and steps work together and a consolidated view of the best practices for effective case decomposition.

**Case Type** = artifacts needed to implement a business transaction



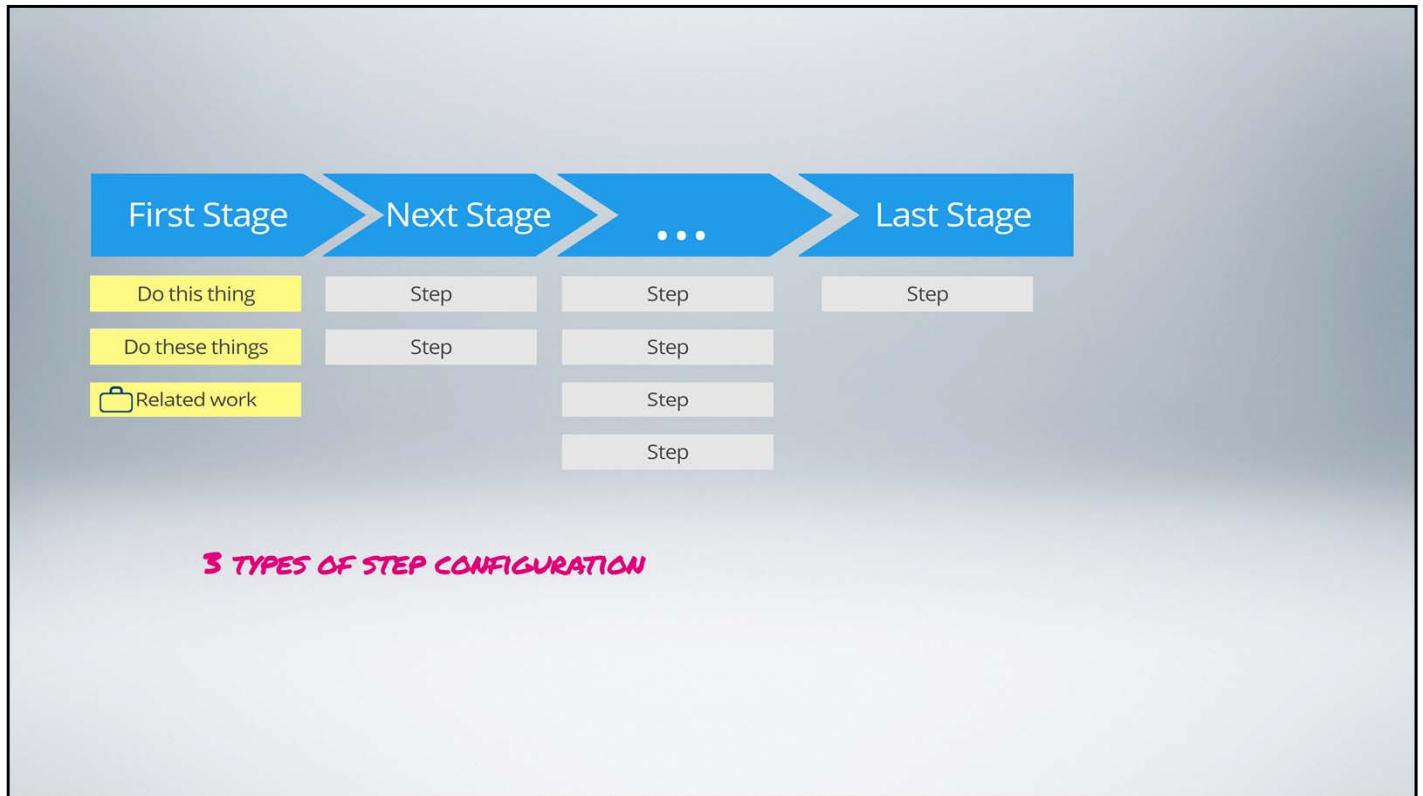
Let's start with "case type."

A case type is a collection of PRPC-related artifacts such as data elements, UI screens, processes and sub cases, decisions and integrations used to implement the tasks for a specific case.

A case type is organized into primary and alternate stages. Use primary stages to represent the "normal course of events,"

and use alternate stages to represent exceptions to the normal course of events.

Use stages to represent a significant change in status or a transfer of authority.



Each stage contains one or more steps. A step is a distinct action taken to help resolve a case and should have a goal that can be expressed as a singular outcome.

Steps can be a “Single Step Assignment” – one actor does one thing, a “Multi Step Assignment” - one or more actors have many different things to do, or a “Case” – which is used to manage an independent, but related, business transaction.



Consider limiting the decomposition of any given case type - we suggest seven, plus or minus two for any given dimension.



And... finally, "watch your language."

Use a noun or a noun-phrase to name a stage and verb plus noun to name steps.

**USE NAMES MOST MEANINGFUL AND RELEVANT TO BUSINESS USERS**

Here, in particular, use names that provide the most meaning, relevance and understanding to the business users.

## Exercise: Decompose a Case into Steps

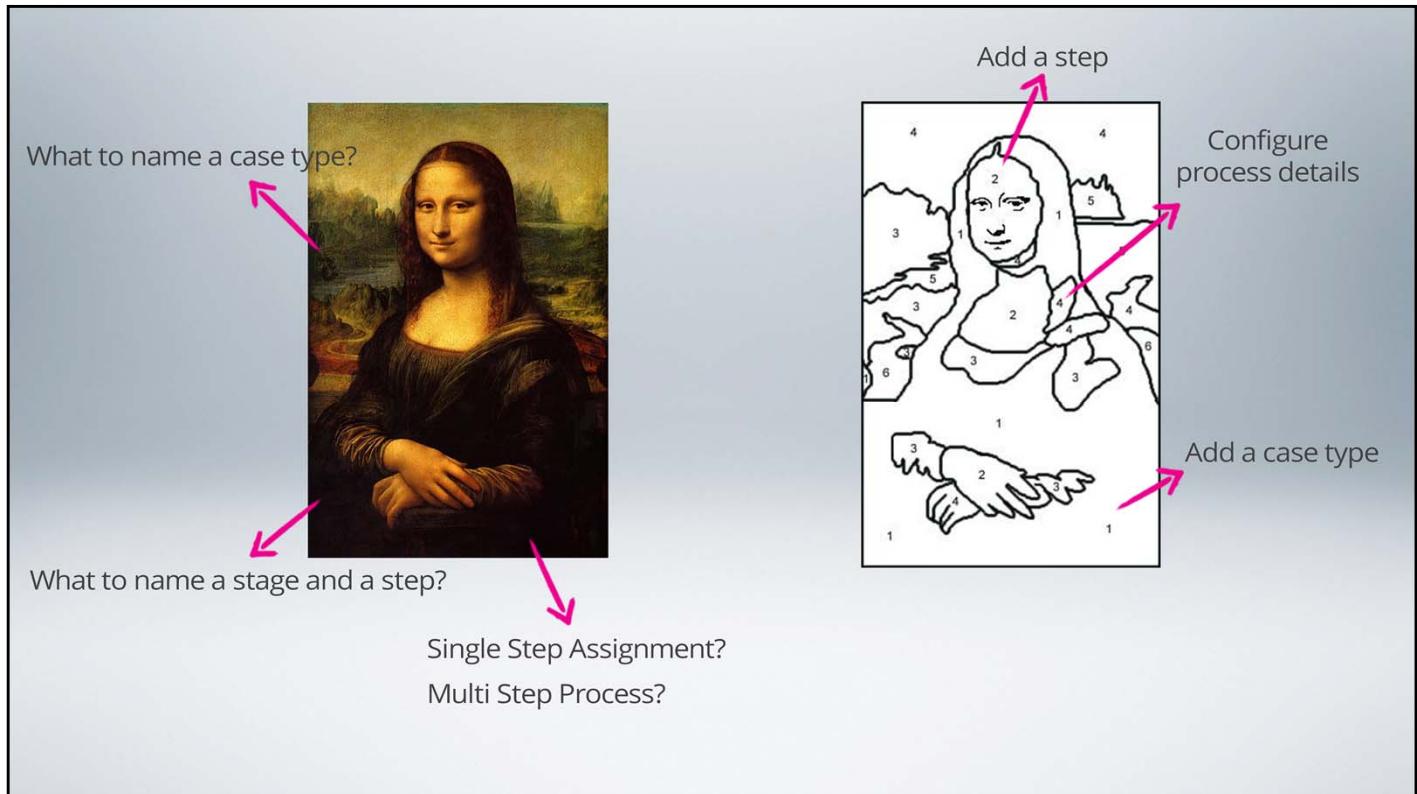


# **Best Practices for Effective Process Decomposition**

**In this lesson, we learn how to build shared, executable process models.**

**At the end of this lesson, you should be able to:**

- State at least three best practices for effective process decomposition



In this lesson, we want to discuss best practices for how to further refine a Multi Step Process.

Case and process decomposition is more an “art” than it is a “science.”

The “science” of process decomposition

is knowing how to create a case type;

and where to click to add a stage or a step,

and how to change the configuration of the step after it is added. This science is pretty well documented and relatively easy to follow.

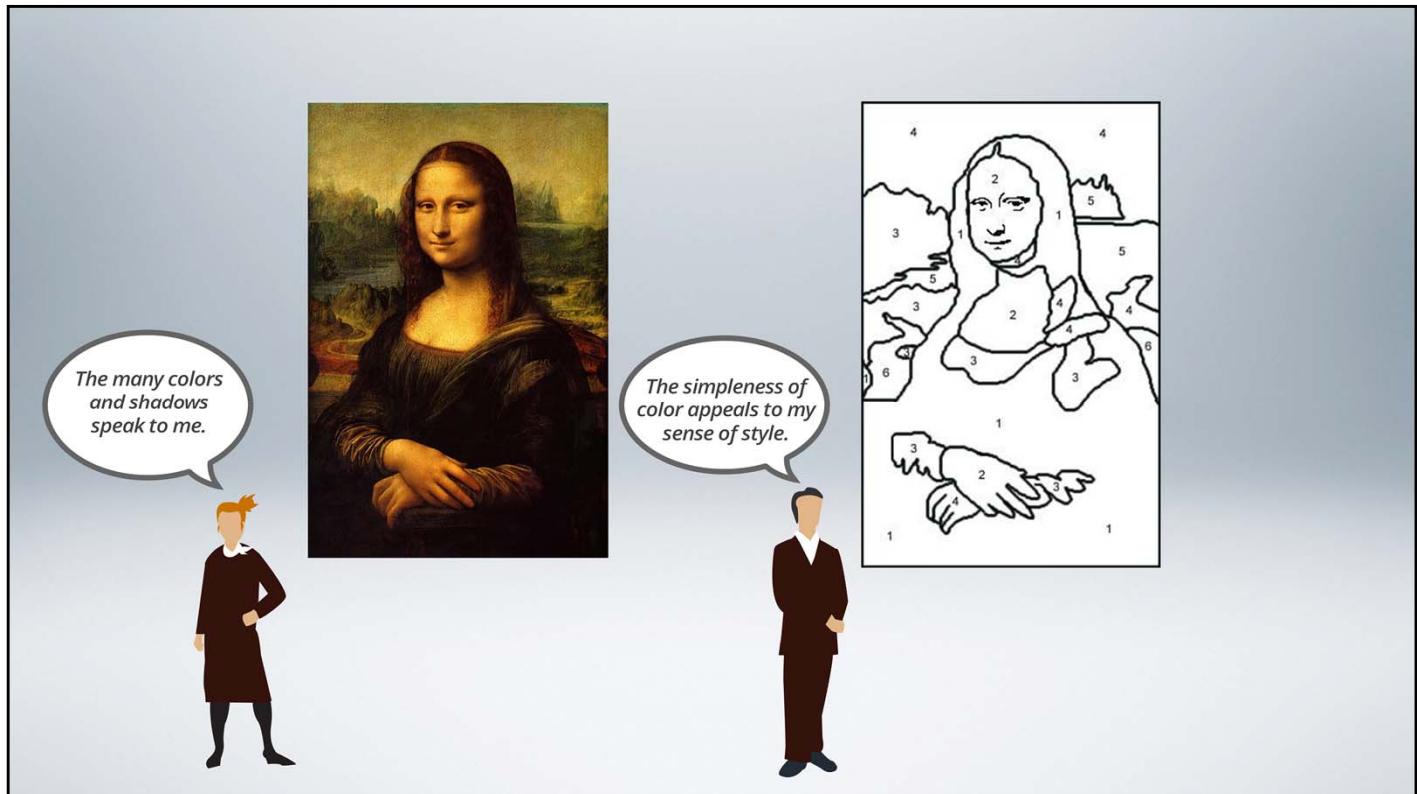
The biggest challenge will probably be the “art” of process decomposition.

What do we name the case types;

the stages and the steps.

And then, there is the question “How do we know what to add to a stage or a step?”

This is the art of case and process decomposition.



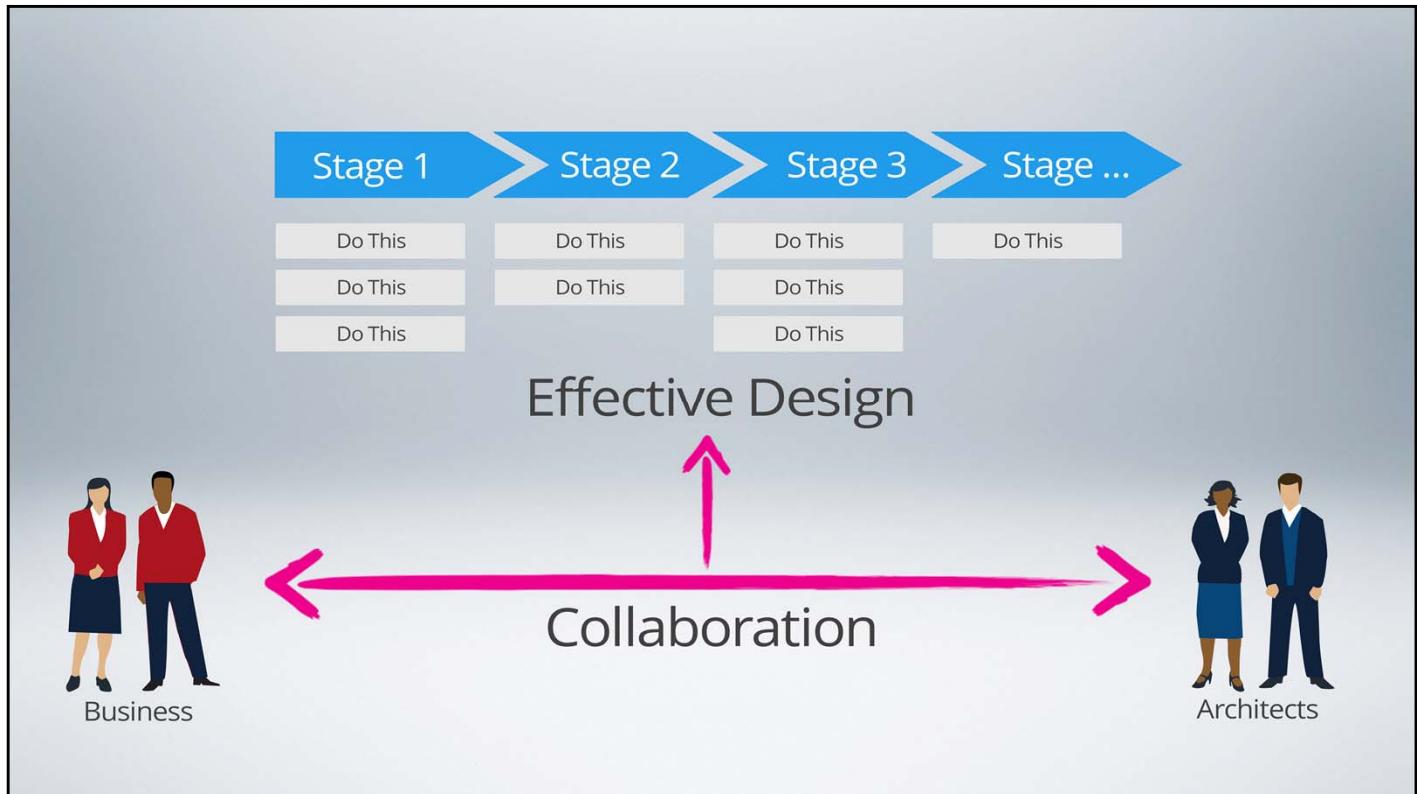
Let's be clear. There is no real right or wrong way to model a business process, and the decision about which way is best usually comes down to opinion.

However, establishing guidelines, or "best practices" – and...of course..., following them - helps us ensure we are consistent in how we represent a specific event or action when translating a business process into a case type – including building the individual flows.



**WHEN THE ENTIRE DEVELOPMENT TEAM IS FOLLOWING THE SAME SET OF BEST PRACTICES, THE ART AND SCIENCE BLEND READILY TOGETHER**

When the entire development team is following the same set of best practices, the art and science blend readily together.



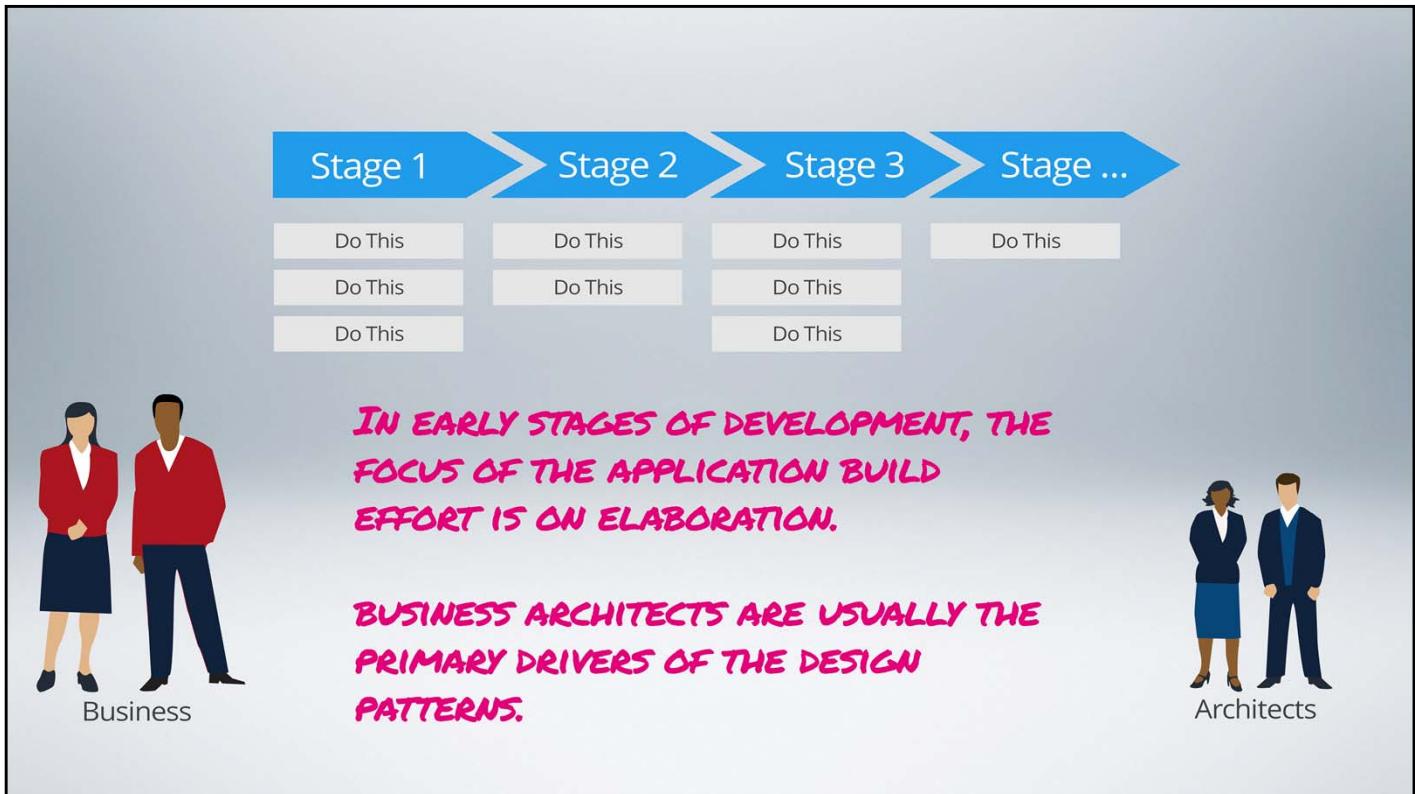
We've said this before - and will no doubt say it a few more times - collaboration is key.

Understanding how to define and represent each step is best accomplished when both the business stakeholders

and the development team contribute.

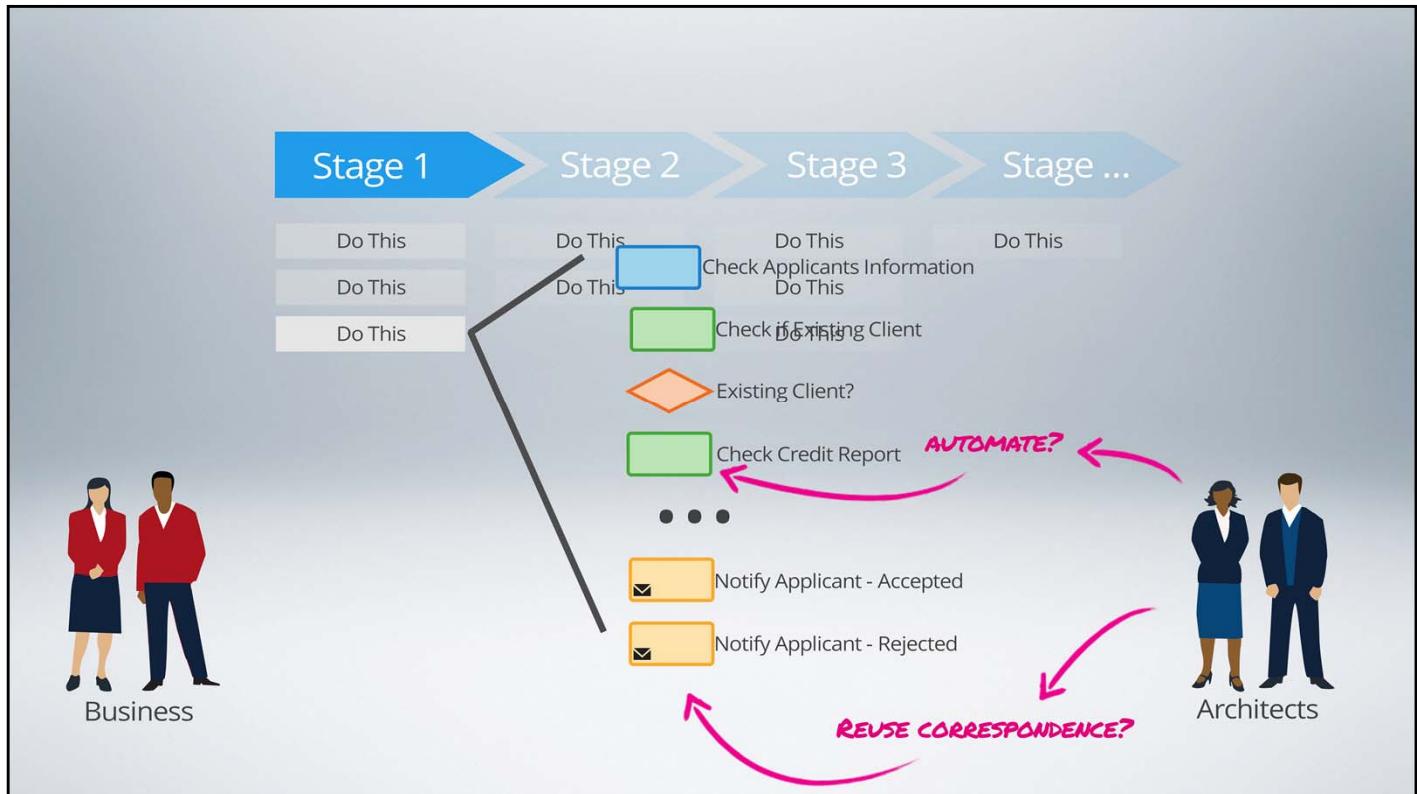
Effective process design

can only be accomplished when business stakeholders are engaged early on in the project. This ensures their interest and understanding throughout the entire application development lifecycle.



Remember that in the early stages of development, when the focus of the application build effort is on elaboration,

business architects are usually the primary drivers of the design patterns.

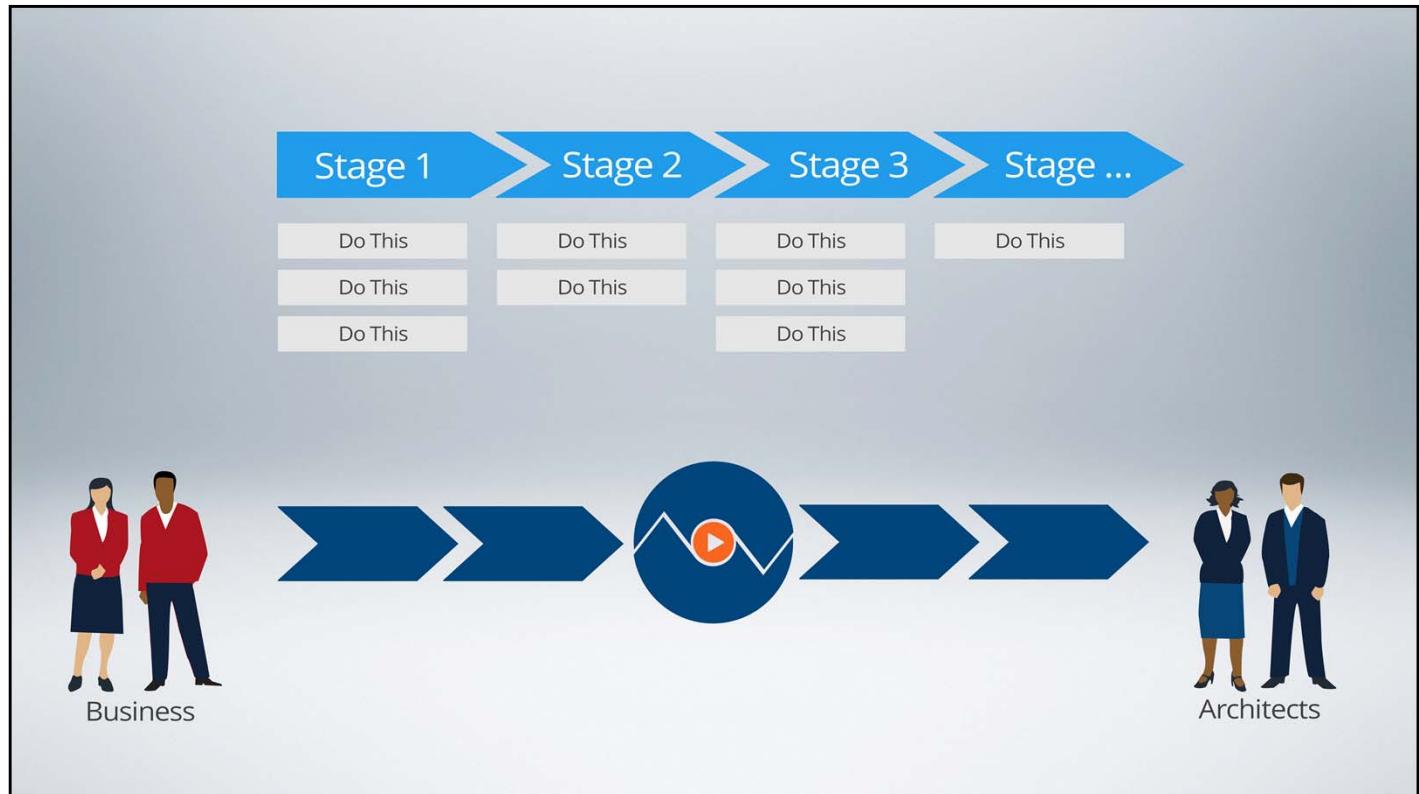


They make sure the case map and individual flows reflect the correct process, and that the steps are in the right order and well named.

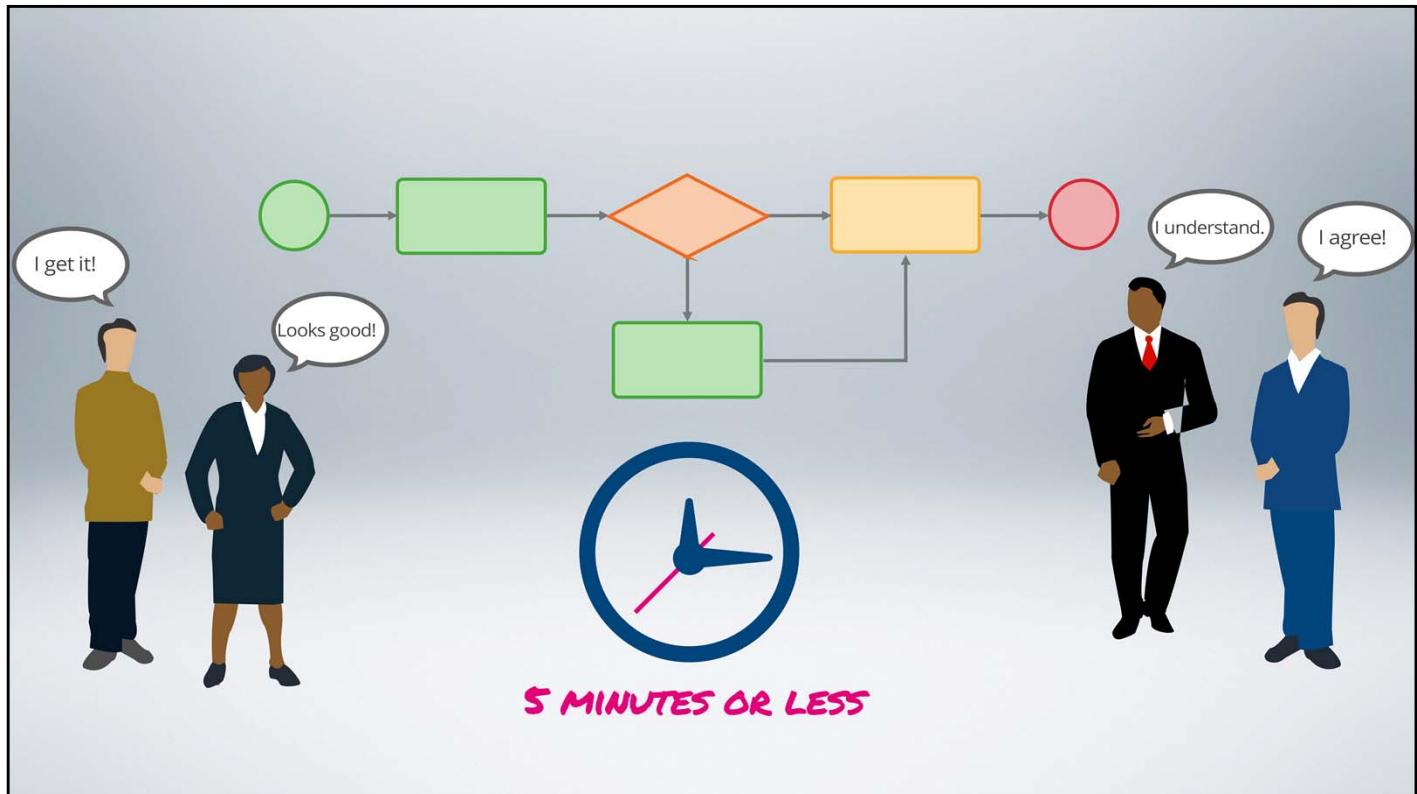
As system architects, our role during this effort is usually more consultative. We focus on areas of the flow that may be reusable,

look for manual steps that can be automated,

and keep an eye on efficiency.



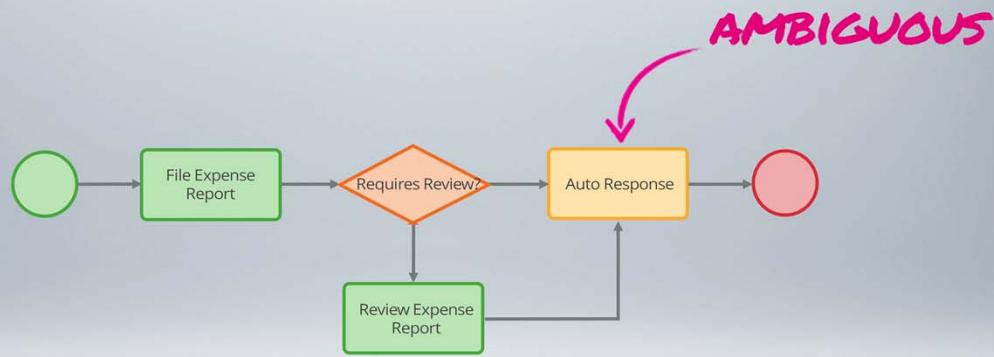
Finally, consider using “playbacks” – which are a focused demonstration of a partially implemented process flow with the goal of discussion, consensus building, collaborative improvement and, ultimately, approval of the model.



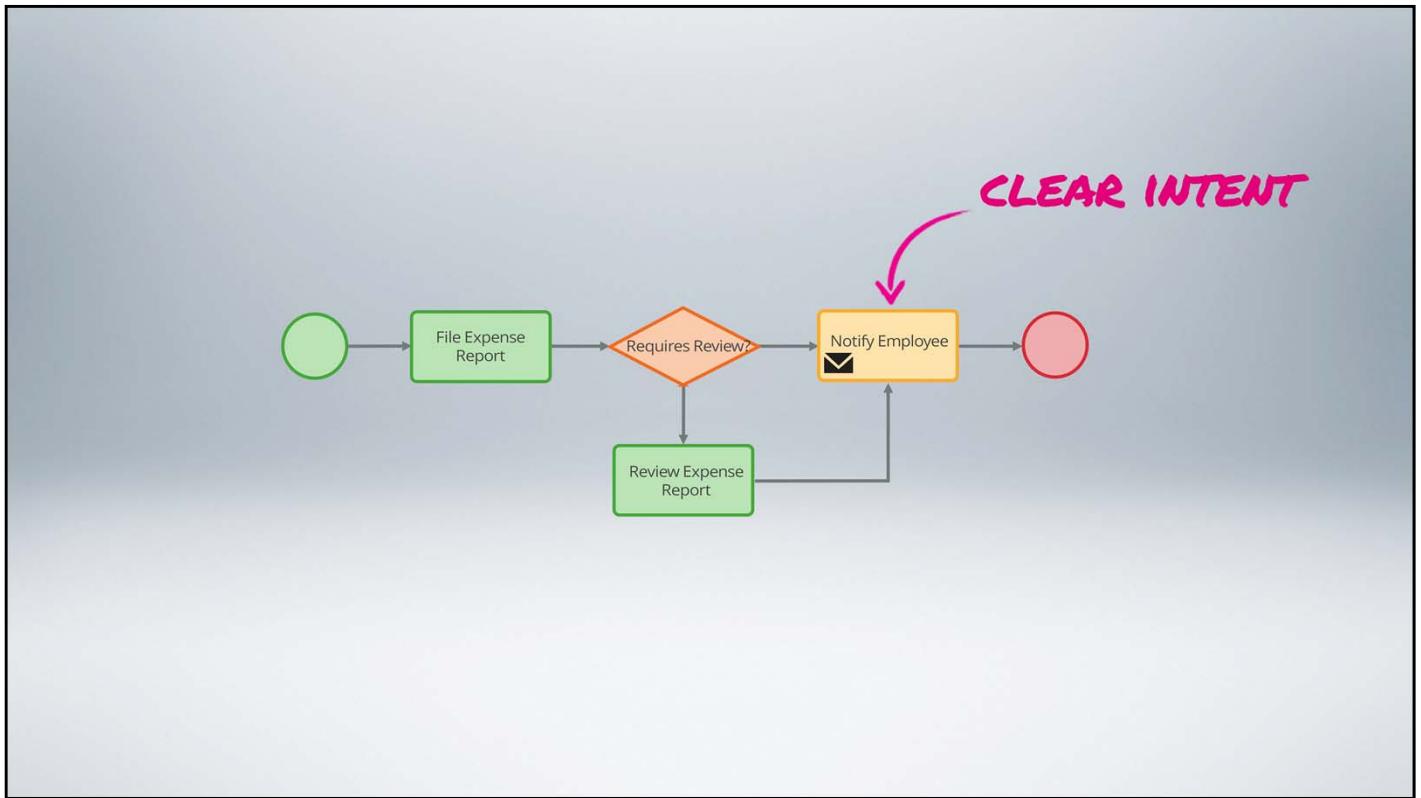
A good process flow diagram is a graphical representation of a business process that is universally understood by all stakeholders.

This means that everyone involved in the project recognizes the same concepts in the same context.

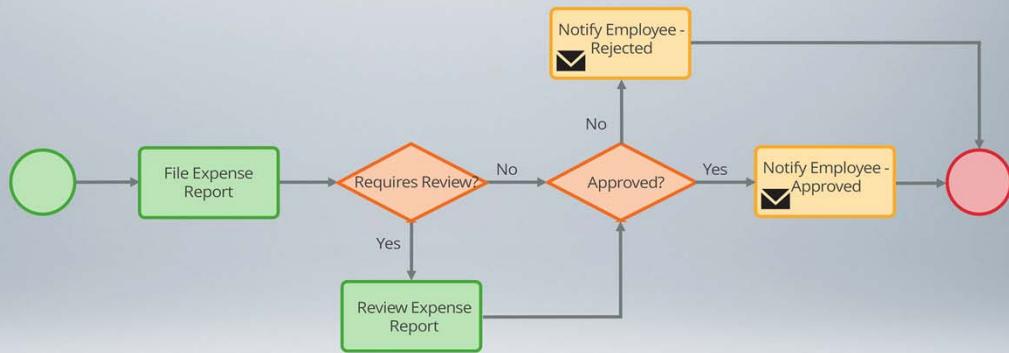
The flow diagram should provide a view into the process that is clearly and easily communicated in 5 minutes or less



Using a business vocabulary, every shape should be labeled so the intent and purpose is clear.  
Avoid ambiguous labels that might leave doubt as to what is actually happening.

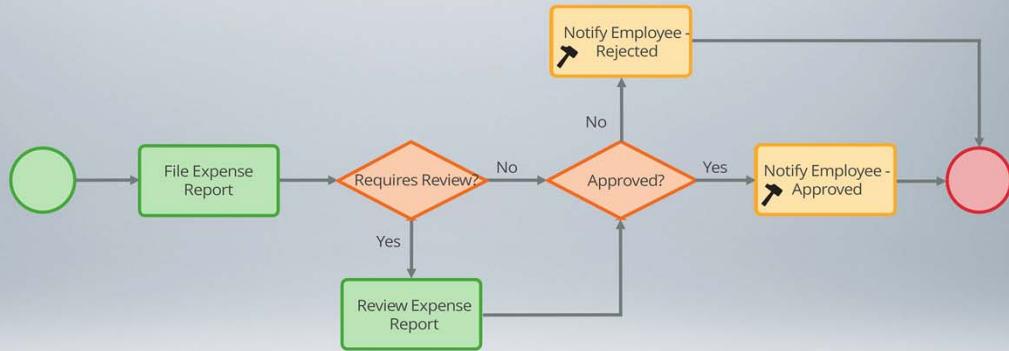


Instead, use labels which make the intent of the step very clear.

**EVERY SHAPE SHOULD HAVE A SINGLE INTENT**

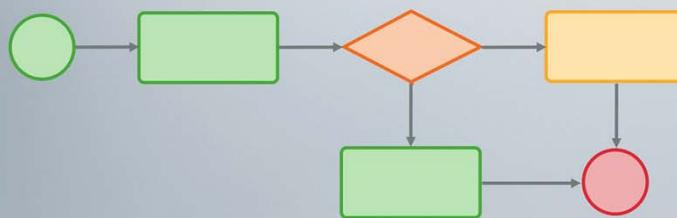
Every shape should have a single intent. For example, if we wanted to send a notification of an approval or a rejection, we would use two Email smart shapes – one for the approval notice and one for the rejection notice.

**EVERY SHAPE SHOULD HAVE A SINGLE INTENT  
AND FOR THEIR INTENDED PURPOSE**

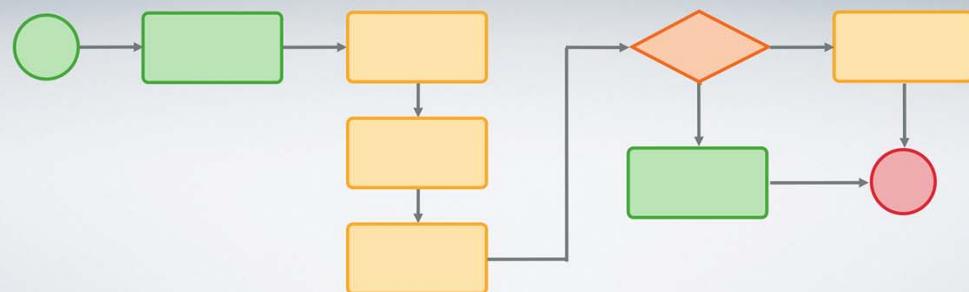


Use shapes for their intended purpose. This not only helps everyone involved understand what is happening but is a functional requirement as well.

What is shown to the business users

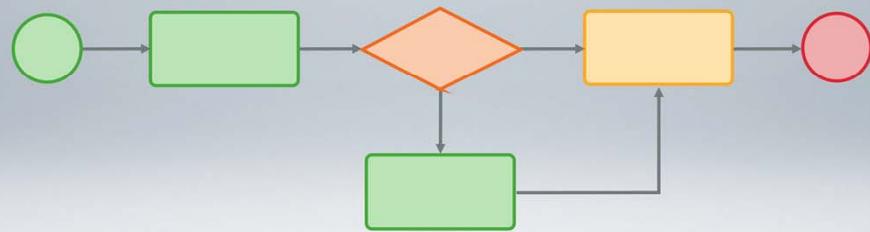


What is actually used



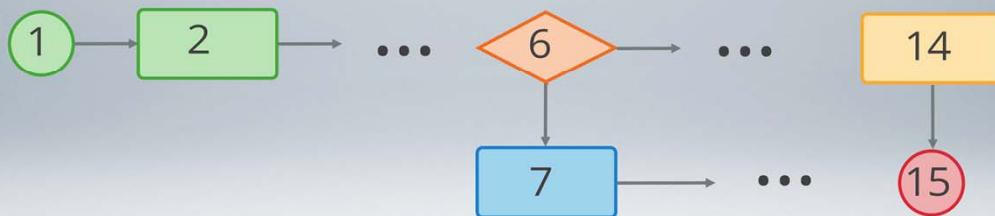
The flow diagram should be able to be implemented as drawn. There should be no need to re-draw a flow diagram in order to provide two separate views: one for the “business users” and one for “how it actually runs.”

## What we model is what we execute



There should only ever be a single representation of any given process.

## 15 OR LESS SHAPES ON A FLOW



at every level of granularity, so consider limiting the number of shapes on any given flow rule to 15 or less. Limiting the number of shapes on any given flow rule not only makes it easier to communicate the intent in 5 minutes or less, but also helps reduce modeling errors and makes maintenance much easier.

## 5 OR LESS CONNECTORS

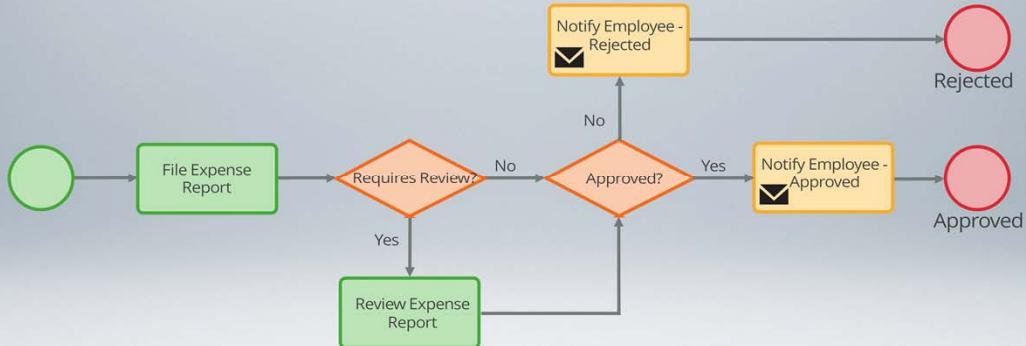


Limit the number of connectors emanating from assignments to no more than 5.

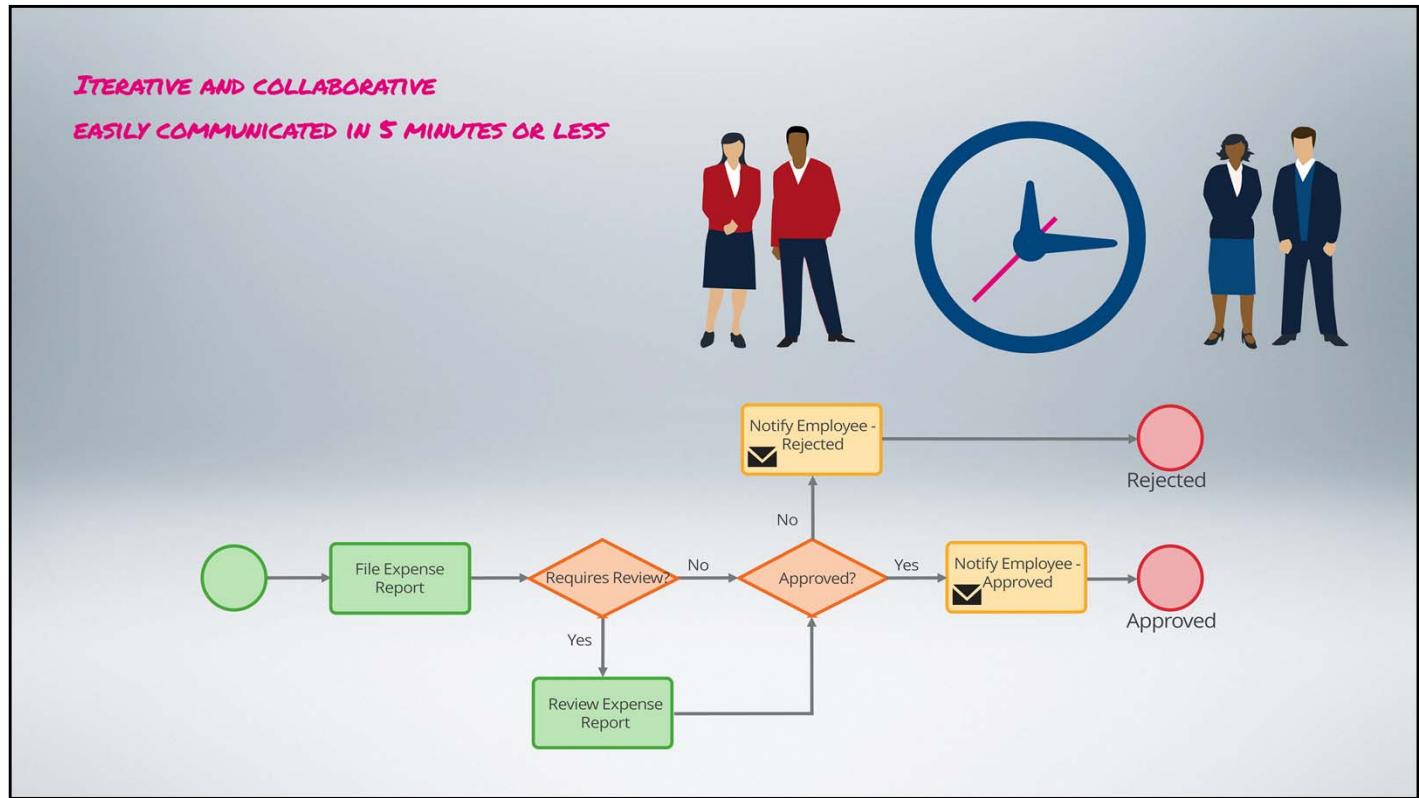
As the number of choices increases, the likelihood increases that an end user might choose incorrectly, or spend time thinking about which choice to make — which can affect productivity.

Limit the number of results emanating from a decision shape to five or less. The more results shown on the diagram the harder it becomes to follow the logic of the results.

## USE SEPARATE END SHAPES



Consider using separate End shapes to represent the different ways a process may end. This allows us to set a unique “Flow Result” and “Work Status” using fewer shapes.

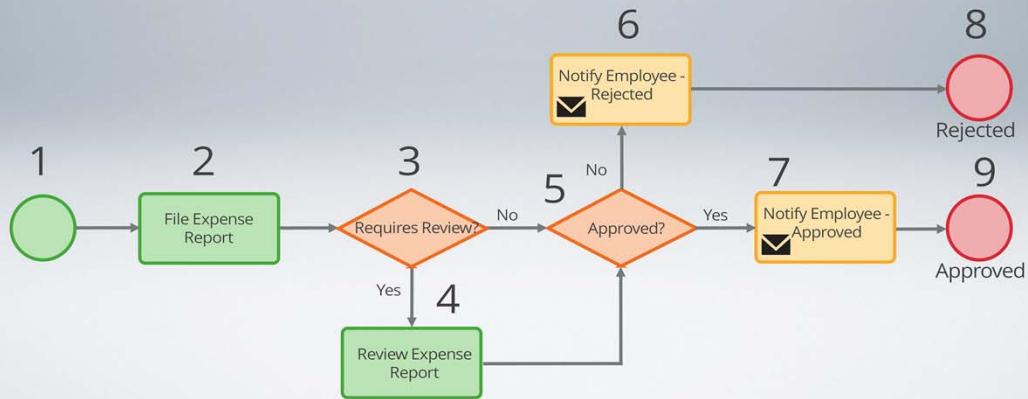


OK, let's see if we can summarize these “bullet points” into a single, cohesive picture.

First and foremost, process decomposition is an iterative and collaborative effort. Engage the business users early in the project lifecycle – and keep them engaged.

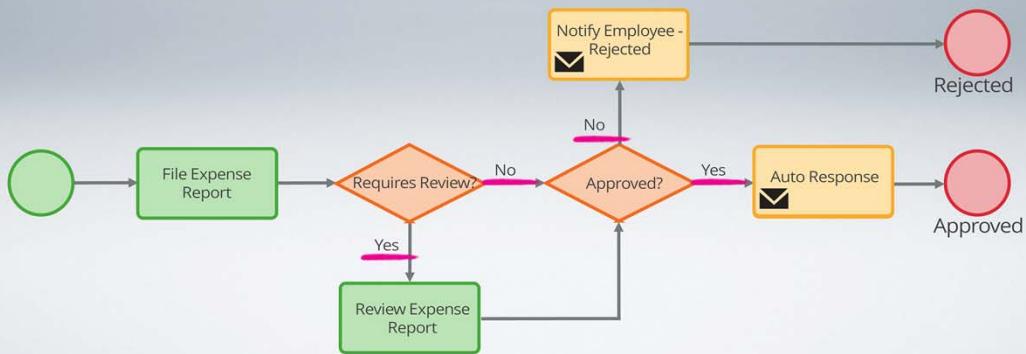
Each flow diagram should be easily communicated in 5 minutes or less.

ITERATIVE AND COLLABORATIVE  
EASILY COMMUNICATED IN 5 MINUTES OR LESS  
15 SHAPES OR LESS



Limiting the number of shapes to 15 or less helps makes this possible.

ITERATIVE AND COLLABORATIVE  
 EASILY COMMUNICATED IN 5 MINUTES OR LESS  
 15 SHAPES OR LESS  
 BUSINESS VOCABULARY  
 LABEL EVERYTHING



Using a business vocabulary,  
 Label everything so the intent and purpose is clear.

ITERATIVE AND COLLABORATIVE  
 EASILY COMMUNICATED IN 5 MINUTES OR LESS  
 15 SHAPES OR LESS  
 BUSINESS VOCABULARY  
 LABEL EVERYTHING  
 SEPARATE SHAPES FOR EACH ACTION



Every shape should have a single intent,

ITERATIVE AND COLLABORATIVE  
 EASILY COMMUNICATED IN 5 MINUTES OR LESS  
 15 SHAPES OR LESS  
 BUSINESS VOCABULARY  
 LABEL EVERYTHING  
 SEPARATE SHAPES FOR EACH ACTION  
 DON'T MIX AND MATCH SHAPES



and use each shape for its intended purpose. For example, use a *Send Email* smart shape to send emails instead of a generic *Utility* shape.

## Exercise: Effective Process Decomposition



# Guardrails for Case Management Design

**Guardrails for Case Management Design helps you establish best practices when developing Case Types in Pega 7.**

**At the end of this lesson, you should be able to:**

- Identify common Guardrail warnings encountered when designing case types
- Identify motivations for establishing best practices as additional guardrails
- Provide a summary of the best practices for stage-based case design

**RULE**

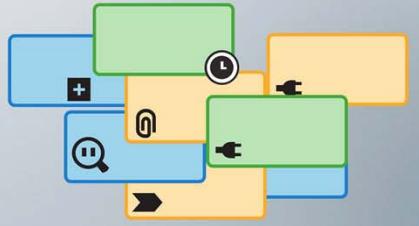


**FLOW**

Caution - Best Practice

**Flow in Draft Mode**

**Too many shapes**



This record has 2 unjustified warnings (view/edit)

Diagram    Parameters    Pages & Classes

Draft on    +    ↻    ⌂    🔍    ✎    X

GuardrailExampleFlow

**BestPractice: Caution**  
For clarity and ease of maintenance, limit the flow to no more than 15 shapes. This flow contains 18 shapes.  
[Add justification](#)

**LIMIT OF 15 SHAPES**

**EASIER TO BUILD AND MAINTAIN**

**PROVIDES CLARITY AND UNDERSTANDING**

**AVOID JUSTIFYING WARNING**

**CONSIDER USING ADDITIONAL FLOWS**

The most common guardrail warnings you will encounter during case design will be “Caution - Best Practice” warnings – and these cautions will almost always be found on a flow rule.

**RULE**

**FLOW**

Caution - Best Practice

Flow in Draft Mode  
Too many shapes  
Overloaded assignments

This record has 2 unjustified warnings ([view/edit](#))

Diagram Parameters Pages & Classes

BestPractice: Caution  
Offer workers no more than five connector flow actions in a single assignment for best productivity and accuracy. There is 1 overloaded assignment in this flow.

Draft on

LIMIT OF 5 CONNECTORS PER ASSIGNMENT  
REDUCES POTENTIAL FOR ERRORS  
INCREASES PRODUCTIVITY

AVOID JUSTIFYING WARNING  
CONSIDER USING ADDITIONAL ASSIGNMENTS OR SCREEN FLOW

For example, when a flow rule is first created, it is in “draft mode.”

This is useful as “draft mode” allows us to model flows with shapes that contain references to other rules that are not yet complete, or do not exist.

This allows us to build and test the flows incrementally without having to build all the other components because most errors are suppressed when the flow is in “draft mode.”

However, to protect against runtime errors in production, a flow in “draft mode” will not run in a production environment.

Given that a flow rule in “draft mode” will not run in production, this warning should never be justified – it must always be resolved. When the flow is complete, and ready for production, turn off draft mode. This is the only way to resolve the warning.

Another common best practice caution is too many flow shapes used for a given flow.

Pega recommends no more than 15 shapes for any given flow.

When a flow has less than fifteen shapes, development and maintenance are easier, because you can more easily see what's happening in the process and are less likely to make a change in one area of the process that has an adverse impact on another area.

As much as possible, work to resolve this warning.

To remove the warning, consider options for reducing the number of shapes such as creating additional flows – adding “steps” to the case type in the Case Designer - to separate the logic into related units of work.

Let's look at one last typical best practice caution for flows – an overloaded assignment.

This caution appears when an assignment shape in the flow has more than five connectors with flow actions.

When an assignment has more than five connectors with flow actions, at runtime, end users are presented with all of those options. As the number of choices increases, the likelihood increases that an end user might choose incorrectly, or spend time thinking over which choice to make — which can affect productivity.

Here again, try to avoid justifying this warning as often as possible.

To remove the warning, consider separating some of the flow actions into separate assignments – or using a screen flow if not already doing so.

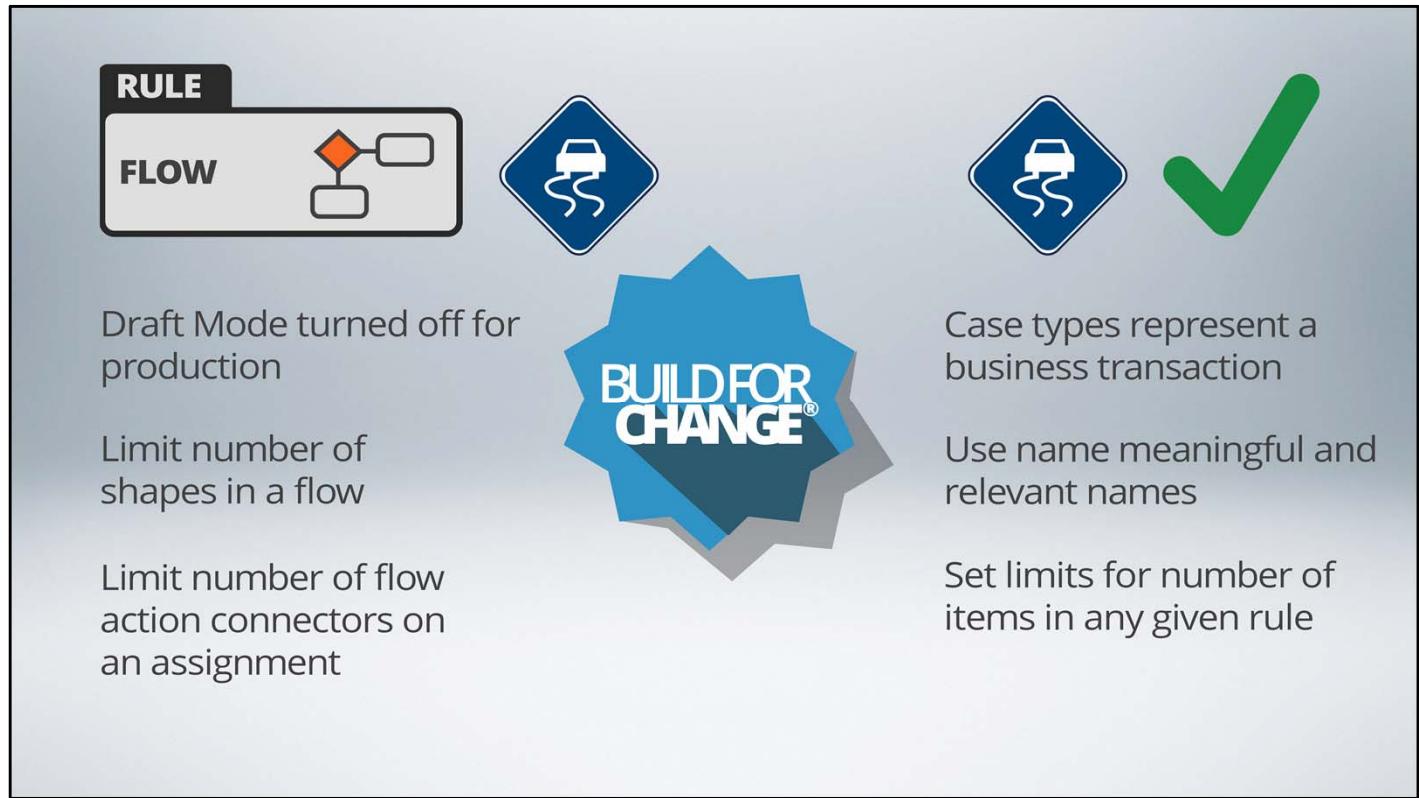


Designing a case type – actually, even the application - is not something you would do on your own.

There are a number of people involved in designing the application, deciding on the case types and the stages, and more technical details such as class structure and data models.

Stakeholders such as business executives, business end-users and subject matter experts business analysts - sometimes referred to as business architects – and more experienced Pega users such as lead system architects, are all involved in the decision-making.

Learning, and adopting best practices – whether they are defined in this course or in your company's center of excellence - will go a long way in making you an effective, contributing member of the team.



These best practices for case design become additional guardrails, which help ensure a consistent development effort and end-user experience.

Let's end this lesson with taking one last look at the best practices for stage-based case design.

Remember, case types should represent a specific business transaction – a case.

Use meaningful and relevant names when naming case types, stages or flows.

Set a limit to the number of items allowed in any given rule. For example, set a limit to the number of stages in a case type, or set a limit to the number of shapes on any given flow. These limits do not need to be chiseled in stone, however they help provide a marker for identifying when further decomposition is necessary.

Adhering to best practices for stage-based case design and conforming to the guardrails for flows lead to applications that are well designed, straightforward to maintain, and architected to *Build for Change*.

## Exercise: Designing Enterprise Applications Using Case Management Exercise Verification



## Module 04: Creating an Effective Data Model

This lesson group includes the following lessons:

- Best Practices for Designing a Data Model
- Best Practices for Managing Data
- Best Practices for Managing Reference Data
- Sharing Data Across Cases and Subcases
- Guardrails for Data Models

# Best Practices for Designing a Data Model

In this lesson, you will learn the best practices for Designing a Data Model in Pega 7.

At the end of this lesson, you should be able to:

- Understand the two different types of data in an application
- Understand how different participants view the data in an application



Just like any application, one of the more important things to determine is how much data to expose? There's usually an abundance of data in a lot of different places, choosing what the right data is can be a daunting task. To determine this, we need to talk about a couple of things, first we need to discuss the two different types of data; flow data and business data, we also need to discuss the three participants view of that data.

The diagram features a blue wavy logo above the text "flow data". To the right, the title "What and Who" is displayed. Below the title is a large grey suitcase icon containing a process flow diagram. The flow starts with an oval, followed by a diamond, a rectangle, another rectangle, and a third rectangle. From the second rectangle, a branch goes down to a fourth rectangle. A pink curved arrow originates from the bottom of the fourth rectangle and points upwards towards the text "TOTAL PRICE > THRESHOLD" in pink capital letters.

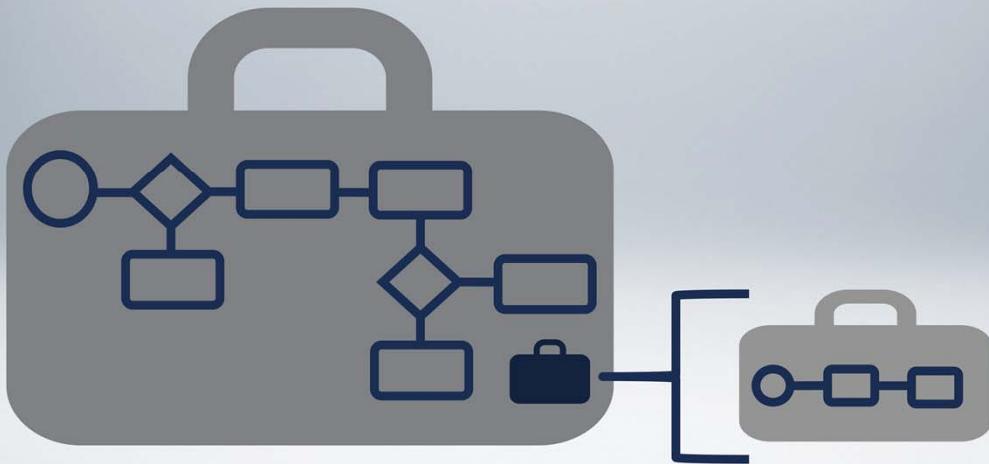
Let's examine the first type of data which is flow data.

Flow data is the information needed by an application to determine what to perform and who should perform it.

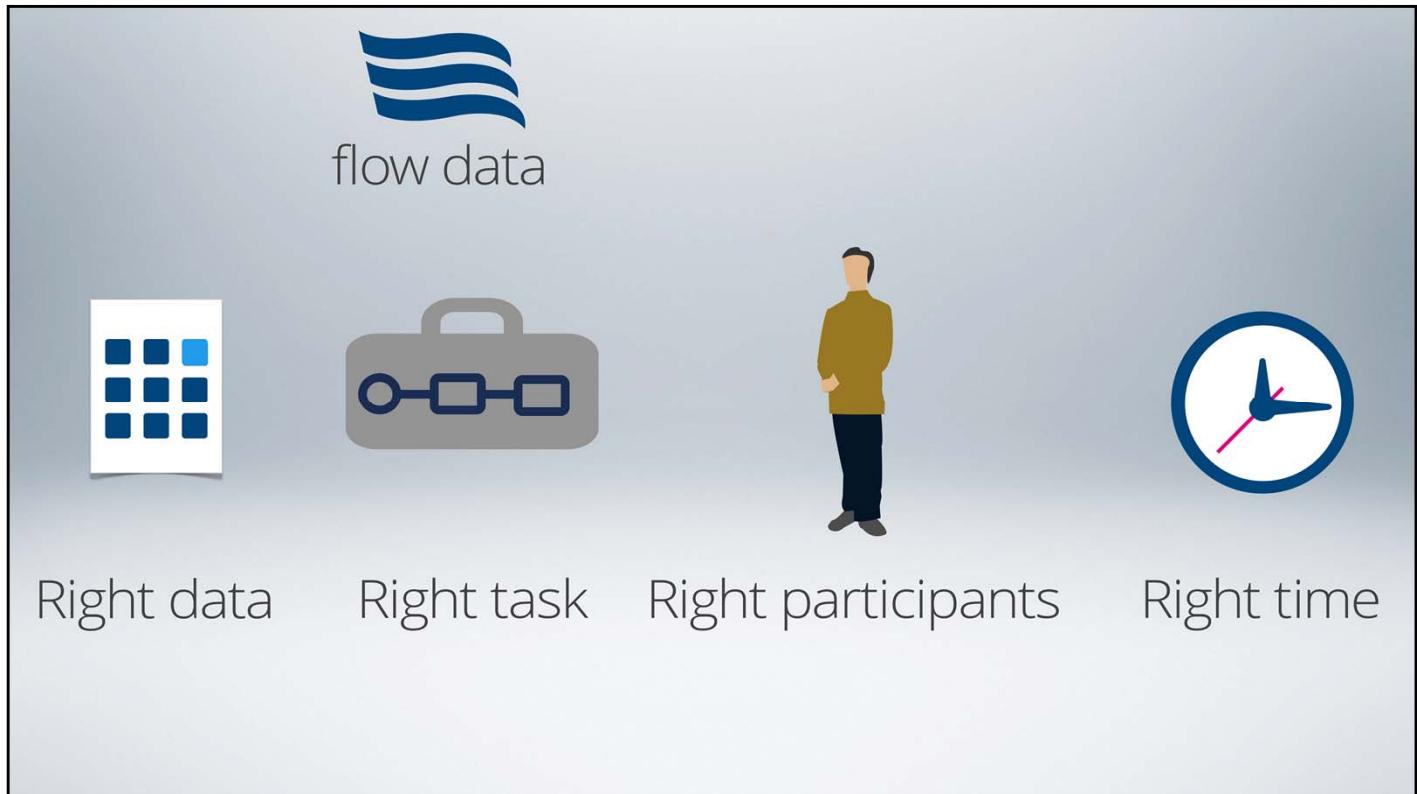
For example, we may have a decision table that uses the purchase order total to determine if a manager approval is required because the total exceeds a threshold.



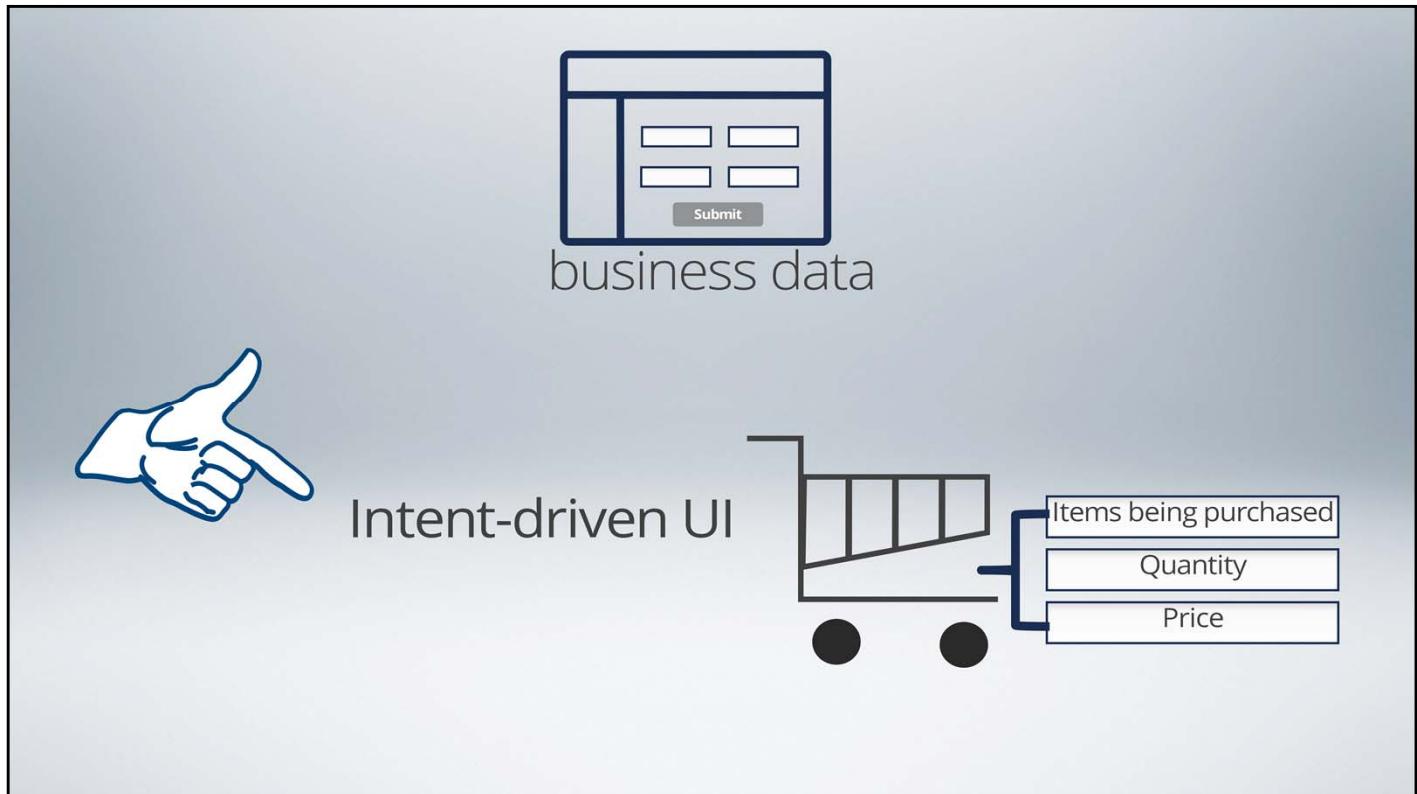
## What and Who



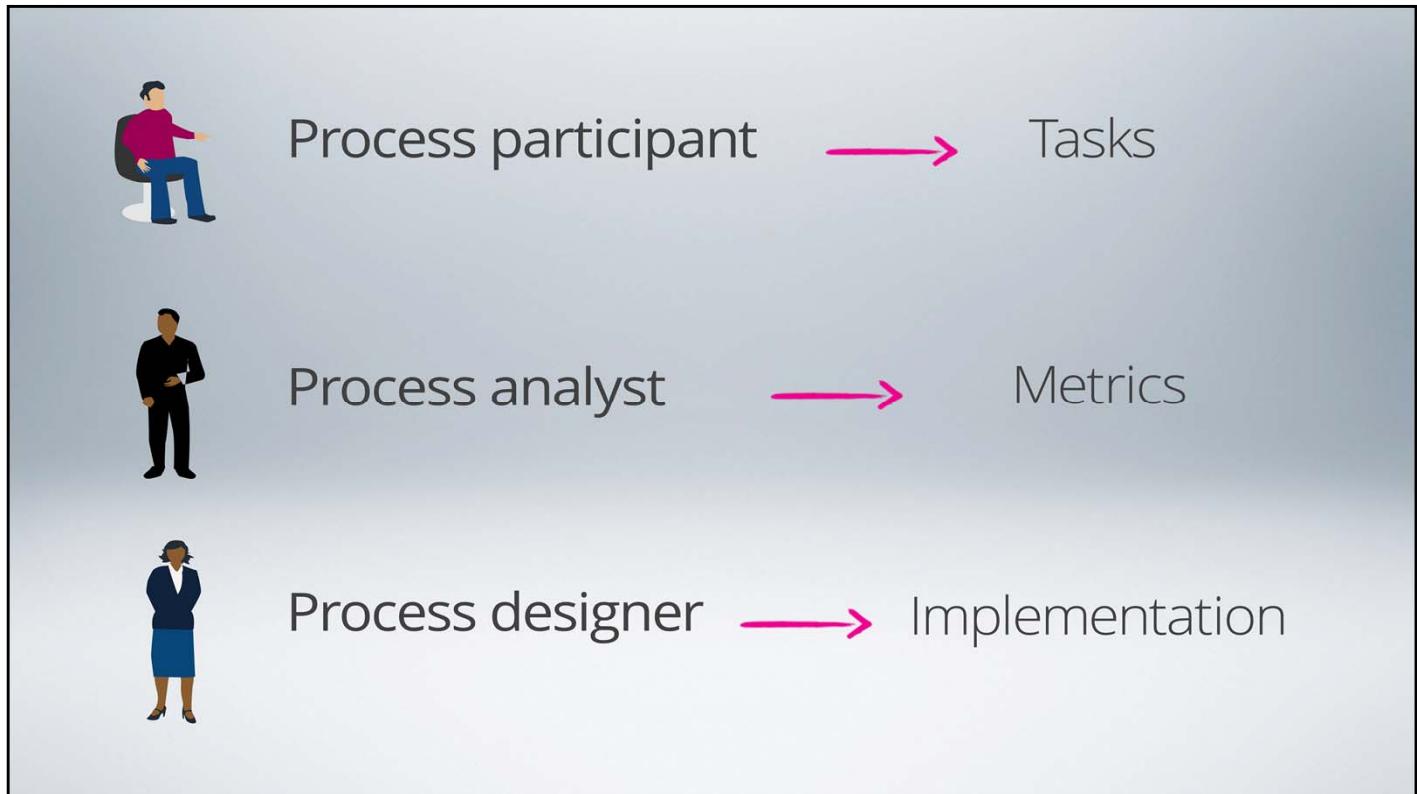
Additionally, it's also possible that flow data could cause a subcase to be created.



Flow Data must be identified very early on in the implementation process, and is the data needed to get the right data to the right task to the right participants at the right time. Without flow data, the process won't work. Flow data can be gathered directly from the Participants, but it can also be retrieved from external sources including, LDAP repositories, databases, and Web Services.



Business data provides context for a given task for a given participant. Pega strongly encourages you to create Intent-driven User Interfaces. An intent-driven User Interface is a screen where the user has no issue determining what is being asked of them to do. Intent-driven UIs help us focus on providing the right business data to the right participant at the right time. For example the form used to review a shopping cart for a purchase should only have the relevant information inside it like the items being purchased, quantity, and price. The form should not contain items like shipping address or department being billed, these data elements are extraneous for reviewing a shopping cart and could distract the user from the task at hand.



When creating an application we need to look at the data required from three different vantage points. The first vantage point is the process participant. The process participant has a task level view of the data in the application as they are responsible for entering data into the application. The next vantage point is from the Process analyst. A process analyst cares about the data needed for reporting, metrics and analysis. The final vantage point is from the Process designer. The process designer looks at the data from an implementation point of view and what data structures should be used to support the needs of the other views.

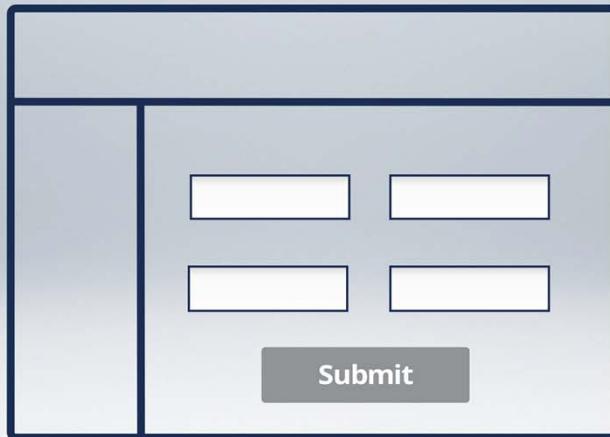
## Process participant



- ✓ Task level view of data
- ✓ Only concerned about data for task at hand

Let's discuss the process participant view a little further. Participants complete tasks in the process and have a task-level view of data. Their perspective of process data is immediate. The process participant is only concerned if they can access and modify the data needed to accomplish their tasks. When determining what the process participant view of data consider the following questions:

## What forms need to be filled out?



Let's discuss the process participant view a little further. Participants complete tasks in the process and have a task-level view of data. Their perspective of process data is immediate. The process participant is only concerned if they can access and modify the data needed to accomplish their tasks. When determining what the process participant view of data consider the following questions:

## What fields should be on each form?

	Name	Price
	<input type="text"/>	<input type="text"/>
	Qty	Total
	<input type="text"/>	<input type="text"/>
	<b>Submit</b>	

Let's discuss the process participant view a little further. Participants complete tasks in the process and have a task-level view of data. Their perspective of process data is immediate. The process participant is only concerned if they can access and modify the data needed to accomplish their tasks. When determining what the process participant view of data consider the following questions:

## Do these fields have restricted values?

	Name	Price
	<input type="text"/>	<input type="text"/>
	Qty	Total
	<input type="text"/>	<input type="text"/>
	(1-5)	
	<b>Submit</b>	

Let's discuss the process participant view a little further. Participants complete tasks in the process and have a task-level view of data. Their perspective of process data is immediate. The process participant is only concerned if they can access and modify the data needed to accomplish their tasks. When determining what the process participant view of data consider the following questions:

## If data is changed, do these changes need to be seen immediately?

	Name	Price
	<input type="text"/>	\$10
	Qty	Total
	<input type="text" value="3"/> (1-5)	<input type="text" value="\$30"/>
	<b>Submit</b>	

Let's discuss the process participant view a little further. Participants complete tasks in the process and have a task-level view of data. Their perspective of process data is immediate. The process participant is only concerned if they can access and modify the data needed to accomplish their tasks. When determining what the process participant view of data consider the following questions:

## Process analyst



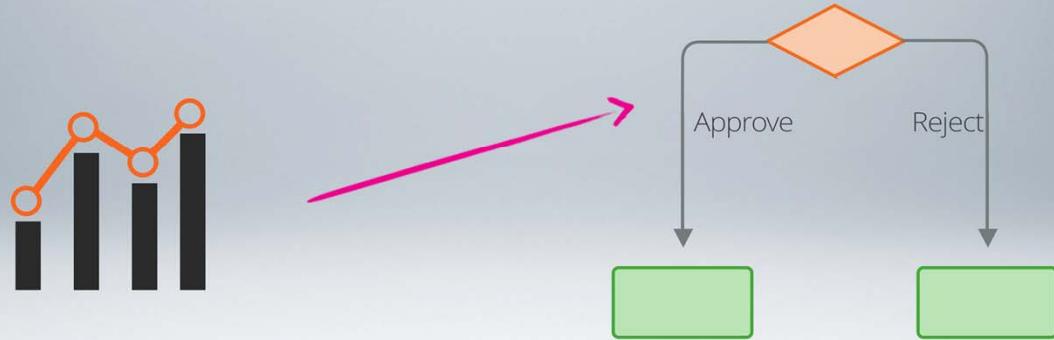
- ✓ Reporting, metrics and analysis
- ✓ Understand the past to improve the future

Process analysts care about data that is used for reporting, metrics and analysis. Analysts deal with understanding the past to improve the future and use historical metrics to figure out how processes can be improved.

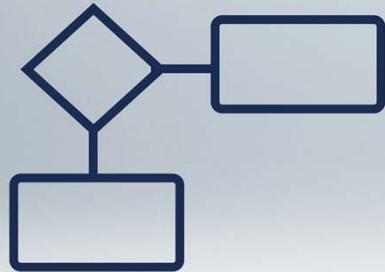
## What information needs to be tracked?



## What information is analyzed and used to make decisions?



## What data is needed after the process is completed?



Other processes



Reporting

## Process designer



- ✓ Implementation view of data
- ✓ Think of future data changes and maintenance

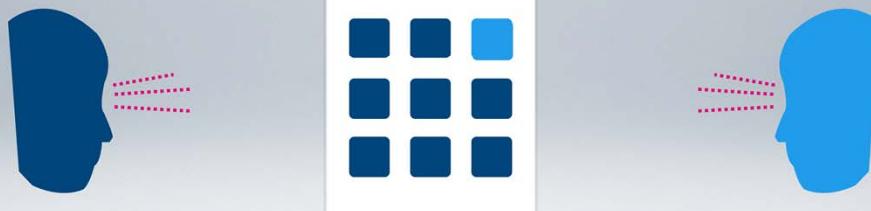
Process designers have an implementation view of data and want to achieve the most efficient data structure for development. In addition to coalescing the owners, analysts, and participant's models, it is also important of Process Designers to think of future data changes and plan for maintenance.

## How do we organize the data we need for reusability?



When determining what the process designers view of data is consider the following questions: How do I organize the data I need for reusability?

## How do we manage data visibility?



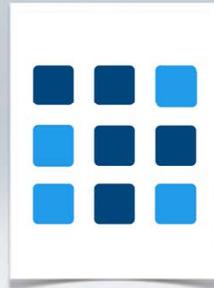
How do we architect data to make it visible to everyone that needs to see it and invisible to people that don't need to see it?

## How do you architect data to optimize reports?

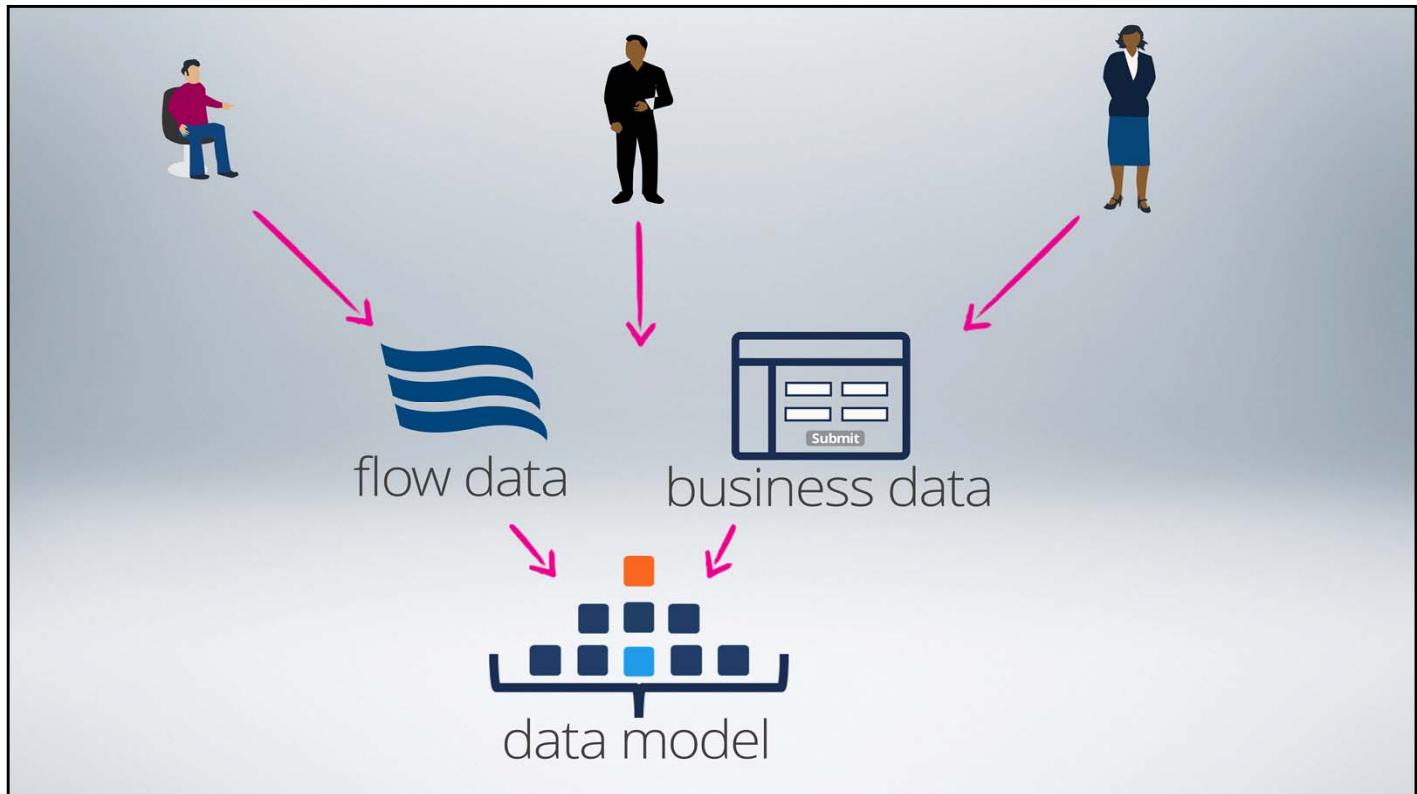


How do you architect data to optimize reports?

Are there any upcoming changes that will affect data?



Are there any upcoming changes that will affect data?



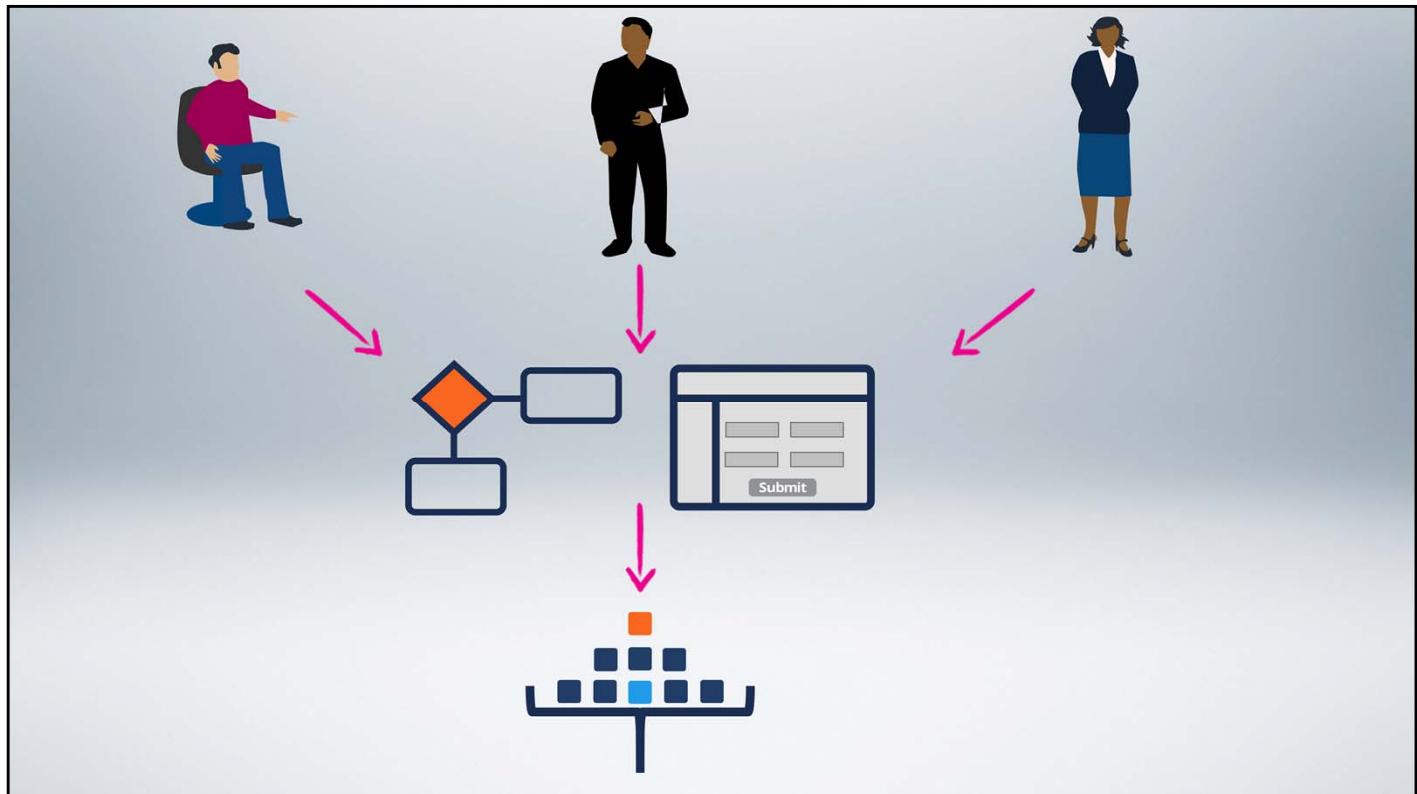
Once we understand the data needs from all stakeholders we can build a data model. Identifying the flow and business data needed at the beginning of a project will take longer than what you normally may do but the benefits are that you should expect to see fewer changes to the data model because of new or unexpected requirements.

# Best Practices for Managing Data

In this lesson, you will learn the best practices for managing data in Pega 7.

At the end of this lesson, you should be able to:

- Describe how to map a data model to data classes
- State at least three best practices for naming data classes
- Create data classes and properties
- Describe the relationship between a data class and a work class
- Use a data class from a work class



Now that we have discussed that a Data Model is comprised of both flow and business data and that data comes from various stakeholders let's dig a little deeper into what comprises a data model.



## Organization and Structure

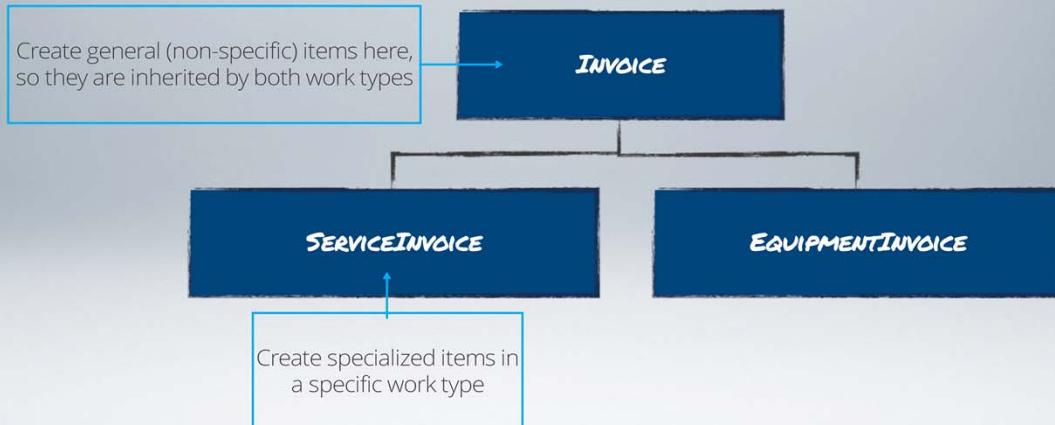
Cross-organizational effort



So what exactly is a data model? A data model is the organization and structure of data and consists of data objects, data object elements, and their associations. Data models are defined outside of Pega. Creating a data model is a cross-organizational effort, this is not something you will build alone in a vacuum. It is very possible that a data model already exists and it is up to you to determine how much of the data model you need in your application.

For example you may define an Invoice data object that has 1 or more Lineitem data objects associated with it. A Lineitem data object has ItemID, Quantity and TotalCost data elements and is associated with an Invoice.

# Consider Generic and Specialized Objects



As always you want to identify which data objects are *generic* and which are more specialized. Let's look at our Invoice example, we may decide that we want to further specialize this class by creating two new data classes one for Service related invoices and one for Equipment related invoices. In the Services data class you would expect to see additional properties for contract information like hours scheduled where you wouldn't need such properties for equipment.

# Map data model to data classes

Data class = template



```

Pco-FW-PurchaseFW-Data-Invoice
  Data Model
    Property
      AttachmentDescription
      AttachmentKey
      eInvoice
      ExpectedShippingTotal
      ExpectedTaxTotal
      ExpectedTotal
      InvoiceDate
      InvoiceID
      InvoiceTotal
      LineItems
      Payment
      Resolved
      ShippingCost
      Subtotal
      Tax
      VendorID
    Decision
    Process
    SysAdmin
    User Interface
    EquipmentInvoice
    ServiceInvoice
  
```

We've talked about some of the basic principles for data modeling, the next task is to take that data model and represent that in PRPC. In PRPC, data objects are represented using data classes. Data classes are just like other classes except that they inherit from the Pega standard class Data-.

The Data- class has a lot of additional properties and subclasses that provide templates for some commonly needed data types, for example a Data-Address type. You can think of a data class as a template for how you want the data to be structured.

Data classes use properties to represent the data elements in a data object.

Remember that properties can either be single valued, like Payment, or a page property that represents multiple elements, like LineItems.

We can also take advantage of our class structure and create more specialized data classes that just add the new additional properties. The Equipment and Service classes can be seen when observing the Invoice data class in the Application Explorer.



Use Nouns



Customer Lineltem

Be simple & descriptive → PurchaseOrder

Use whole words



Amount not Amt

Now that we understand data classes, what should we call them? When working with data classes we should follow these guidelines when naming them: Use a noun for a class name such as Customer or Address.

Use camel case with the first letter of each word capitalized.

Make class names simple and descriptive. Avoid using technical jargon.

Use whole words. Avoid acronyms and abbreviations unless the abbreviation is much more widely used than the long form, such as URL or HTML.



Letter and alphanumeric → Address1

CamelCase → UseShippingAddress

Case Sensitive → emailaddress  
EmailAddress

Distinct, unique  
property names → PrincipalAmount  
instead of Amount

Additionally, when naming properties keep the following guidelines in mind. Start every property name with a letter and use only alphanumeric characters and dash characters.

Use CamelCase for property names, capitalizing the first letter of each word in the name, such as UseShippingAddress.

Remember that property names are case sensitive; "emailaddress" and "EmailAddress" are two distinct properties.

Use distinct, unique property names within an inheritance path to avoid poor runtime performance. Instead of using a property named Amount, use more descriptive names such as PrincipalAmount or TotalOrderAmount. When creating a property using the same name as an existing property in the inheritance path, PRPC presents a warning message and lets us to confirms whether we want to create the property.

**Data****No** @ or \$**No** Special Characters

# % \*

**No** Punctuation Marks

! ? . -

**No** Reserved Words

Top, Parent, Local, Param, Primary

Additionally here are some things to avoid when naming data classes or properties. Do not use @ and \$ in a property name even though they are legal Java identifier characters, as they are not legal characters in a property name.

Do not use special characters.

Do not use punctuation marks in a property name, for example, dashes, dots, etc.

Do not use a property name that matches a reserved page name or keyword, such as Top, Parent, Local, Param, or Primary.

## Demo



## Creating Data Classes

We have spent most of our time using the Case Designer to create your application. You've also been exposed to the Application Explorer to see what the structure of our application is. But what about data? We can use the Data Explorer to find out more about the data classes that exist for our application.

The Data Explorer shows a summarized view of the data classes used in our application. It is not the exhaustive list of all data classes but we can add more if we need to. For instance say we want to look at the Invoice data class. To do this we need to add it to the view.

This list shows us all of the classes we can currently see in the Data Explorer, if we want to remove one from the view we can uncheck one of the classes. This does not delete the data class it just removes it from the Data Explorer view. By clicking View All we get the complete list of all of the data classes in our application.

This is a pretty large list, remember that we are looking to add the Invoice data class, let's try searching for it.

To add the Invoice class, select it and click OK.

Invoice is now added to our list of data classes in the Data Explorer. If we click the Invoice class we can see what properties are defined for it.

Show selecting of Invoice and clicking it.

What if we want to create a new data class? Let's say we want to add a new data class type that extends Invoice and is specific to Services. We can do that from where we modified the list of classes we wanted to view.

Instead of adding an existing class to the list we want to create a new one, therefore we need to click the Create New link.

Let's name it ServiceInvoice and since we want it to inherit from the Invoice class let's add that as the parent class for Pattern inheritance.

On this next screen we can add the properties we want to our data class, and set what type they are.

Next we are given the option to define the display type for your properties

Lastly, we are given a confirmation screen for the properties we want to add to the data class. If what we see here is correct we can click Finish.

We can now see the class has been added to the Data Explorer, select it to see the properties that we just created.

Show selecting of the ServiceInvoice and give it a pause so students can take a look at the right side to see the properties.

Now let's examine what was created in the Application Explorer.

Instead of the Work class, we need to examine the Invoice class. Notice that Data classes are in a separate class structure from the Work classes.

The ServiceInvoice class is located as a subclass of Invoice, we can expand it to see the properties we added.

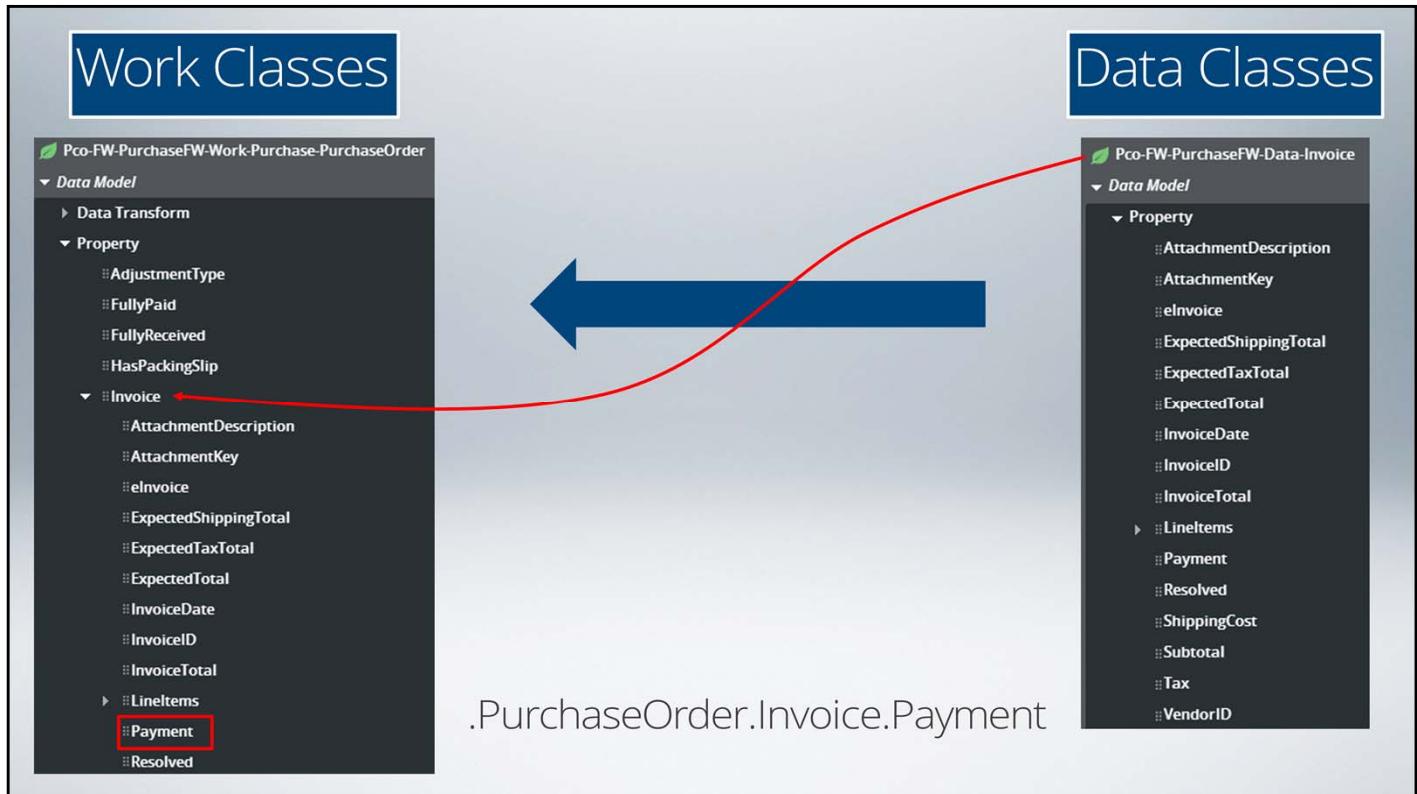
## Pega 7.1.6 Update Notes

### Updates for Pega 7.1.6:

Ease-of-use updates made to forms

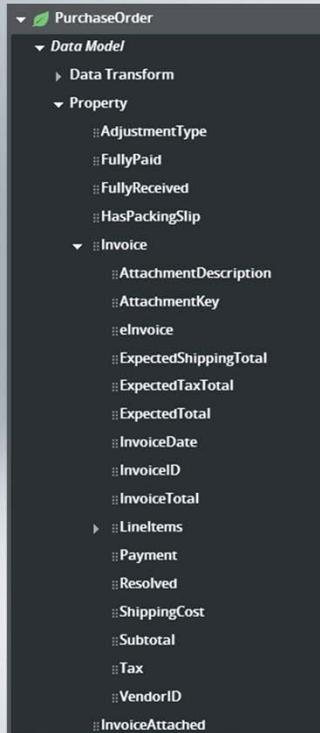
- Submit button replaces Finish button in last step of data object type creation
- +Create menu option replaces New in Explorer menus
- The Create and Save As forms now include the option to set record context.

The screenshot shows the 'Create Property' dialog box. At the top right are 'Create and open' and 'Cancel' buttons. The main area is divided into sections: 'PROPERTY RECORD CONFIGURATION' and 'CONTEXT'. In 'PROPERTY RECORD CONFIGURATION', there is a 'Label\*' field containing 'Cell Phone Number', an 'Identifier' field showing 'CellPhoneNumber' with an 'Edit' link, and a note: 'A short description or title for this record'. Below is a link 'View additional configuration options'. In the 'CONTEXT' section, there is a 'Choose app layer' section with radio buttons for 'Employee Reimbursements' (selected) and 'PegaRULES'. To the right are 'Apply to\*' and 'Add to ruleset\*' dropdowns, both currently set to 'MainCo-EmpReimb-Work' and 'EmpReimb'. A 'View all' link is also present.



Remember that we can think of data classes as templates for how we want the data structured. How do we instantiate one of these templates and actually use it in an application? To do that, we add a Page property to a work class and specify the data class we want to use. In our example we've been looking at the Invoice data class, the PurchaseOrder work class uses the Invoice data class as a property. Notice in the properties for PurchaseOrder there is an Invoice property and all the properties in the Invoice data class are listed.

To access the properties inside the Invoice class use the same dot notation that you have used before. So if you wanted to access the Payment property you would use .PurchaseOrder.Invoice.Payment.



How are the properties given values?

Mostly entered by the user  
as they use the application

**REMEMBER:**

**DATA CLASS IS USED IN A WORK CLASS**

**DATA CLASS ITSELF WILL NEVER BE POPULATED –  
JUST A TEMPLATE**

Now that we have created a data class from a data model, used a data class in a work class, how does the data get populated in the work class? It's important to know that the data class itself will never be populated, it is merely a template of how you want the data structured.

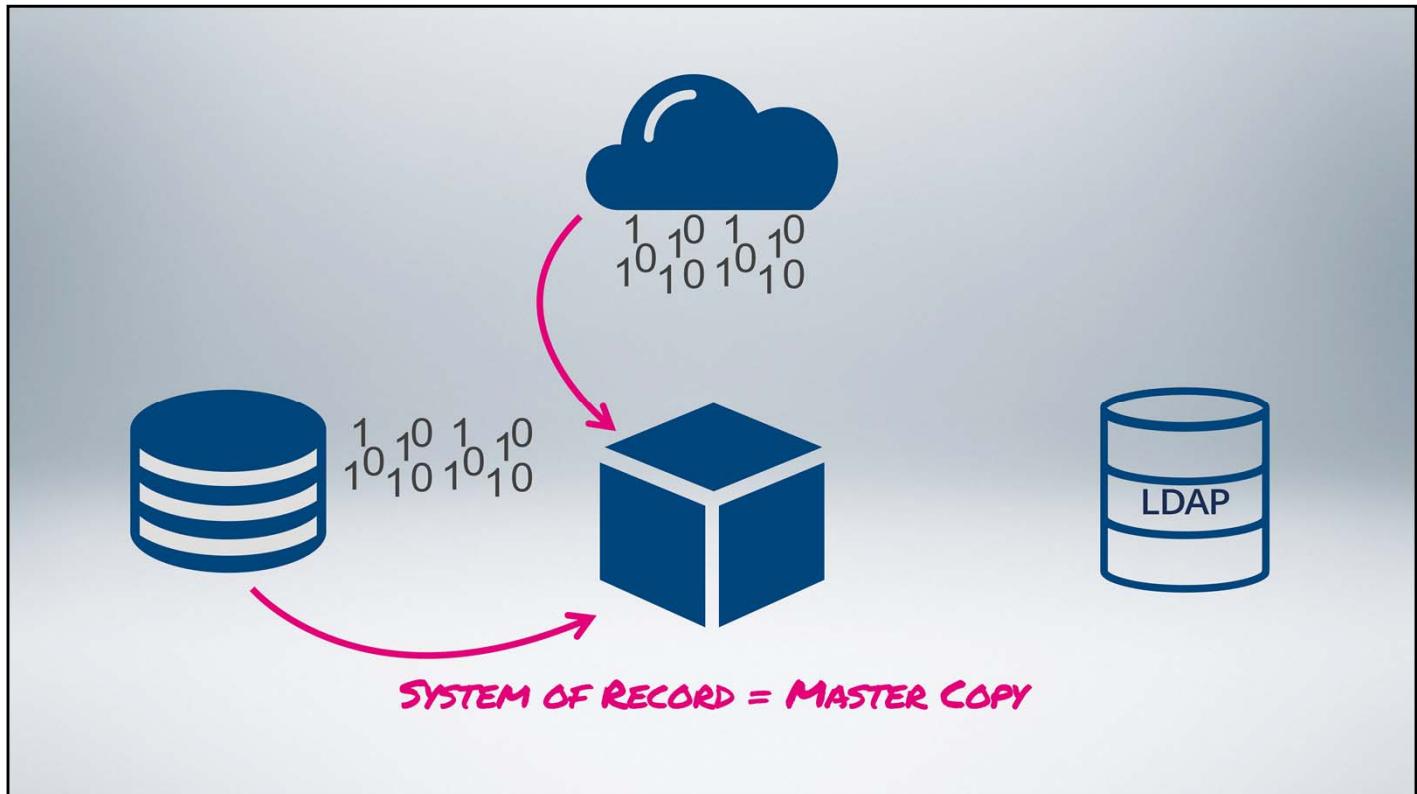
There are a few places where data can reside, for now though the data we are concerned with is being entered by the user as they use the application. For example, with our PurchaseOrder, properties such as Lineltems are being entered by the user in the application.

# Best Practices for Managing Reference Data

**Applications often need to access reference data that can be used for any number of things. This lesson introduces the concept of data pages as the best practice to manage reference data in an application.**

**At the end of this lesson, you should be able to:**

- Describe what a system of record for data is and how it impacts an application
- Create data tables to manage reference data
- Define the pros and cons for two ways to access reference data in an application
- Discuss refresh strategies and scope options for data pages
- Understand how parameters affect sourcing of data with data pages
- Create a data page using a data table to source the data



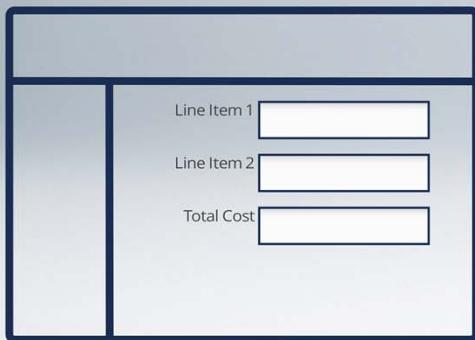
In most companies data can be spread out in a lot of different places throughout the enterprise. A system of record is the master copy for a given data element or piece of information.

The need to identify systems of record is important in organizations where applications have been built by taking output data from multiple source systems, reprocessing this data, and then presenting the results again for a new business use. There is always a single system of record for any given piece of data even though it could be replicated in multiple locations.

A system of record is not restricted to one centralized location. What does this mean? It means that a user's personal information might be stored in an HR database and their login credentials stored in an IT database. Each is the system of record for that specific data. Which systems are systems of record is another example of something that an entire organization determines.



Pega System of Record = Transaction of case  
Some reference data



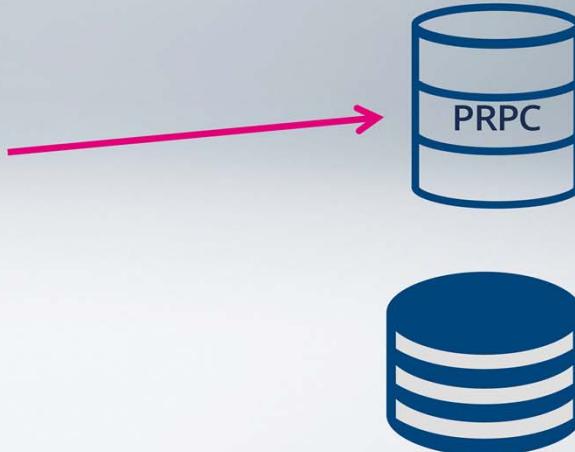
Is a Pega application ever the system of record for anything? The answer of course is yes.

Pega is the system of record for the transaction of any case that is created.

It can also be the system of record for some of our application reference data. There are two scenarios when we need to access a system of record, when we need to pull data from the system of record and when we need to push data to a system of record.



Pega System of Record = Transaction of case  
Some reference data



Let's discuss a PurchaseOrder example to highlight these scenarios. Let's say we have an existing customer who is submitting an order. They add some items and then checkout. All the data entered (such as which products, quantity, total cost) are stored as part of the case and Pega is the system of record for that information.

When the user goes to check out they are asked to supply a shipping address.



Pega System of Record = Transaction of case  
Some reference data



We can pull the current shipping address for our customer from a customer database. The customer database is the system of record for that bit of data.



Pega System of Record = Transaction of case  
Some reference data

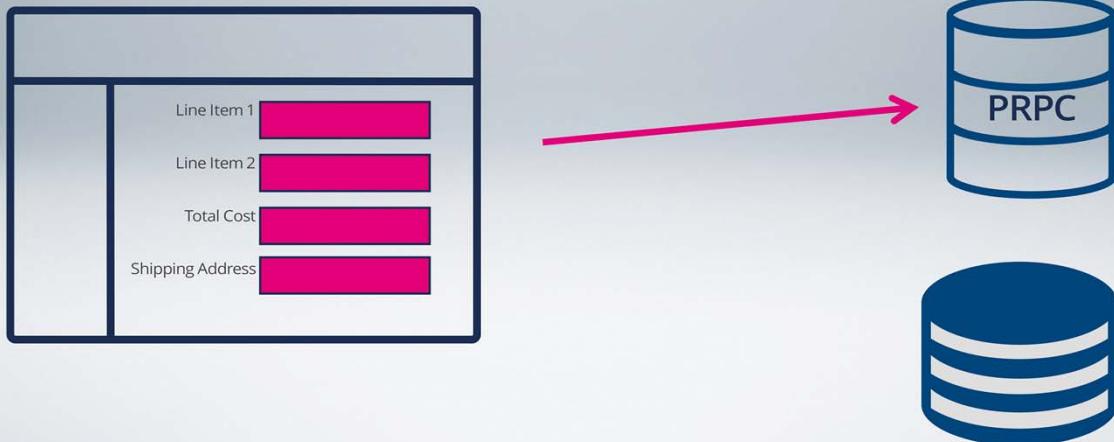
	Line Item 1 <input type="text"/>
	Line Item 2 <input type="text"/>
	Total Cost <input type="text"/>
	Shipping Address <input style="background-color: pink;" type="text"/>



We could also give the user an option to update the address and save the new address. When they save the address we push the updated address to the customer database to maintain the data integrity of the address.



Pega System of Record = Transaction of case  
Some reference data



We will also store a copy of the shipping address used for that order with the rest of the case data in the PRPC database.

Pushing data back to a system of record is something that a senior or lead system architect is tasked to do, it involves some forethought and integration efforts. In this lesson, we'll focus on how to pull data from a system of record into an application.



ID	Type	Name
0021	dental	dental
0022	vision	basic vision
0023	dental	dental plus

### DATA TABLES STORE REFERENCE DATA

In Pega, data tables are used to manage reference data. A data table is a simple structure which is used to store reference data outside of the application.

Data tables are created in Designer Studio using the Data Table wizard. The values for a data table are maintained manually in Designer Studio. We strongly encourage you to use data tables for small sets of data.



ID	Type	Name
0021	dental	dental
0022	vision	basic vision
0023	dental	dental plus

**Require a unique key**

### DATA TABLES STORE REFERENCE DATA

Data tables typically have a single field that represents a unique key.



ID	Type	Name
0021	dental	dental
0022	vision	basic vision
0023	dental	dental plus

**Require a unique key**

**Require single value properties**

### DATA TABLES STORE REFERENCE DATA

Other fields must reference single valued properties.



ID	Type	Name
0021	dental	dental
0022	vision	basic vision
0023	dental	dental plus

**Require a unique key**

**Require single value properties**

**Store non-volatile data**

### DATA TABLES STORE REFERENCE DATA

So when should we use a data table? We want to use a data table when we have non volatile data that will only change periodically.

## Demo



## Creating Data Tables

Reference data can be managed by creating a data table. For this demo we want to create a data table that tracks which currency codes are used by our application. We need a data table to track currency codes and what currency the code is for. To create a data table in Designer Studio go to the Designer Studio menu and select Data Model > Data Tables > Data Tables.

The Data Tables landing page displays all the data tables that exist within PRPC. Let's create a new data table.

Now let's fill out the data table form. First we give the data table a name and a description.

Next we define which class it inherits from, all data tables should inherit from Data- or a class that inherits from Data-. For our data table let's use Data-.

From this page we can also add the columns we want in our data table. Remember that every data table needs a unique identifier, we will add an ID field for that. We will also reuse the label field for the name of the currency.

Next we will add a column to keep track of the currency code.

Once configured click Generate to build the data table. When complete we'll see a summary of the data table we just created.

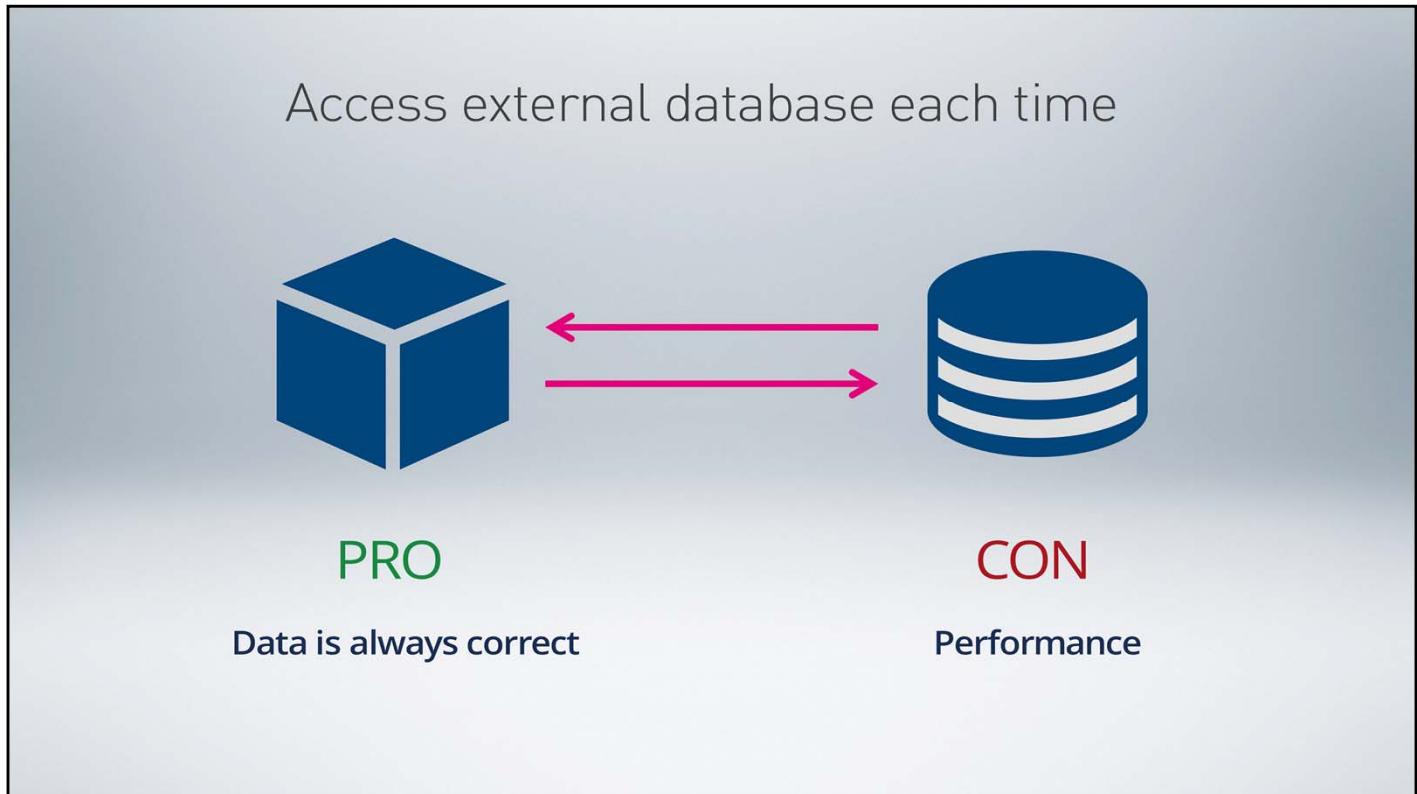
Now that our data table has been created lets add data to it. To add data we click the Gear icon.

To add a row of data click the plus icon and then enter a value for each of the fields.

Then click the save icon to add the row.

Let's add a few more rows of data, this time we won't save until we've added all the rows we want.

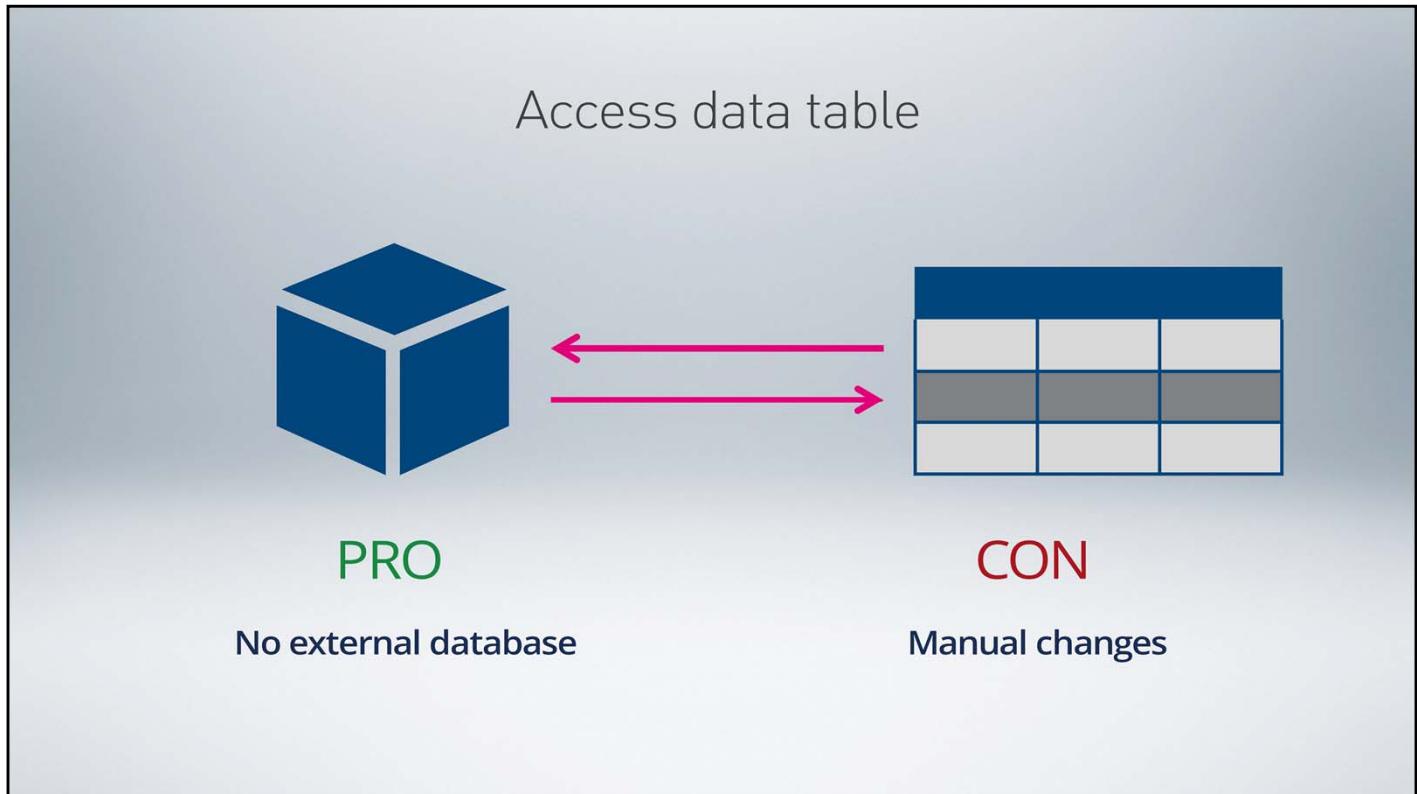
Now we successfully created a data table and populated it with data.



One situation that always occurs when building an application is that the application needs to access reference data. One way to tackle this problem, is for the application to access the data each time it is needed.

An advantage to this approach is that we can guarantee that the reference data is always correct.

The disadvantage of this approach is that the more this data is used, the more trips to the external resource is needed. This could really affect performance and would do so unnecessarily if the reference data we are accessing doesn't change very much.

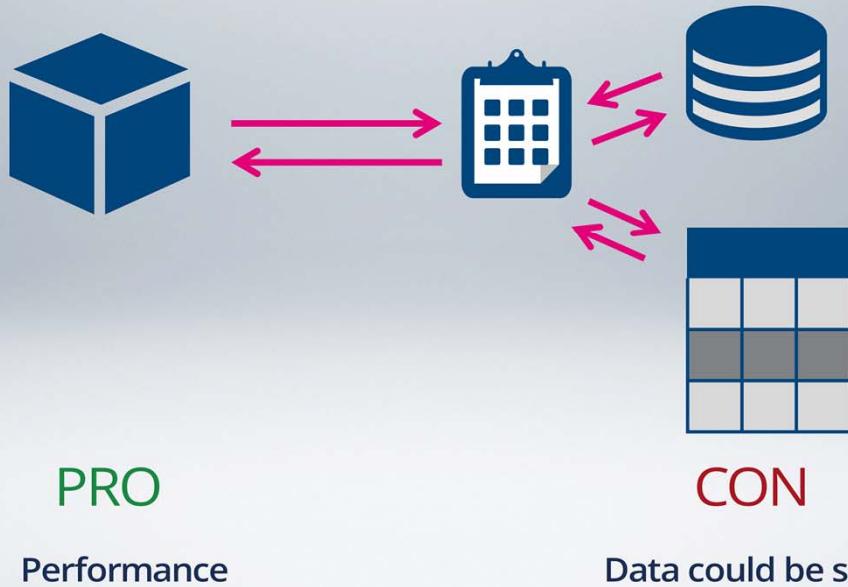


Our next option might be to use a data table. By putting our reference data in a data table we eliminate the need to go an external resource, as the data is stored in the PRPC database.

This helps the performance because PRPC optimizes accessing that data.

A disadvantage to this approach is that if the data needs to change it must be done manually. Additionally, while the performance will be better by using a data table, we are still making a trip to a database. Is there another way?

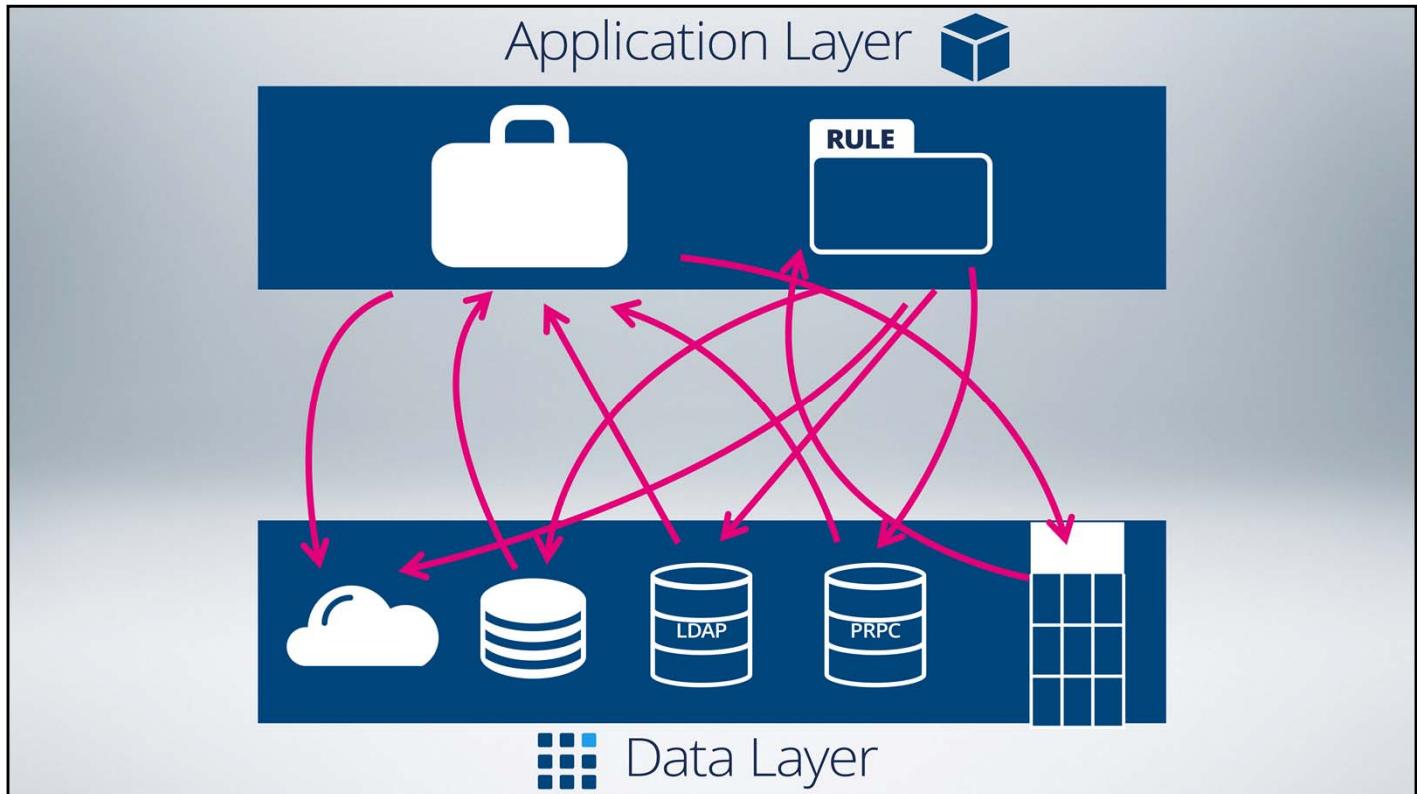
## Use a data page



To improve performance in using reference data let's discuss how to cache the data. In order to cache the data we need to use a data page. A data page loads the data into memory and stores it on the clipboard, it is then accessible to our application.

By storing the information in a data page we gain an increase in performance since the data is in memory.

The downside of using a cache is always the freshness of the data, it is possible to have stale data in the cache. How often we refresh the data in the data page is configurable and we will discuss refresh strategies for data pages shortly.

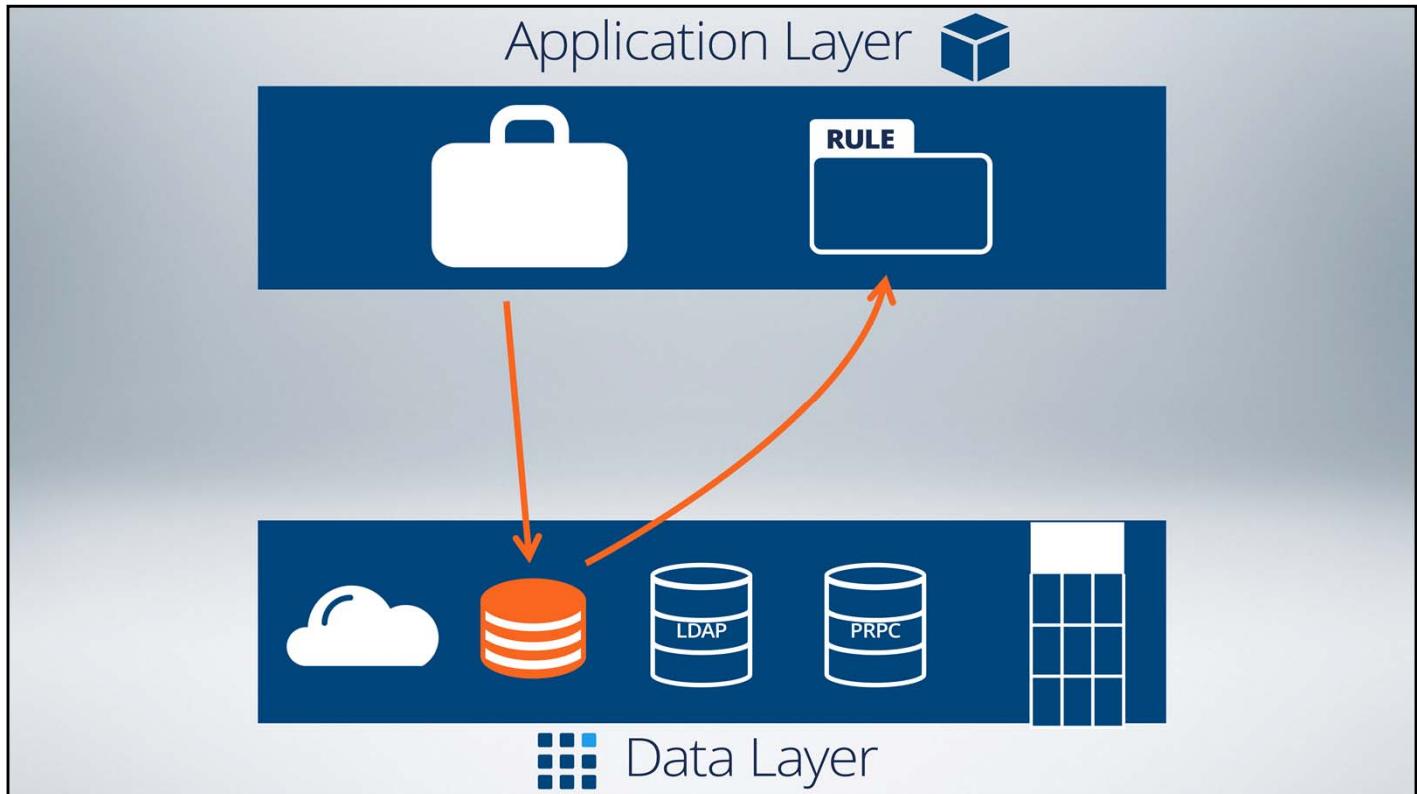


In addition to data pages improving performance of an application by caching data, they are also a tool used to increase maintainability of an application. There are really two layers in any application, a data layer and an application layer.

We've discussed the data model and that we use data classes to represent the data model, this is what makes up the data layer.

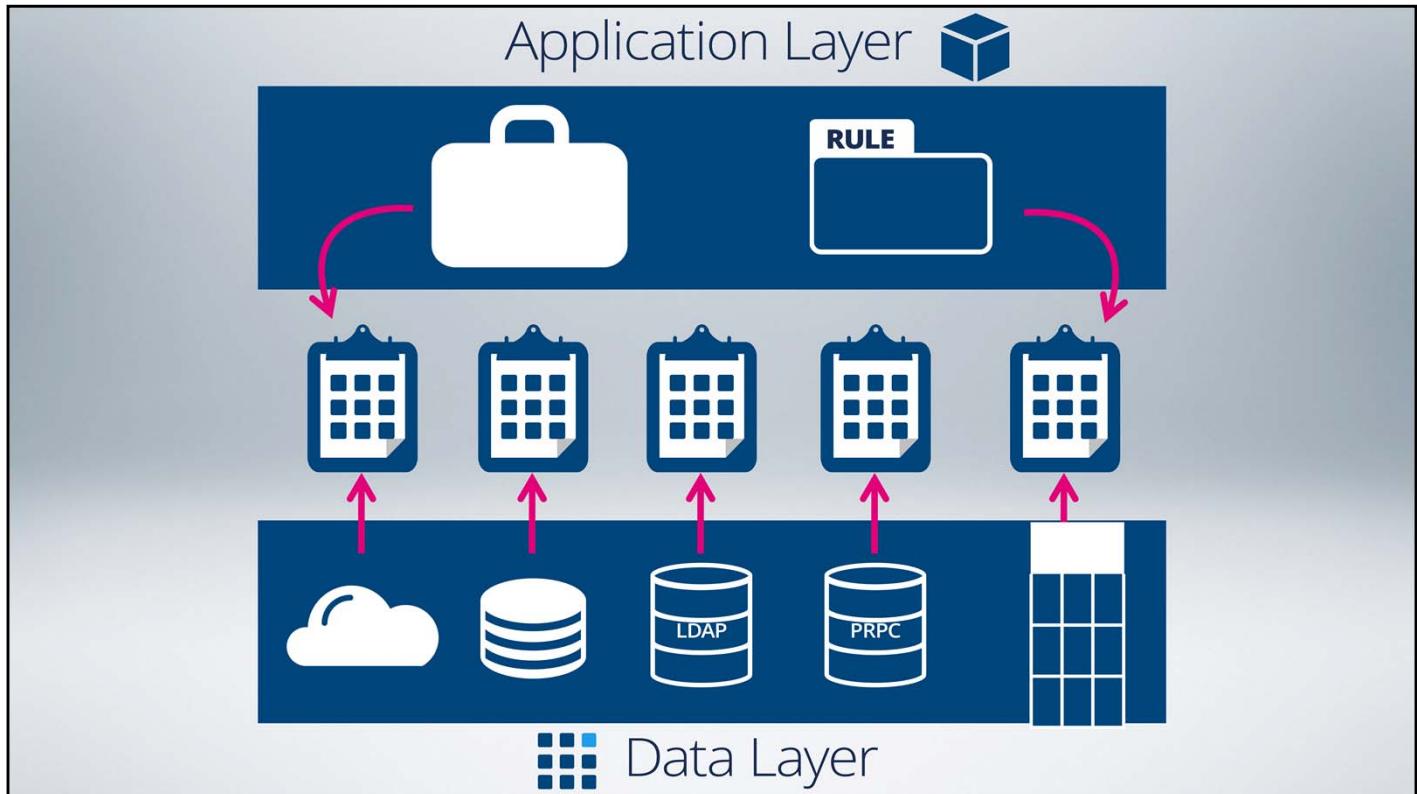
The Application layer consists of the different rules that need access to that data.

It is possible for the rules to access the data directly



but what would happen if we needed to change one of the data sources?

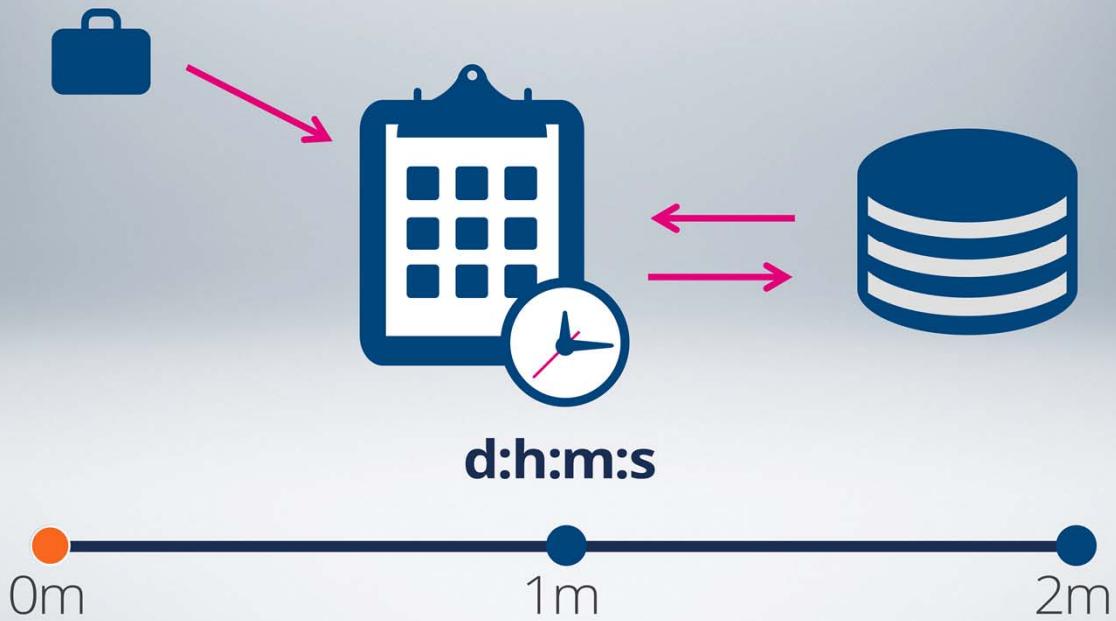
We would then have to update all the rules that used that data class and update it to the new one.  
This does not make our application very maintainable.



We can use data pages as a layer in between the data layer and application layer to make our applications more maintainable and by caching the data in memory we can increase the performance of our application.

There are three aspects to Data Pages that we need to discuss 1) how often is the data refreshed, 2) how long does the data page stays in memory and 3) how to get data into the data page.

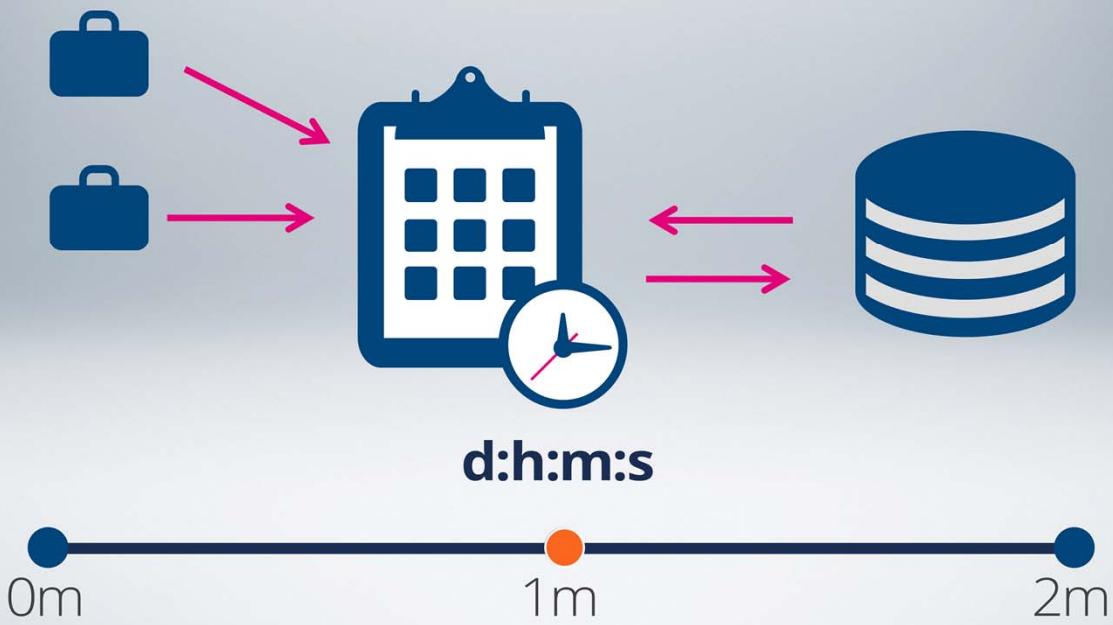
## Reload if Older Than



Data pages use refresh strategies to make sure that they do not have stale data. The first refresh option is the Reload if Older Than option.

With this option we define an expiration time interval in days, hours, minutes, and seconds.

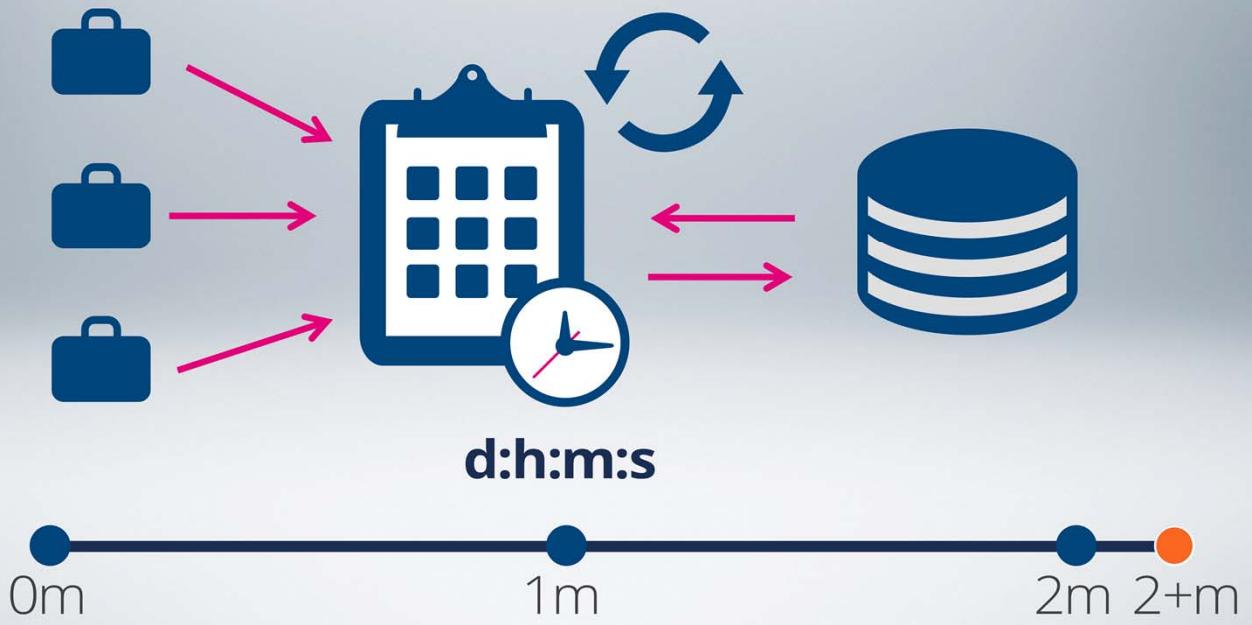
## Reload if Older Than



If we have supplied a Reload if Older Than value, the data page becomes stale when the time interval has elapsed since the page was first loaded. The page is never refreshed more than once per user interaction. Let's say we have a countdown of 2 minutes. First a case would need to access the page for an initial load.

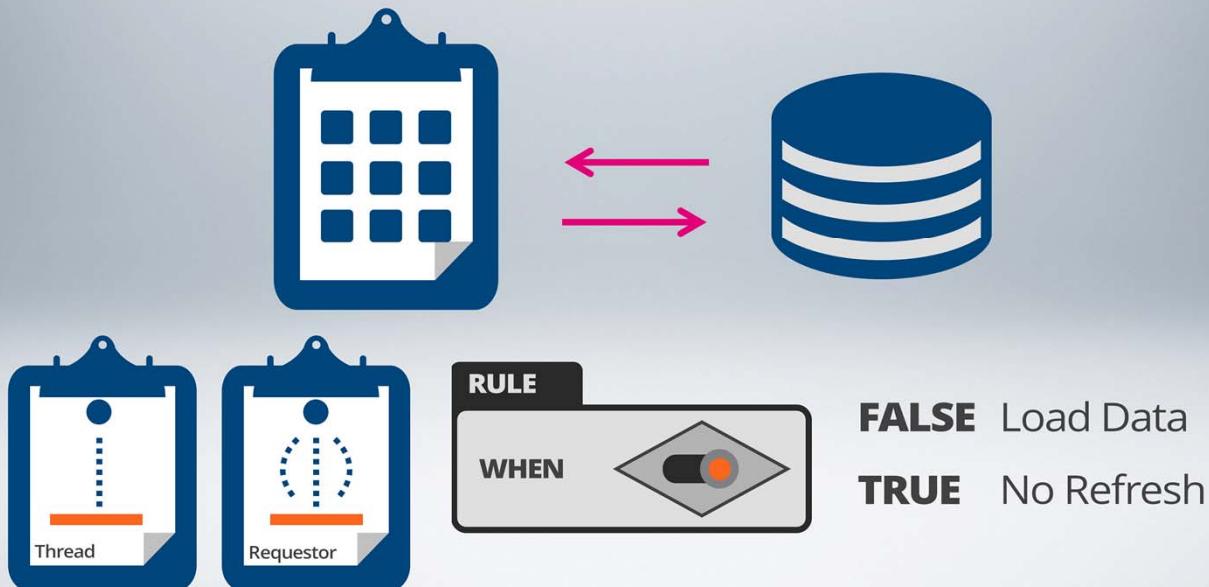
After one minute, another case accesses the data page. Since it is less than two minutes the data is not refreshed and the existing copy is returned.

## Reload if Older Than



After 3 minutes, a third case accesses the data page. Because more than 2 minutes have elapsed, the data page is refreshed.

## Do Not Reload When

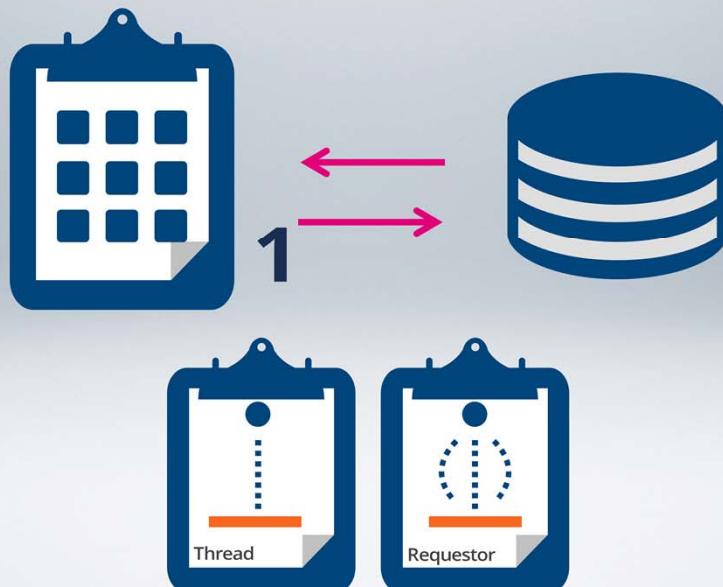


The next refresh strategy is to use the Do Not Reload When option. This strategy has us identify a When rule to be evaluated when a requestor accesses a page with a Page Scope of Thread or Requestor.

If the When rule evaluates to false, the page contents are refreshed. The page is never refreshed more than once per user interaction.

To find this rule at runtime, rule resolution uses the class in the Page Class field as the Applies To class, and the RuleSet list of the requestor. This field appears only when the Page Scope is set to Thread or Requestor.

## Reload Once per Interaction



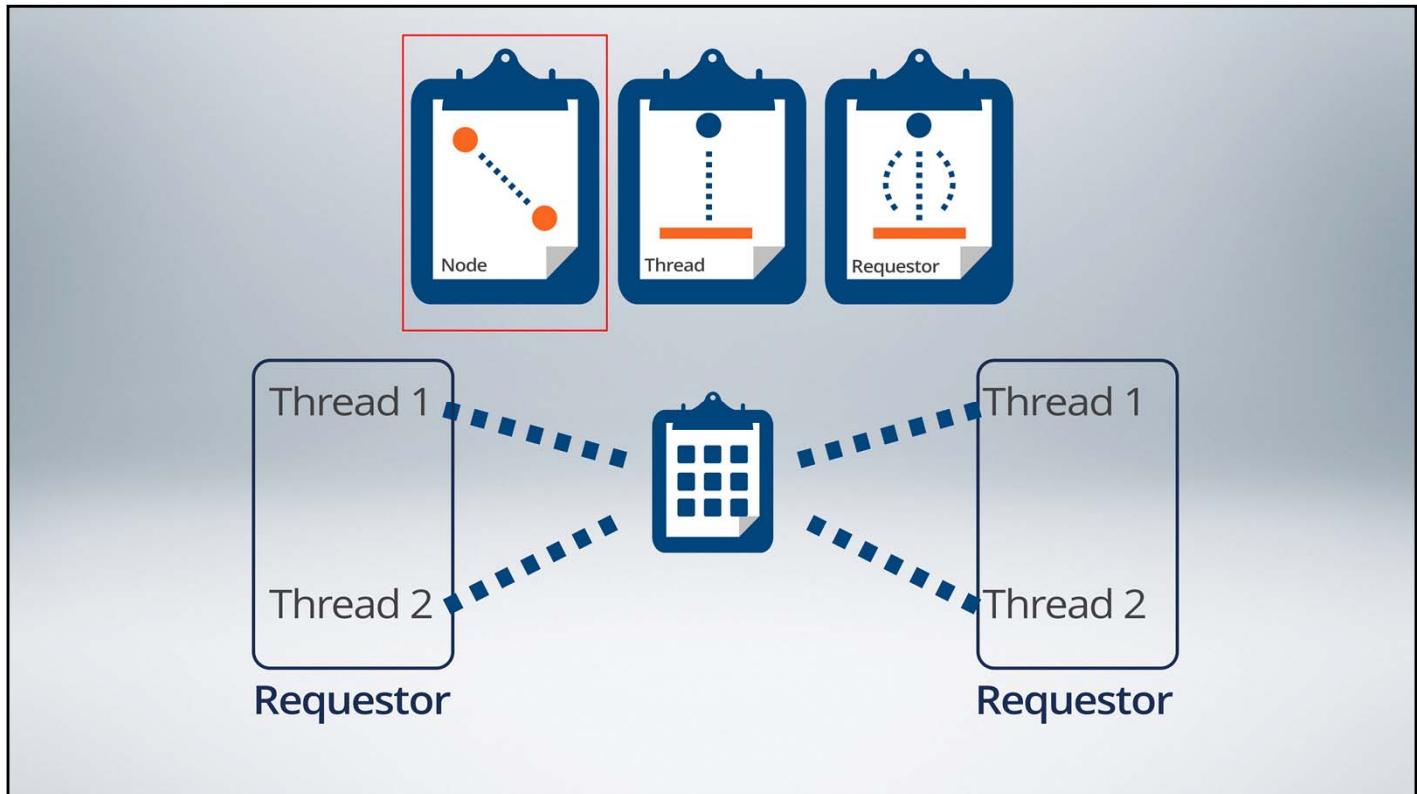
The third refresh strategy is the Reload Once per Interaction strategy. If selected PRPC refreshes the Data Page exactly once per user interaction. This option is available only for rules with Page Scope set to Thread or Requestor. When checked, at runtime PRPC ignores any values in the Reload if Older Than and not Reload When When fields.

For Thread-scope pages determine the refresh strategy carefully, especially if your refresh operation is costly in terms of elapsed time or use of system resources. This involves a trade-off of possibly stale data versus additional processing. For example, refreshing upon each interaction may introduce avoidable extra processing if once-per-hour is good enough. But in a high-frequency access situation, refreshing once per minute may be less often (and so less costly) than once per interaction.



Three scopes for data page

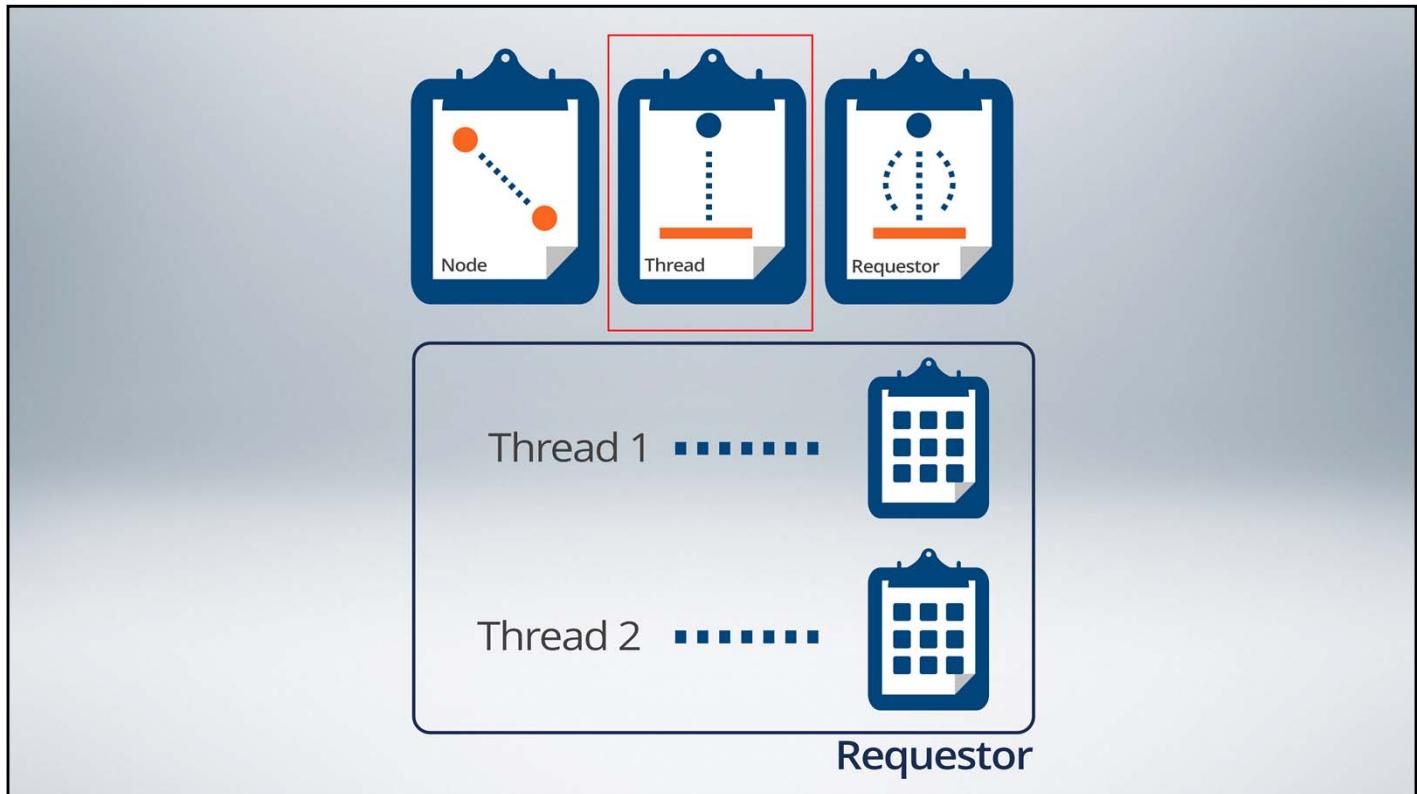
Let's discuss how long a data page can live in memory. Data pages can have one of three different scope values: Node, Thread or Requestor.



The first scope we will discuss is Node scope.

Using Node scope for your data page, places the data page on the Clipboard with the other node-level pages.

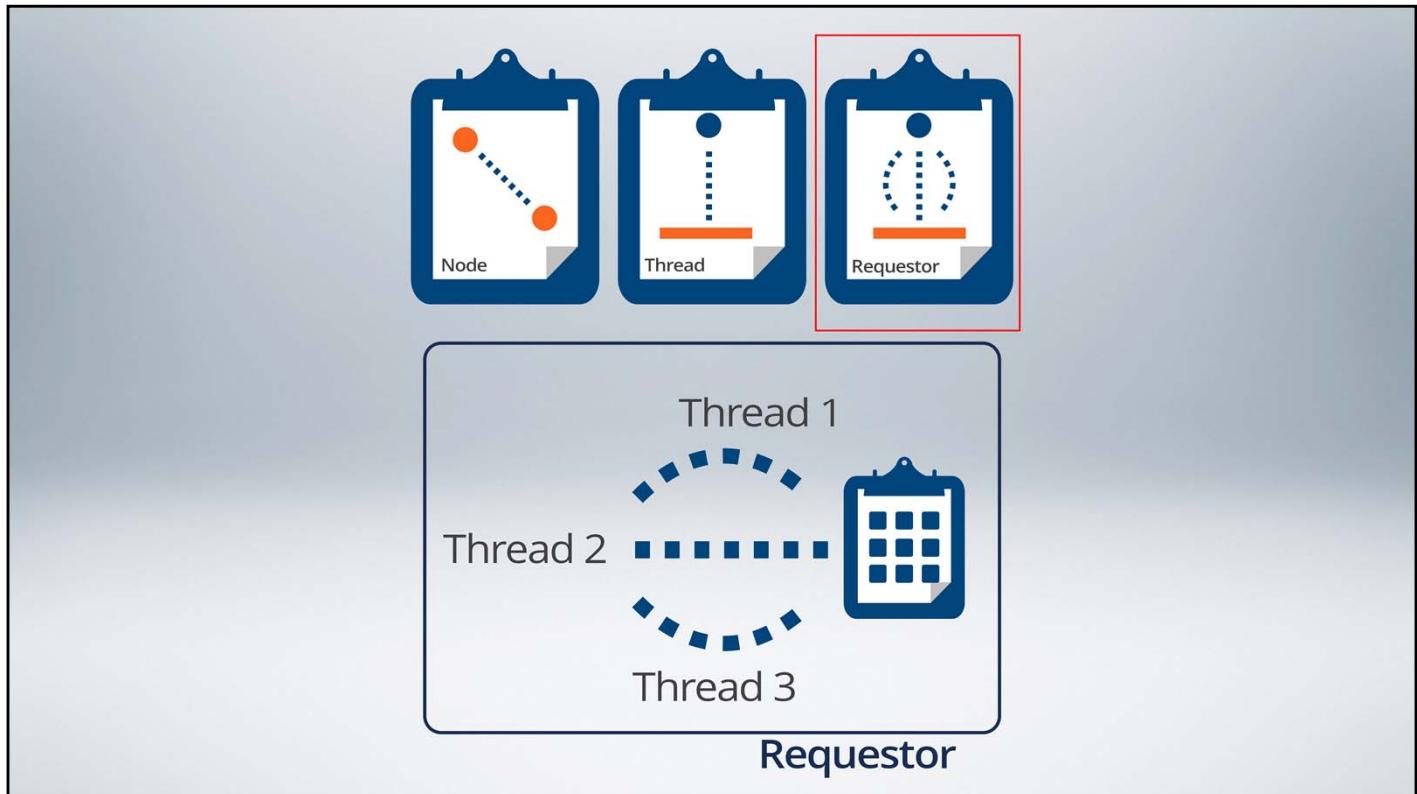
The data page is shared across all requestors of that page, for that node. Your data page loads on the initial request and for subsequent requests they access the same instance of that data page.



The next scope option for a data page is Thread.

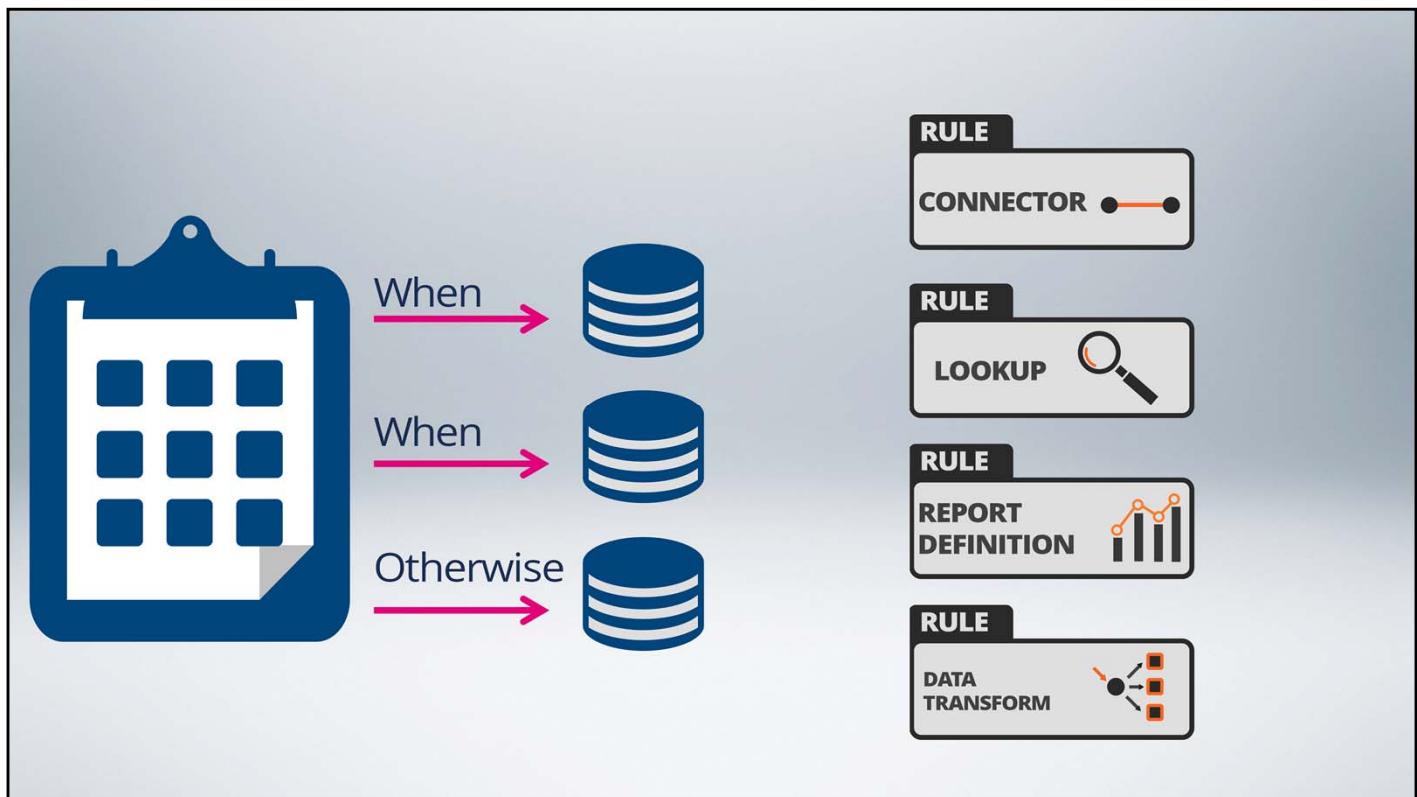
When you choose Thread the data page is loaded and maintained in each Thread that references this page name.

If a requestor has multiple Threads, then each Thread has its own instance of this page. This scope type provides clipboard pages that are created when needed and are dependent upon the current work item context that exists. For example, a Product rates data page might be retrieved from an external Products database. For each account type in memory (item), the rates might be different. So on a screen for one client, with three products, we'd have three data pages to display the available rates for those products.



The last scope option is Requestor scope.

Requestor scope means that your Data Page is shared across all threads for a given Requestor. For example, we could retrieve the approval limits for an operator from an external data source. The limits apply for the duration of the requestor's session, as they work on making underwriting decisions for various accounts.



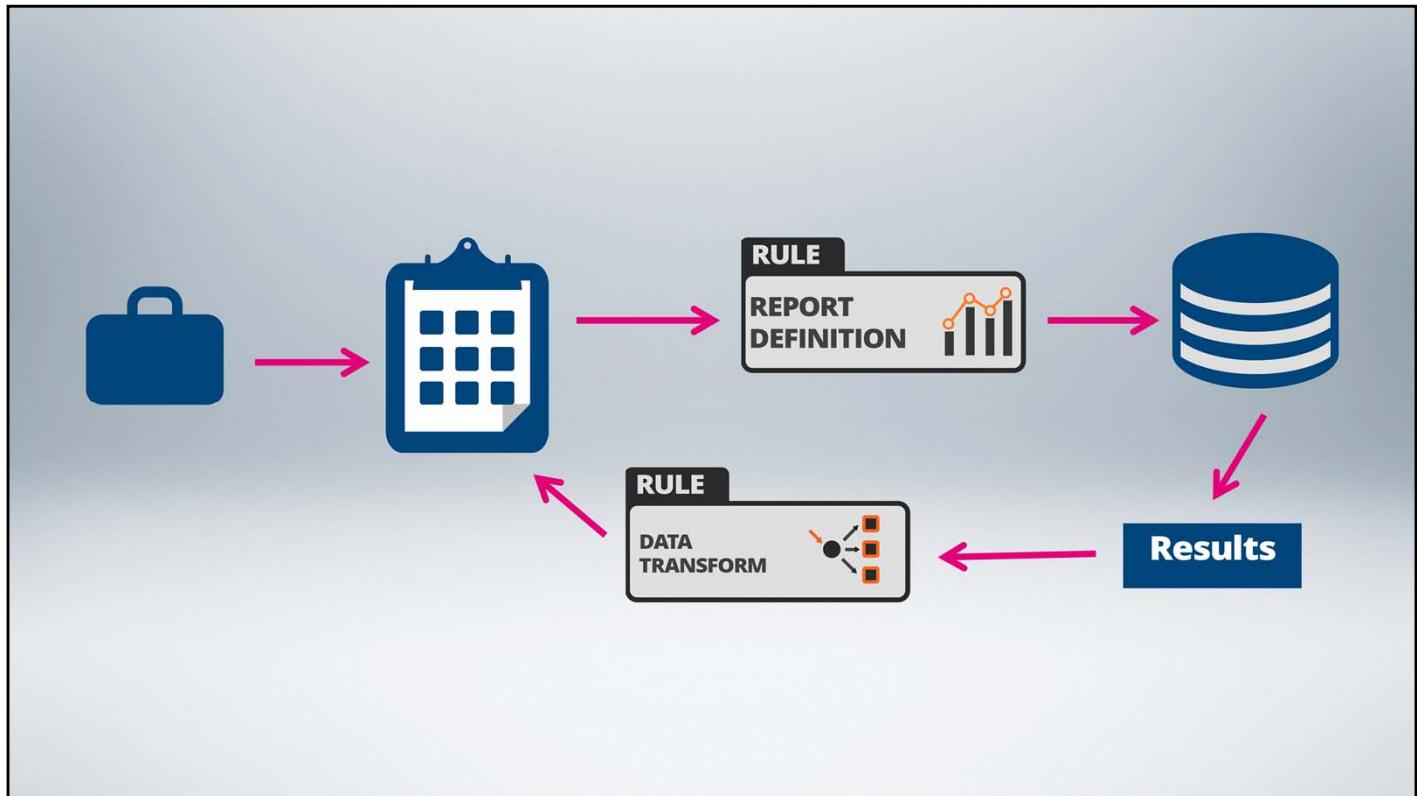
Data Pages determine what data to load based on its Data Sources configuration. Data pages can contain multiple data sources which can be passed the parameters that were passed to the data page.

When there is more than one data source, a when condition appears next to all but the last, which serves as the “otherwise” condition. The when rules can also be passed parameters.

This allows different data sources to be chosen based on the parameters that were passed to the data page and eliminates the need to do source selection within a single load activity or data transform. This also empowers application developers to use the same data page definition in different contexts, controlling source selection as necessary.

There are a variety of ways that a Data Page can source data including:

- Connectors which allow us to interact with external systems
- Lookups to get data from a data table
- Report Definitions which allow us to define a set of data to examine
- Data Transforms which allow us to copy, map, or modify data

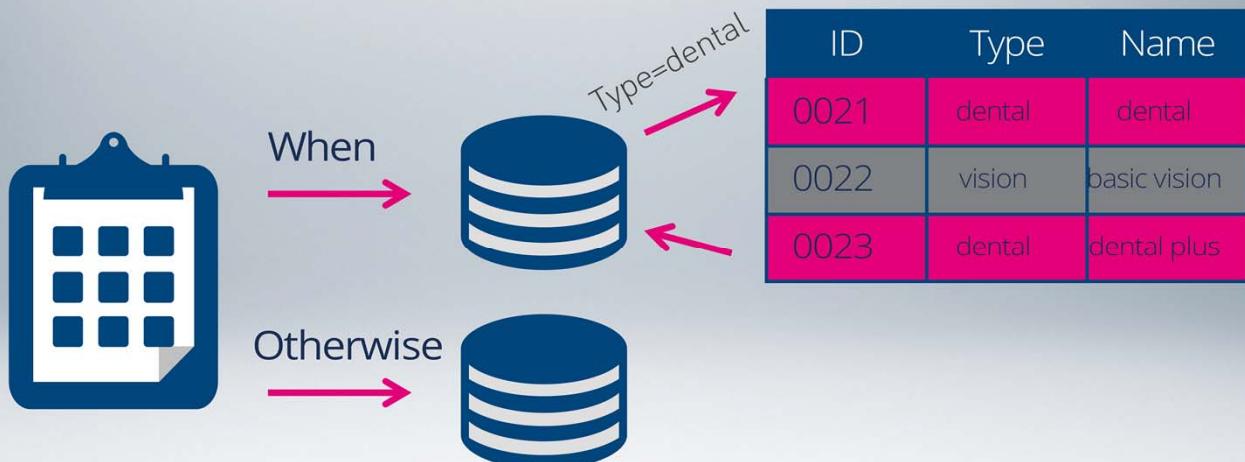


Let's examine specifically how a page is loaded with data. A data page is not loaded until it has been requested.

When it's requested, once the appropriate source is determined, that source is executed. In the example above a report definition was used to access a database.

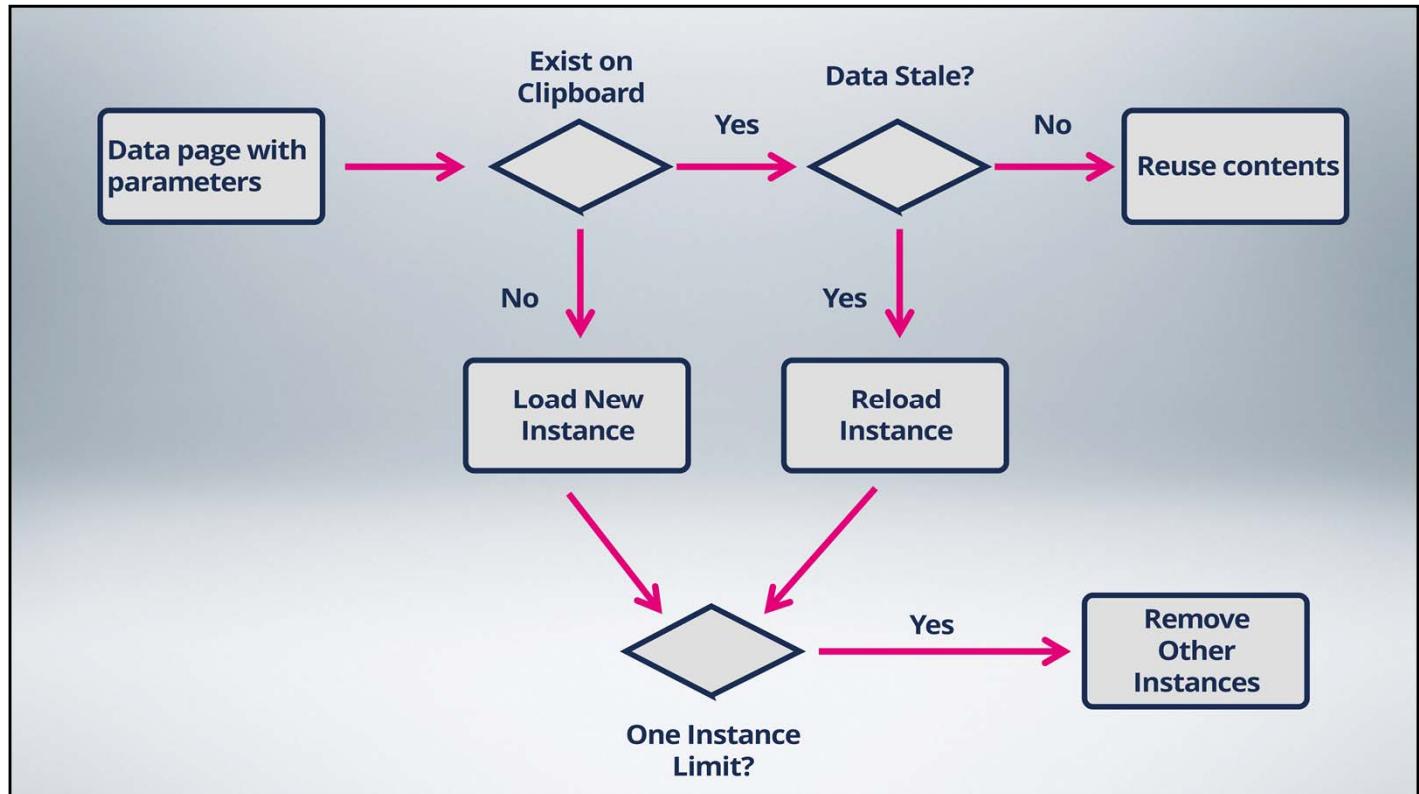
Once the results are returned there is the option to run a data transform to set the values of the data page with the returned results value. If the data page and report definition are from the same data class the data transform step is not needed.

## Using parameters with a data page

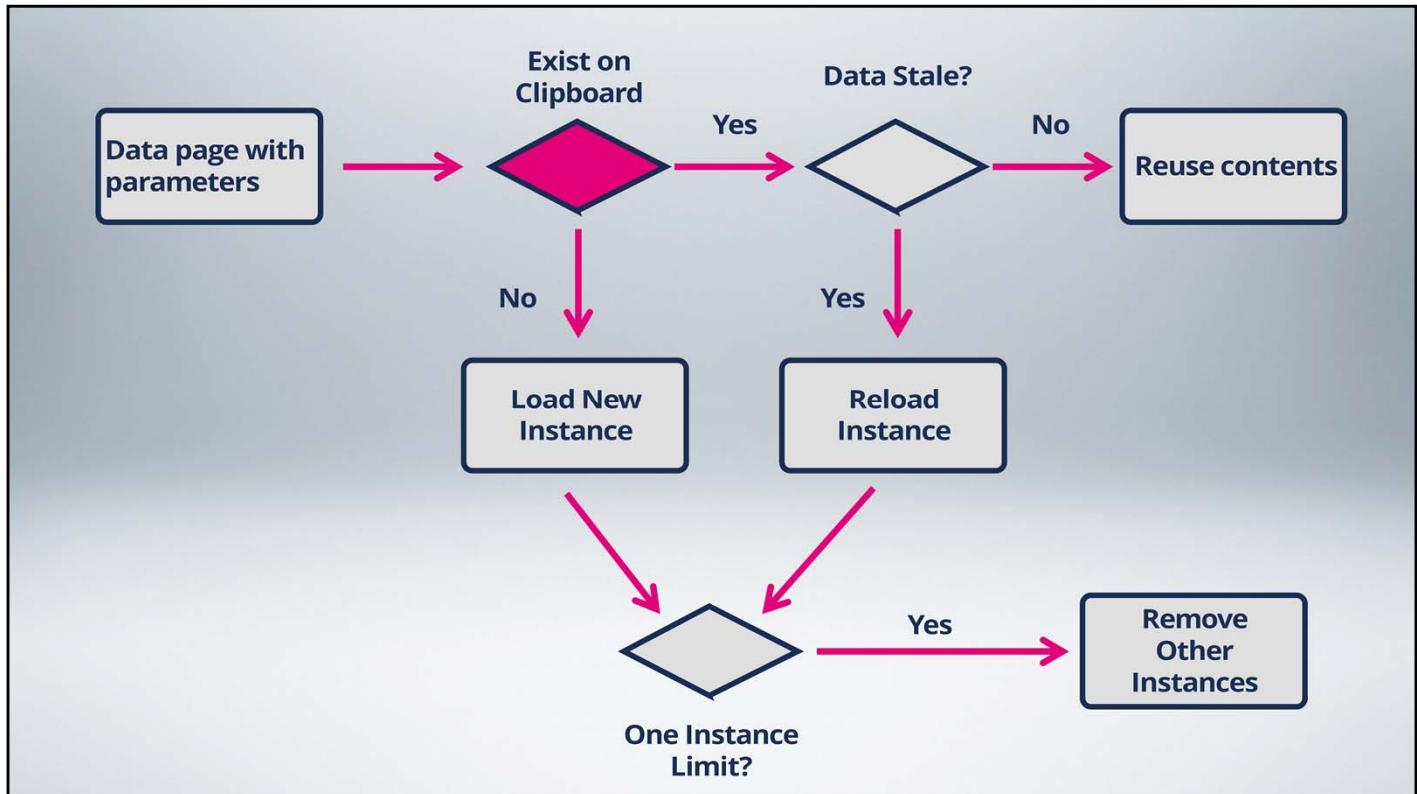


When working with reference data there are times where only some of the potential data is needed. For instance if there is a table that lists medical, vision and dental plans when a user is on the dental selection page we would only want to give them choices that are actually dental plans.

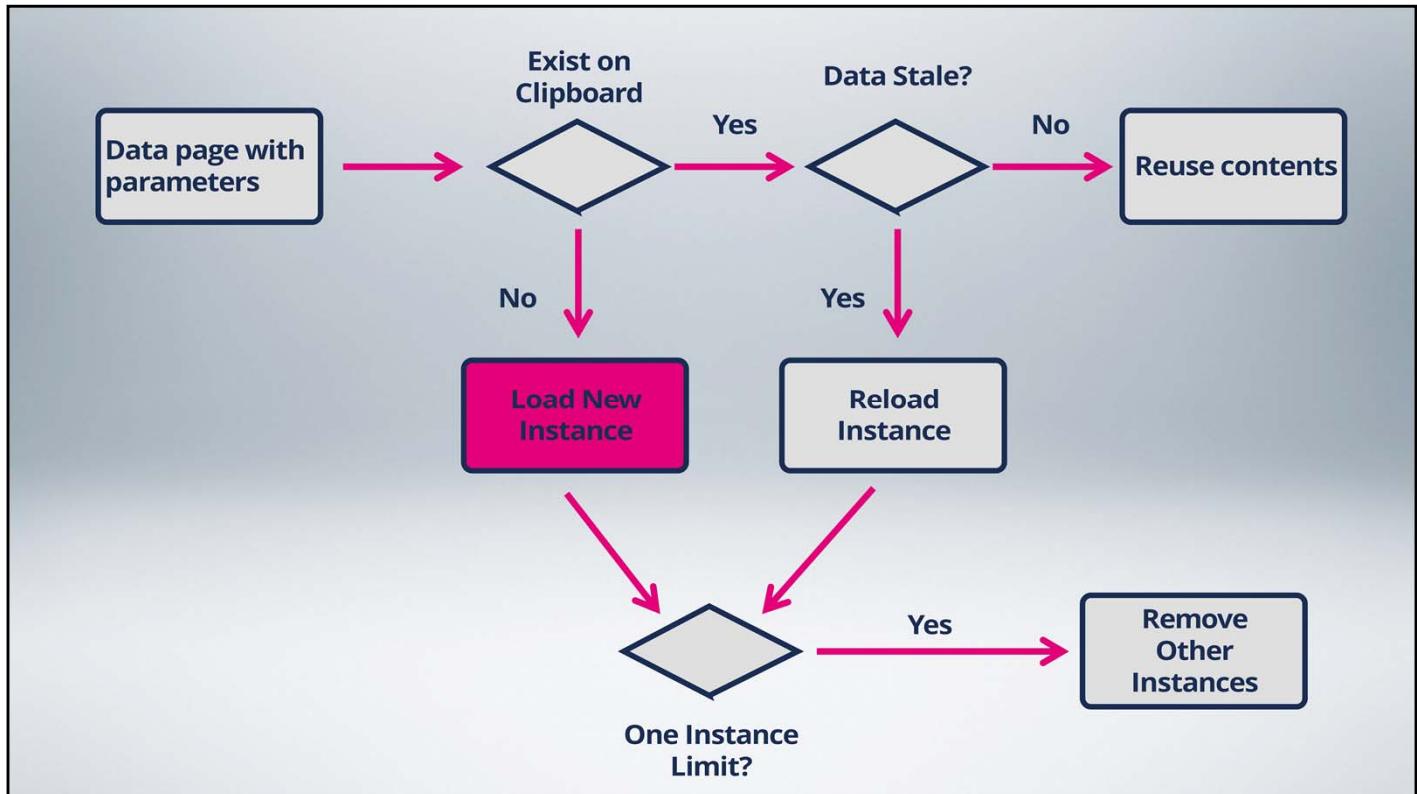
We can actually use data pages to cache this operation by passing the data page a parameter. The data page can then pass this parameter to a report definition that can use it to select the proper data. What will be returned to the application is a list of data pages, one for each of the dental plan options.



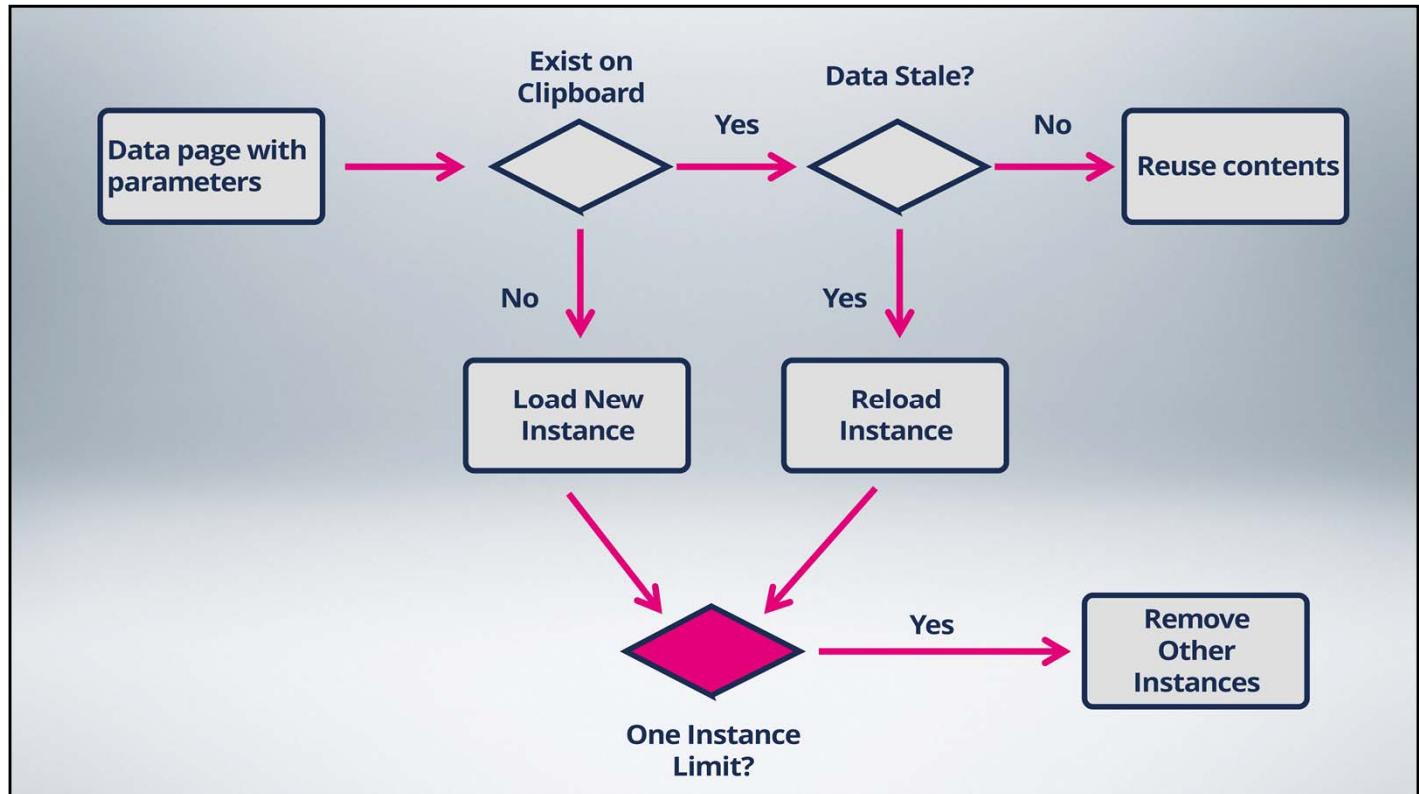
Let's discuss how data pages that use parameters are loaded.



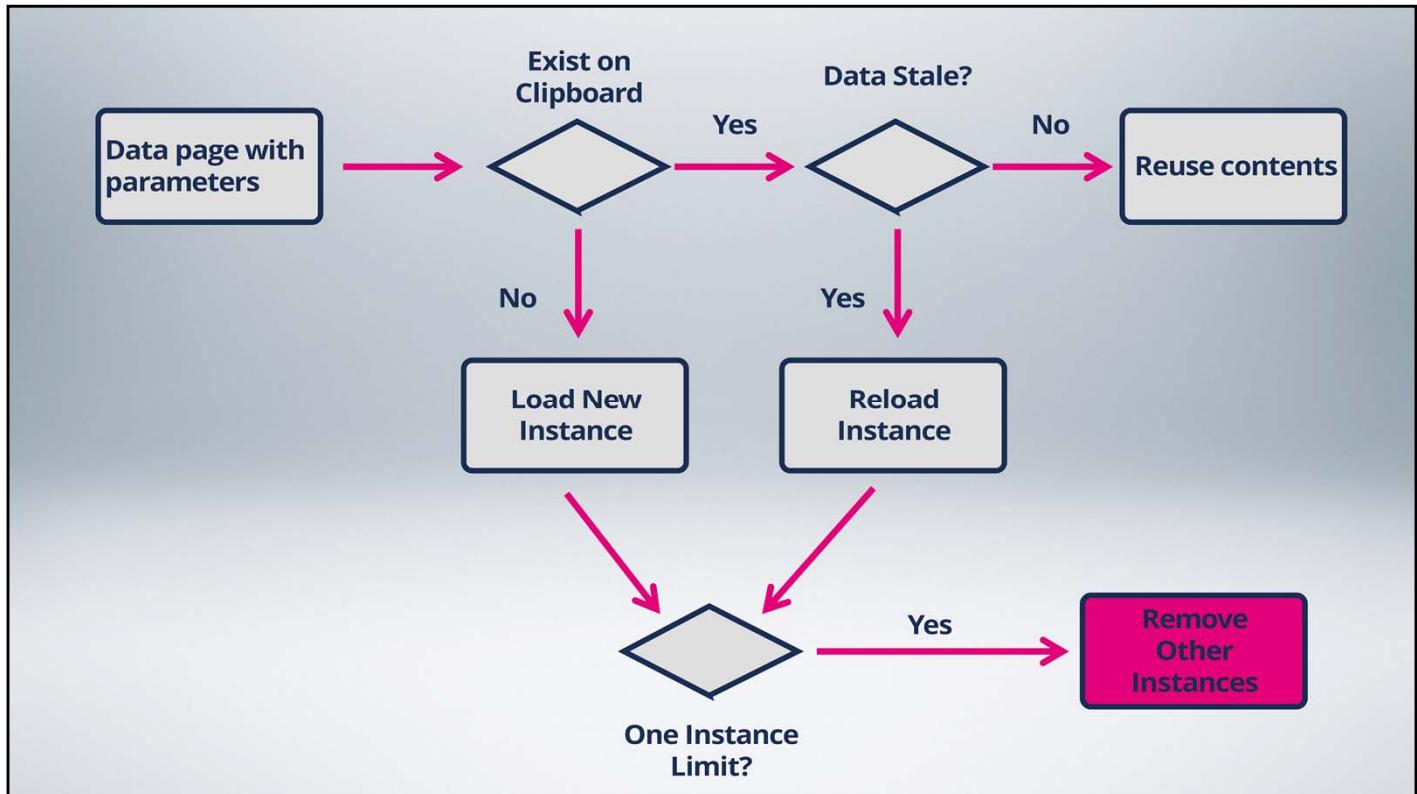
When a request is made the first thing that is checked is whether or not the data page with that parameter exists on the clipboard.



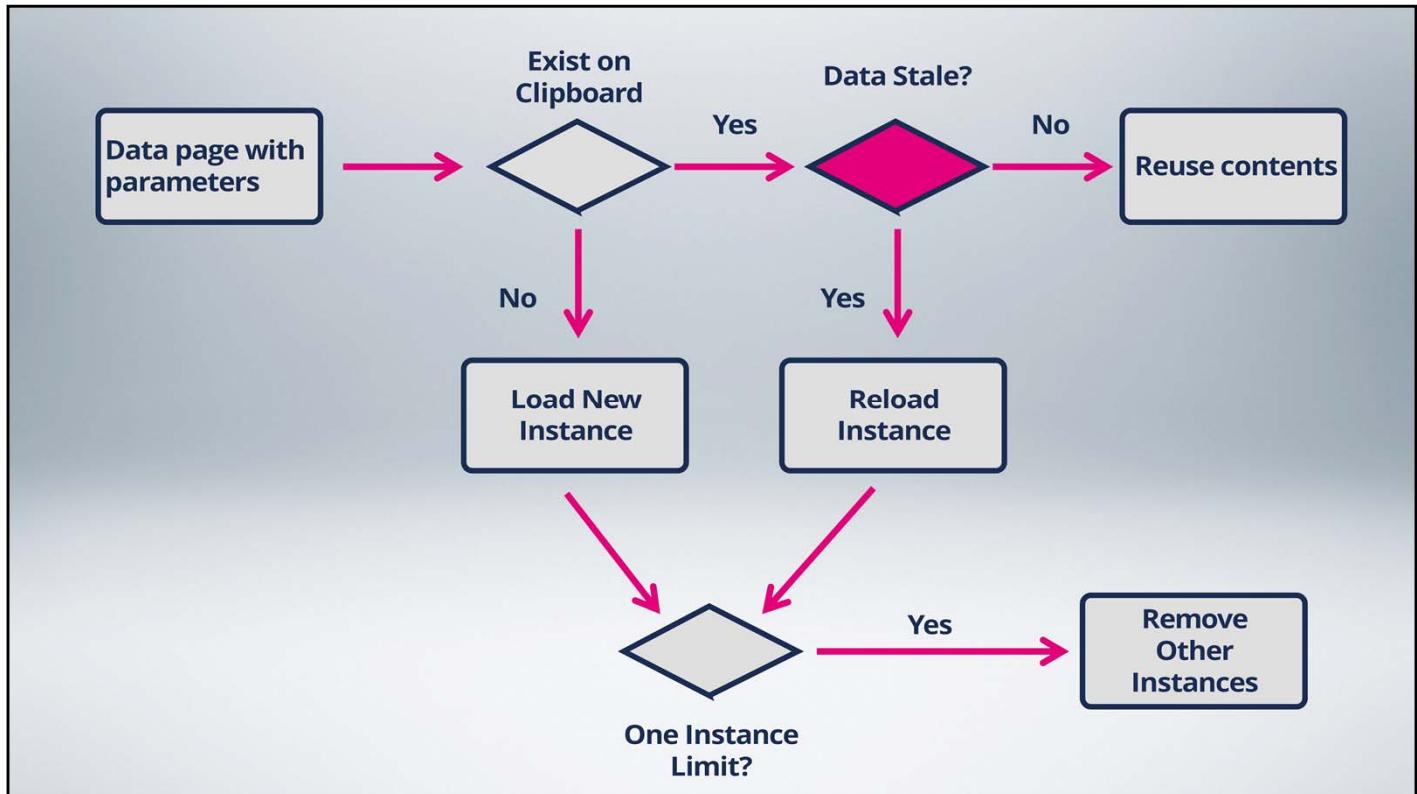
If it doesn't, a new instance is created.



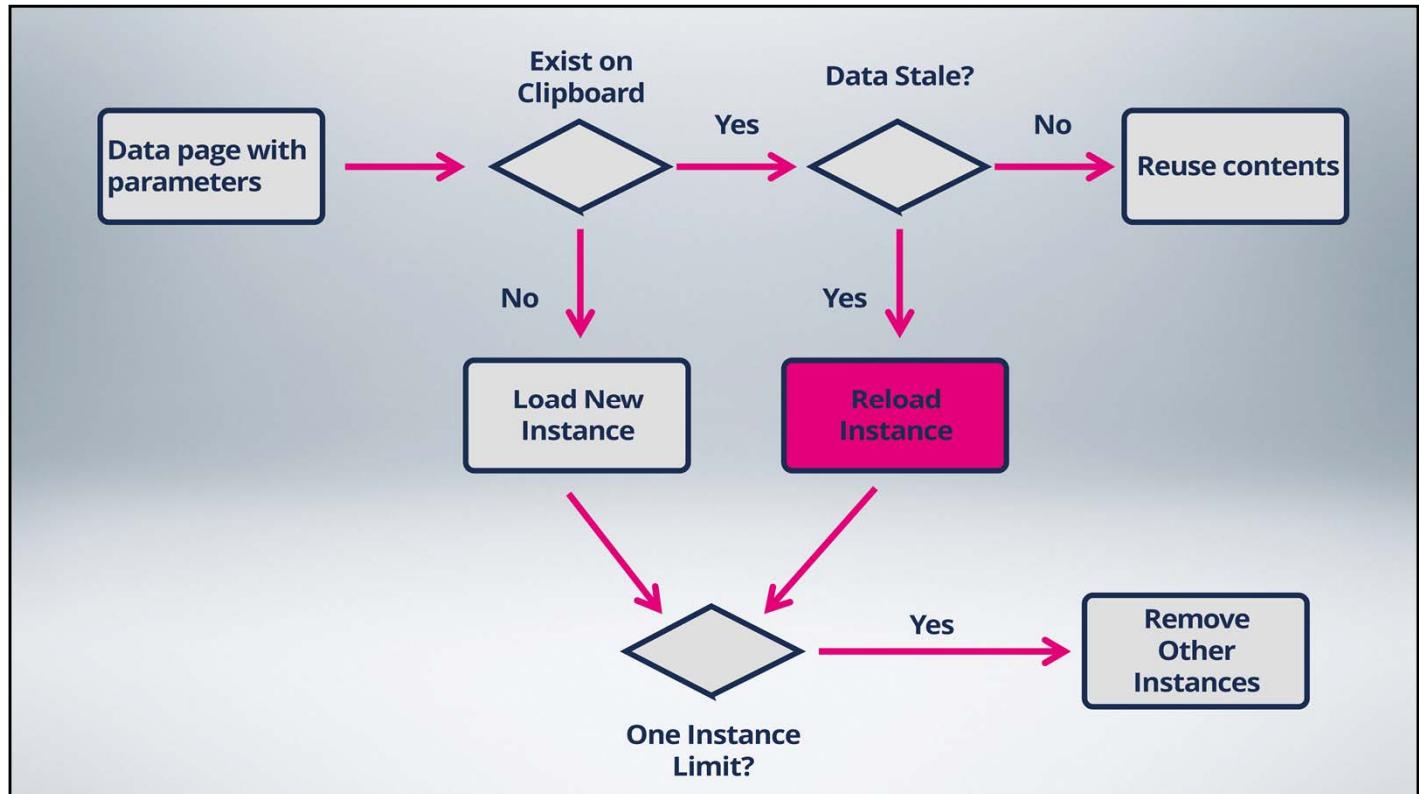
Data pages can have a one instance limit for its pages on the clipboard,



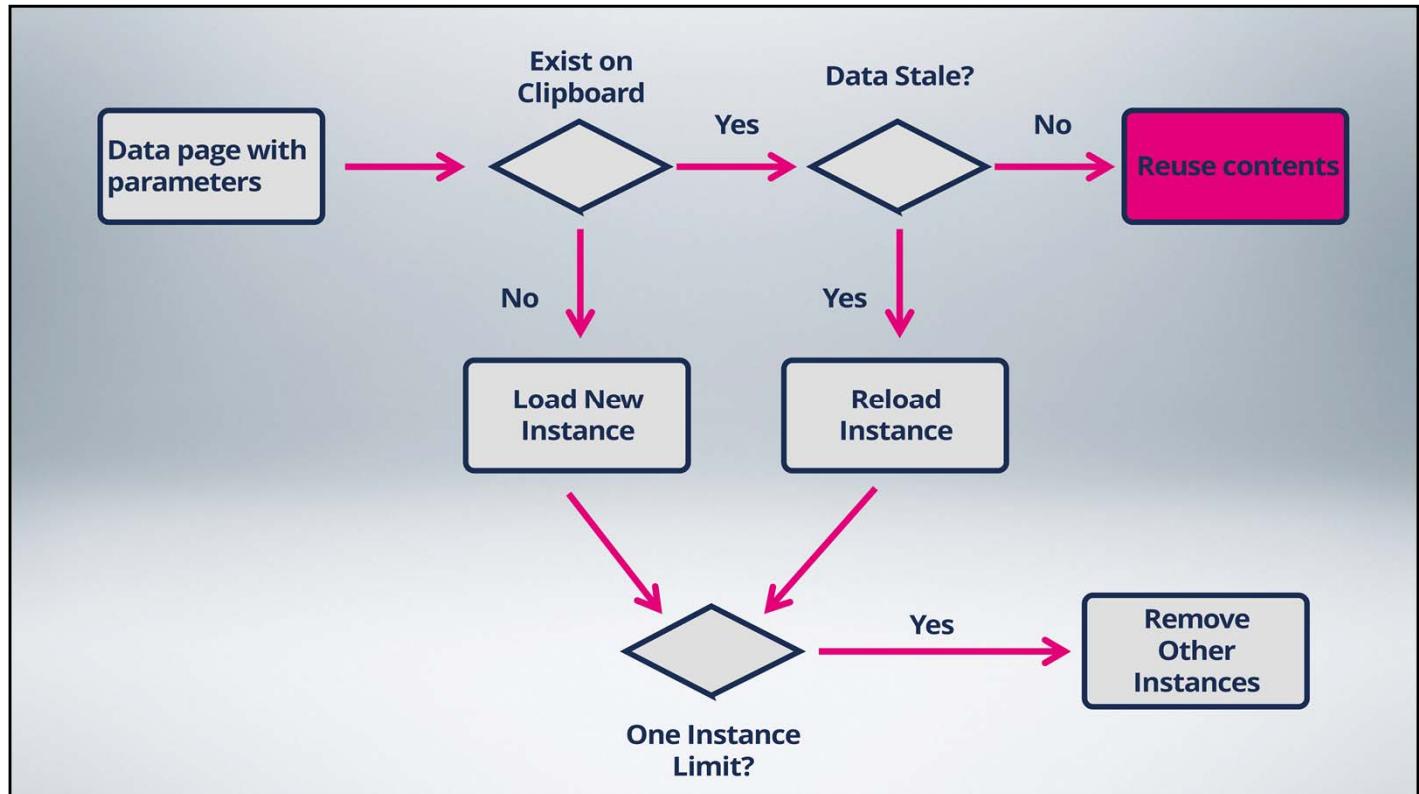
this is checked and if the limit is one, any other instances are removed before the new page is added.



If the initial page did exist on the clipboard the first check is to use the refresh strategy to determine if the data on the data page is stale.



If it is stale the data is reloaded.



If the data is not stale than whatever is currently on the clipboard is used.

## Demo



## Creating Data Pages

Let's discuss how to create a data page in Designer Studio. For this example we want to create a data page that takes a vendor ID and retrieves all the information about that vendor. We create a new data page from within the Data Explorer.

Next we give the data page a short description which will be used as the name. We also provide a RuleSet and Version to which the data page will belong.

Now we get to pick if we want a single page or a list of pages returned. If we expect multiple results to be returned we should choose List as the type.

Next we define the type that this data page uses. For our example that would be the Vendor data class.

Next we define if the page is read only or editable.

Then we can choose the scope of the page.

Next we need to add a data source that loads the data into the data page. For our example there is a data table that contains the vendor information that we need. To access the data table we use the Lookup option as the source.

In the Class Name field we define which data table we want to use.

When loading from a data table we often need a data transform to manipulate the response. One data transform has been provided.

Next we need to add the parameter since we only want some of the data.

The parameter we need is the VendorID.

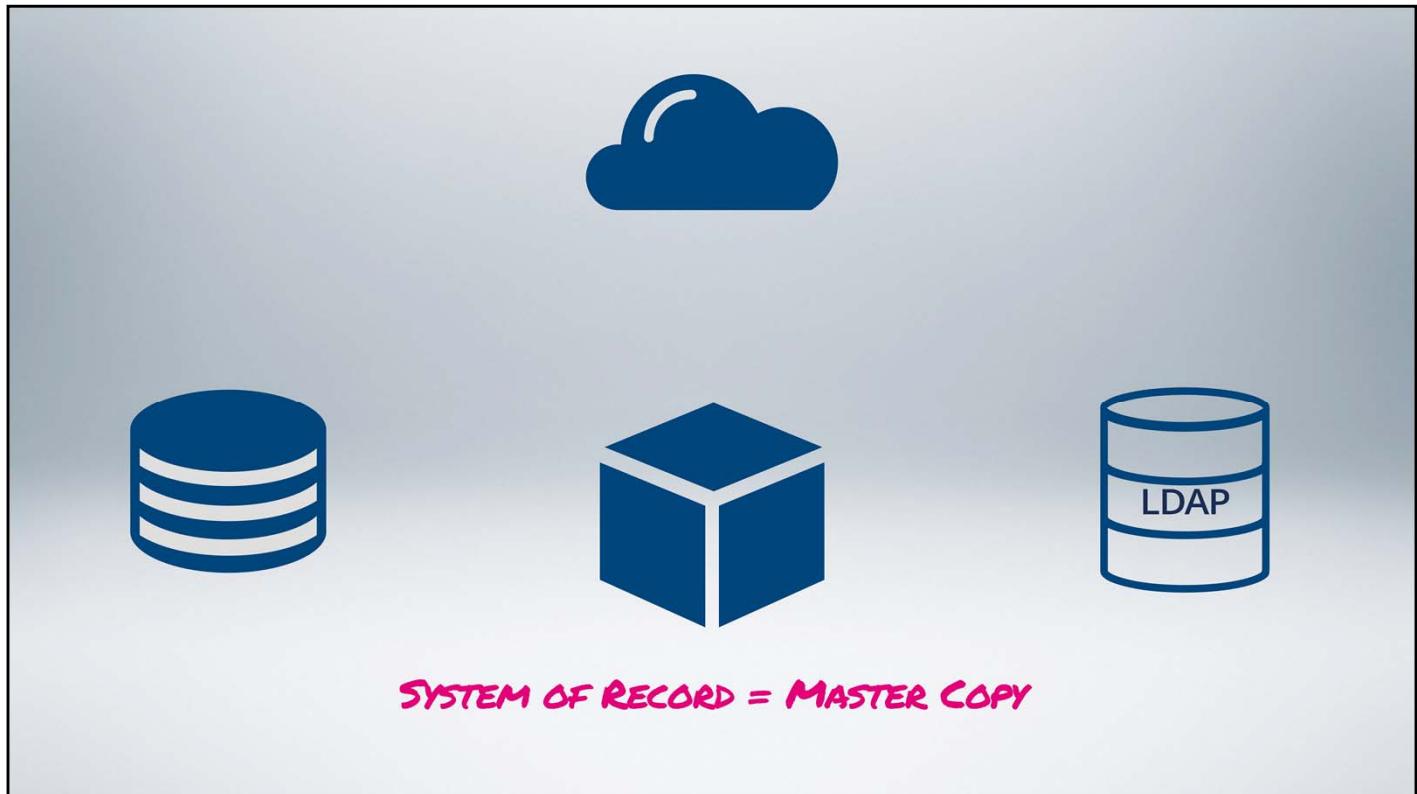
We also discussed setting a refresh strategy. To configure a refresh strategy click the Load Management tab.

For our example, let's use time as our refresh strategy and set the days to seven.

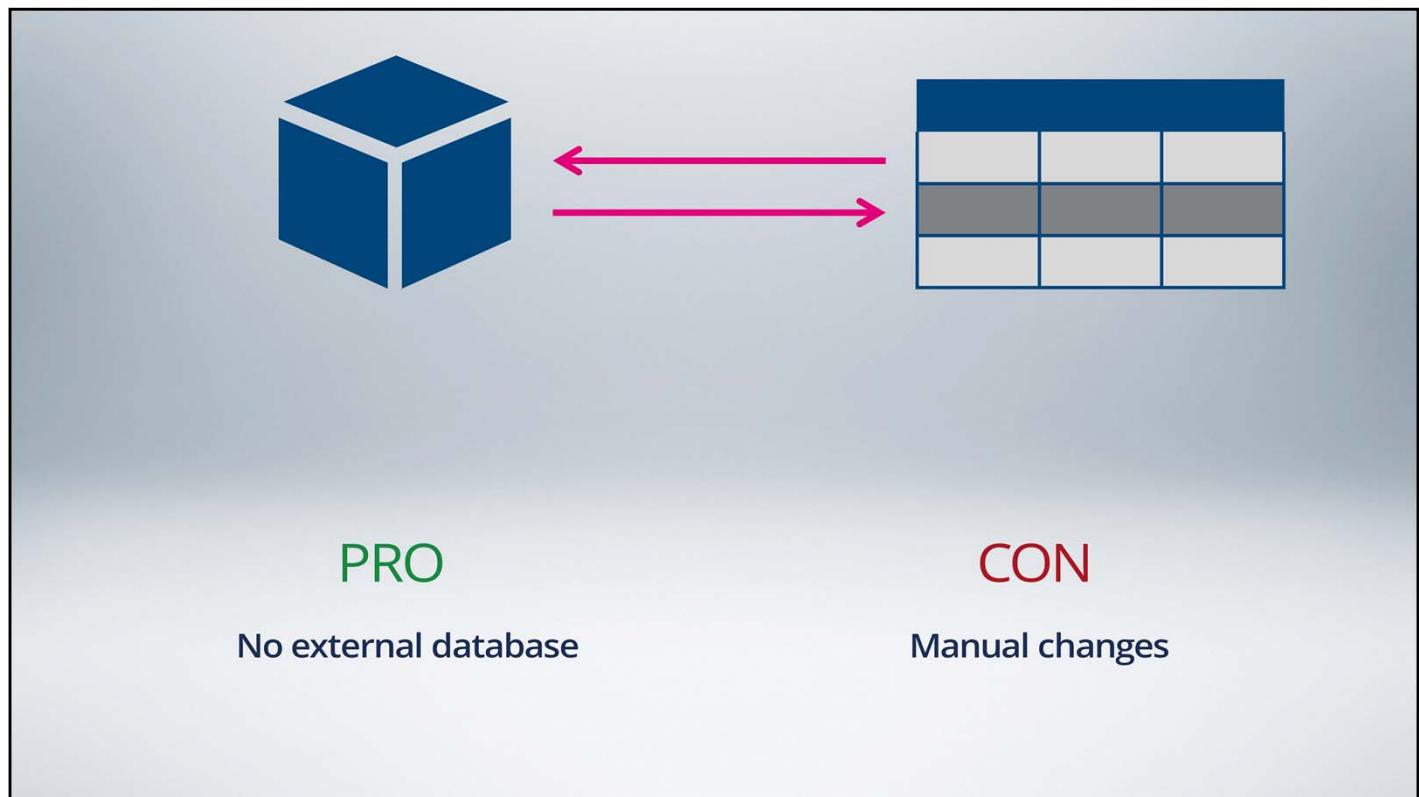
Finally we save our data page.

We can now see our data page in the Data Explorer. We need to refresh the Data Explorer first.

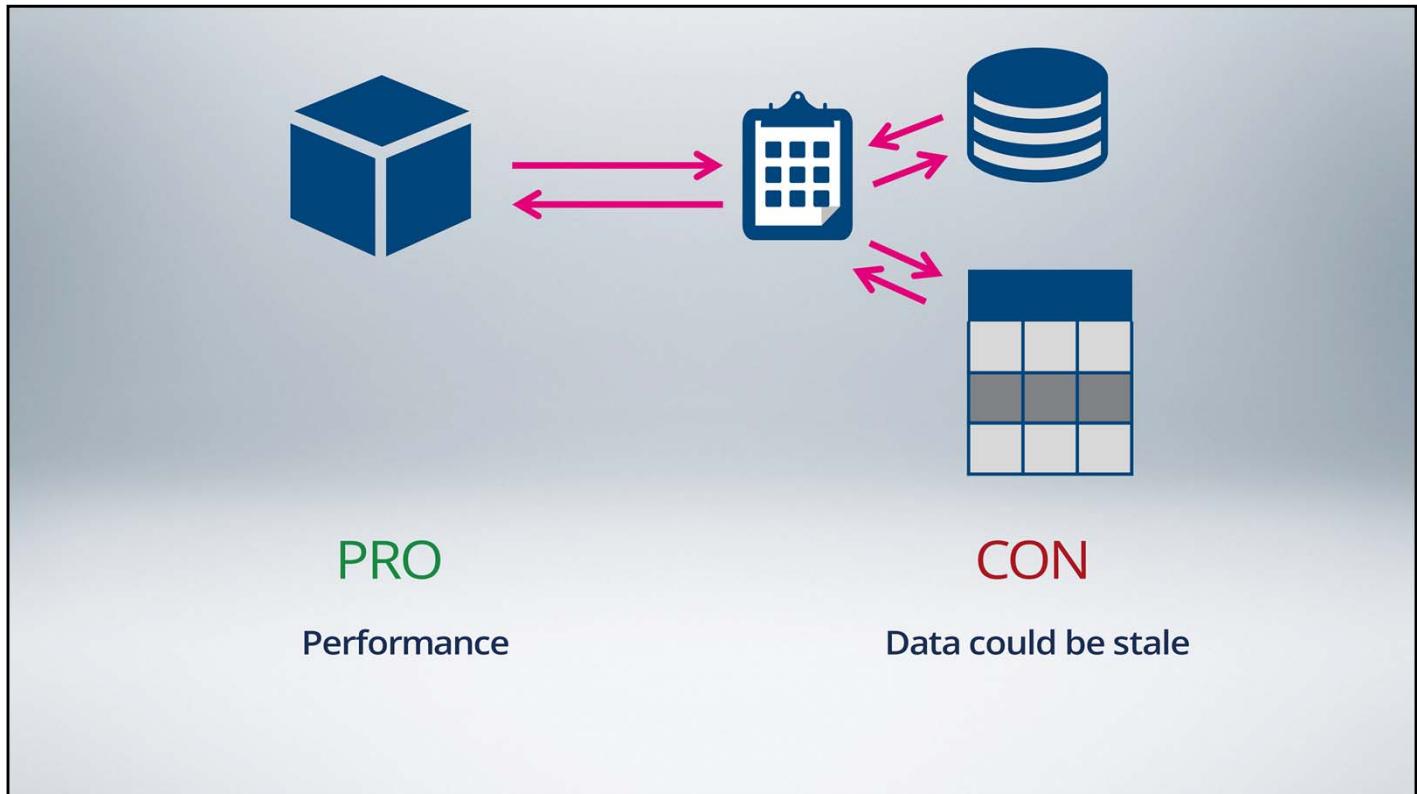
Since we loaded the data page using the Vendor data class we can expand the Vendor data class in the Data Explorer to see the data pages defined for it. We can then click on the data page to configure it further if needed.



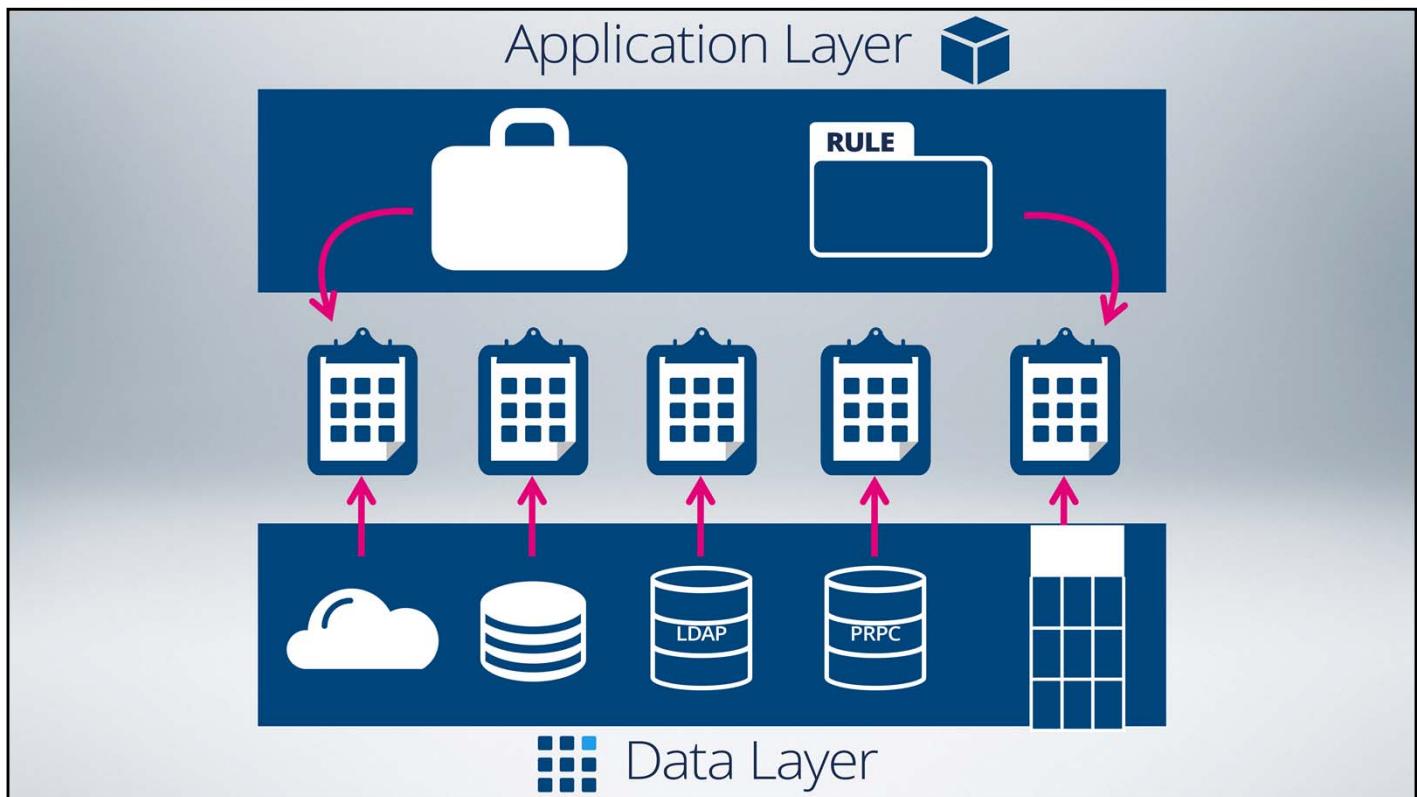
When building an application it's important to know where the system of record for a piece of data is located. This data is then used as reference data in an application.



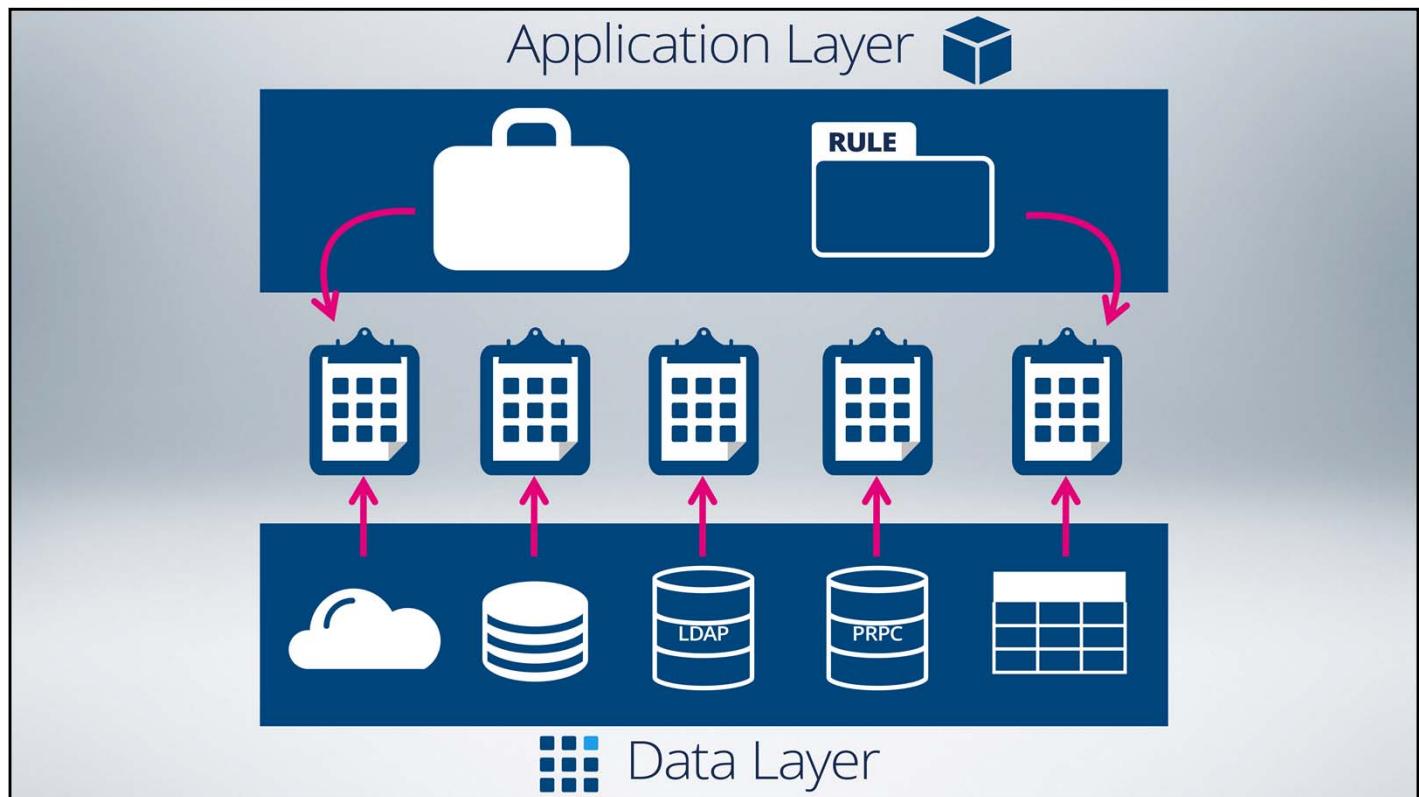
In PRPC reference data can also be created using a data table. Data tables are simple structures that are used to store non-volatile data.



Once we know where the data is and have created a data model we can use that data in our application by creating data pages. A data page loads the data into memory and stores it on the clipboard. This gives us a performance increase since it is faster to access memory than an external system. This could lead to stale data in our application but to handle that, data pages contain a refresh strategy to make sure that the correct data is contained within them.



By having our application access data pages instead of the data model directly we create more maintainable applications that are Built for Change®.



By having our application access data pages instead of the data model directly we create more maintainable applications that are Built for Change®.

## Exercise: Managing Reference Data in Data Tables

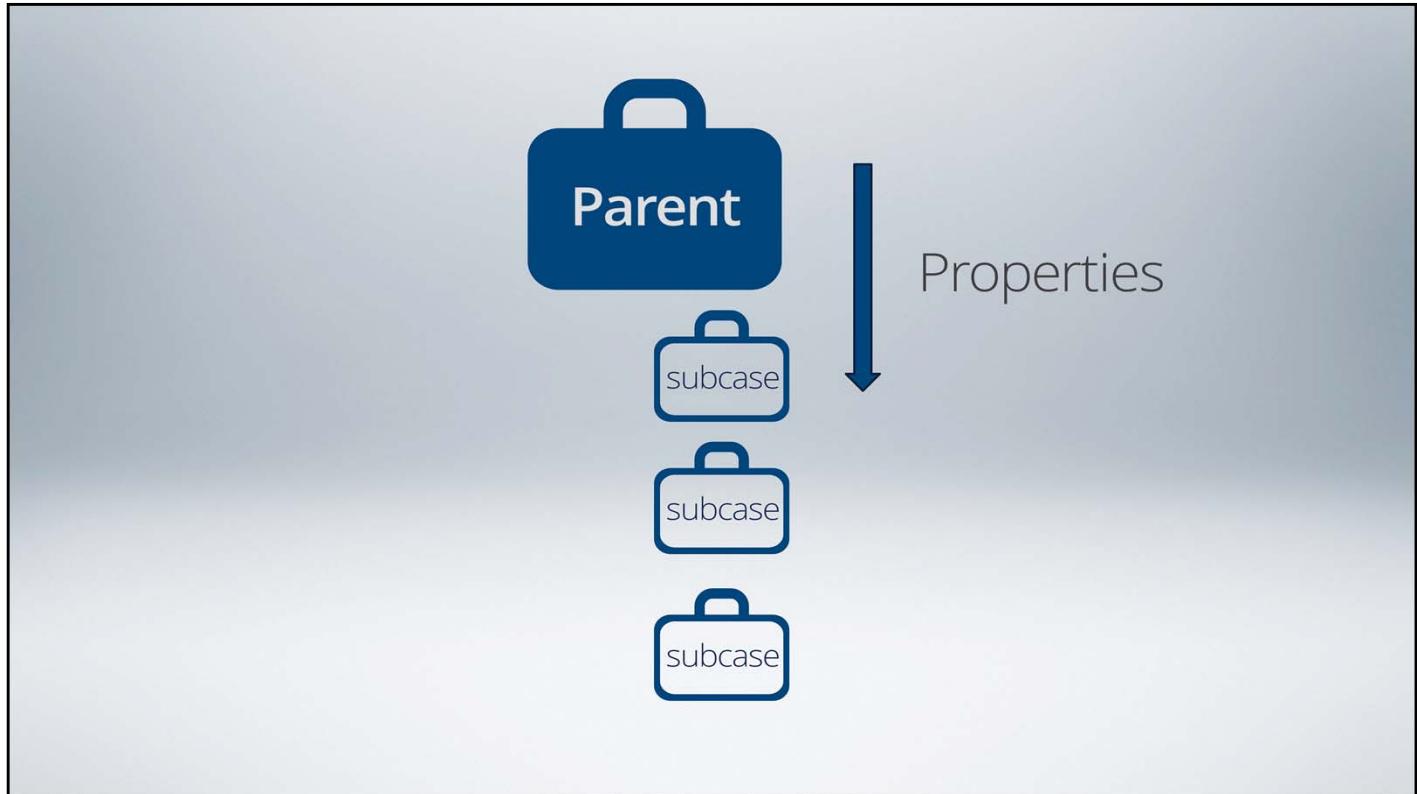


# Sharing Data Across Cases and Subcases

Often times when cases create subcases data needs to be sent from the parent case to its subcases. This lesson covers the different approaches in propagating data from a case to a subcase.

At the end of this lesson, you should be able to:

- Describe the need for data propagation between cases and subcases
- Configure a case to use data propagation

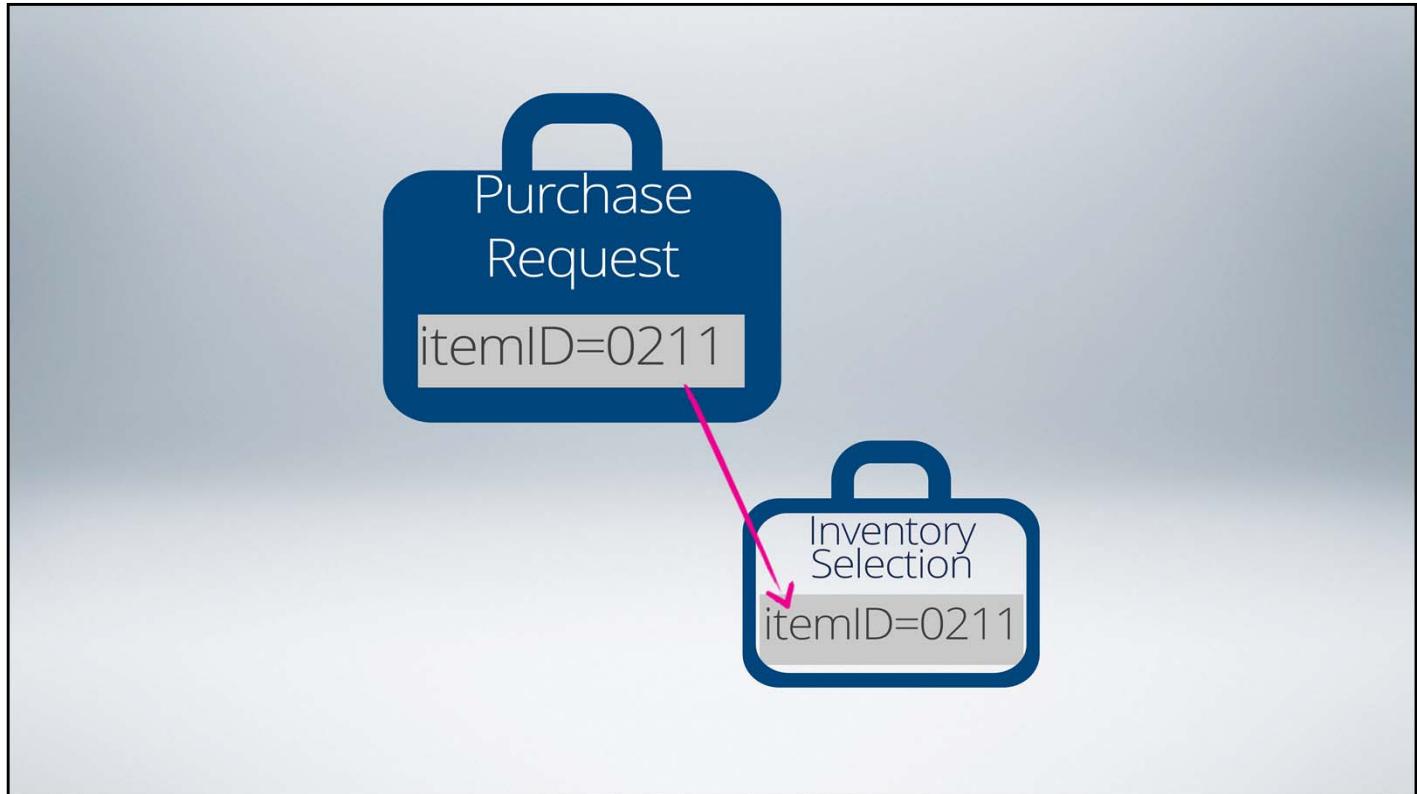


In case management applications where cases are nested in a parent/child configuration, it is quite often necessary to initialize the subcases with data coming from the parent case. In order to do this we use the Data Propagation feature.



Let's look an example to explain this data propagation concept. In the context of our Purchase Request application, two subcase types may be instantiated; Inventory Selection and Purchase Order.

The Inventory Selection subcase needs the itemID's to check to see if the those items are in stock and the Purchase Order would need the billingAddress of where to send the shipment to. The mechanism of copying data within the case hierarchy is called "data propagation".

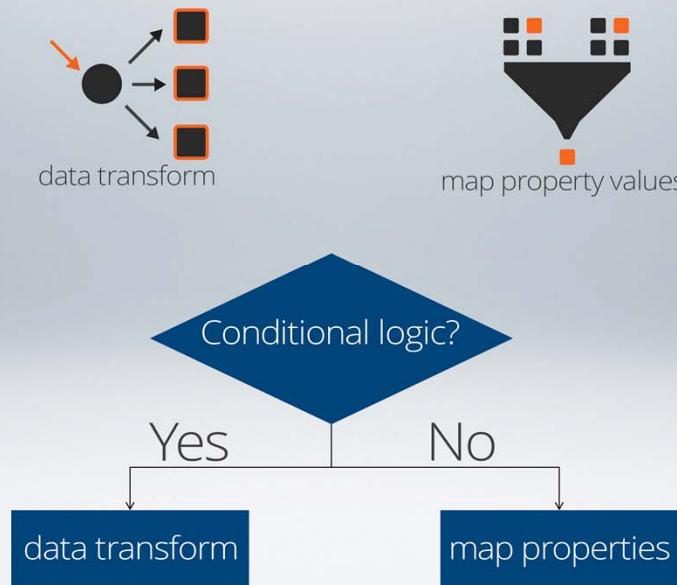


Data Propagation is done only upon instantiation of a subcase. If you propagate a property from the Parent case to a child case, and the property value later changes on the Parent Case, the property on the child case will not be updated. In our example let's say that an itemID is set to 0211, when the Inventory Selection subcase is created the itemID value is propagated.



If further along in the process the itemID as part of the Purchase Request changes to 1776 that value is not automatically propagated to the subcase. Keep this in mind as you are configuring Data Propagation into your case management application.

## Ways to Propagate Data in the Case Designer



There are two ways to propagate data from a case to a subcase. The first way is to map the properties to propagate in the case designer, the other to specify a data transform that will set the correct values. Which option should be chosen? Well it all depends on if you need to do some conditional logic in order to propagate the data. If you need some conditional logic to determine what to propagate, such as you need to loop through a list to see what's selected, then you should use a data transform. If the values can be copied without any modifications then you map the properties.

## Demo



### Configuring Data Propagation

In this demo, we need to configure data propagation for the Purchase Request case. The Purchase Request gathers LineItems and we want to pass the LineItems to the Purchase Order subcase. We configure data propagation from the parent case, which in our scenario is the Purchase Request case.

The data propagation configuration is configured on the Details tab.

We then click Edit to configure Data Propagation.

In the data propagation editor there is a section for each subcase from the parent case. In our example notice there is one for Purchase Order and one for Vendor Maintenance.

We need to configure a new property for propagation. As we can see some properties have already been configured for propagation into the Purchase Order. To add a new property we click the plus symbol.

As we enter LineItems the autocomplete appears and we select the LineItems from Purchase Order.

Now we add the property's data that we want copied in. Once again we can use the autocomplete and select the LineItems but this time from the Purchase Request case.

We also have the ability to add a data transform to do more advanced propagation. This can be done in conjunction with directly mapping properties or everything can be done in the data transform.

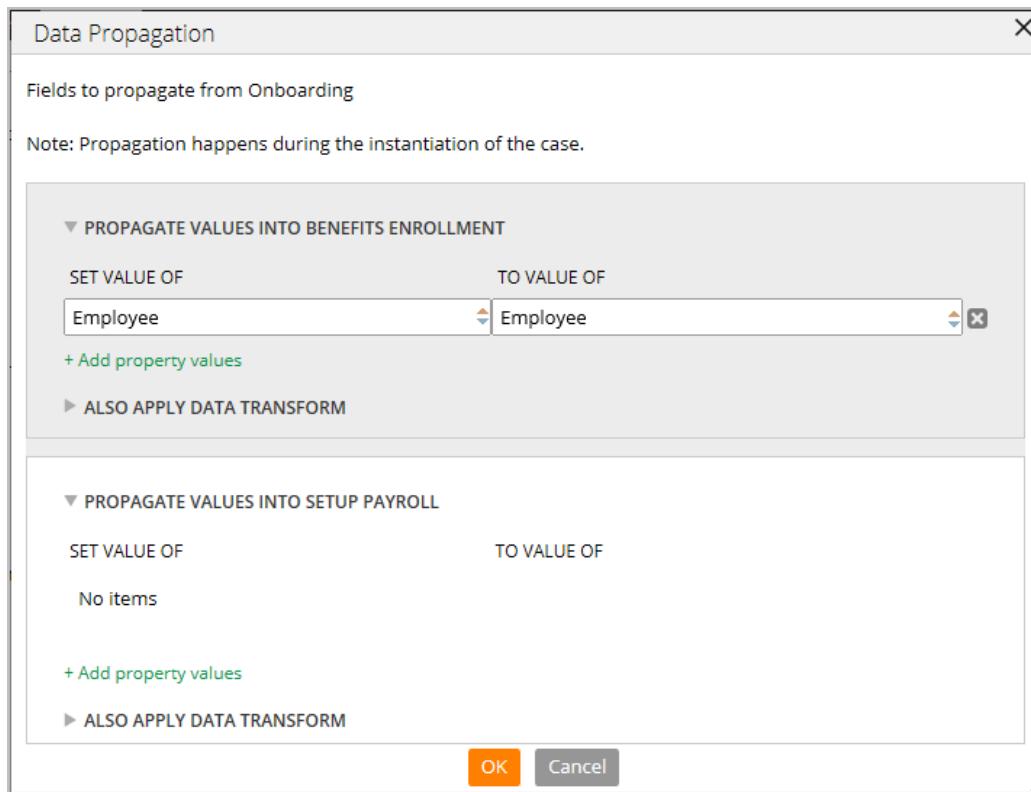
When we are finished configuring our data propagation we click OK and then save the case.

## Pega 7.1.6 Update Notes

### Updates for Pega 7.1.6:

Data Propagation form updated

- Updated labels and instructions make it easier to quickly understand the meaning and purpose of each field.



## **Exercise: Propagating Data from a Case to a Subcase**

### **Exercise: Adding Default Data Transforms**

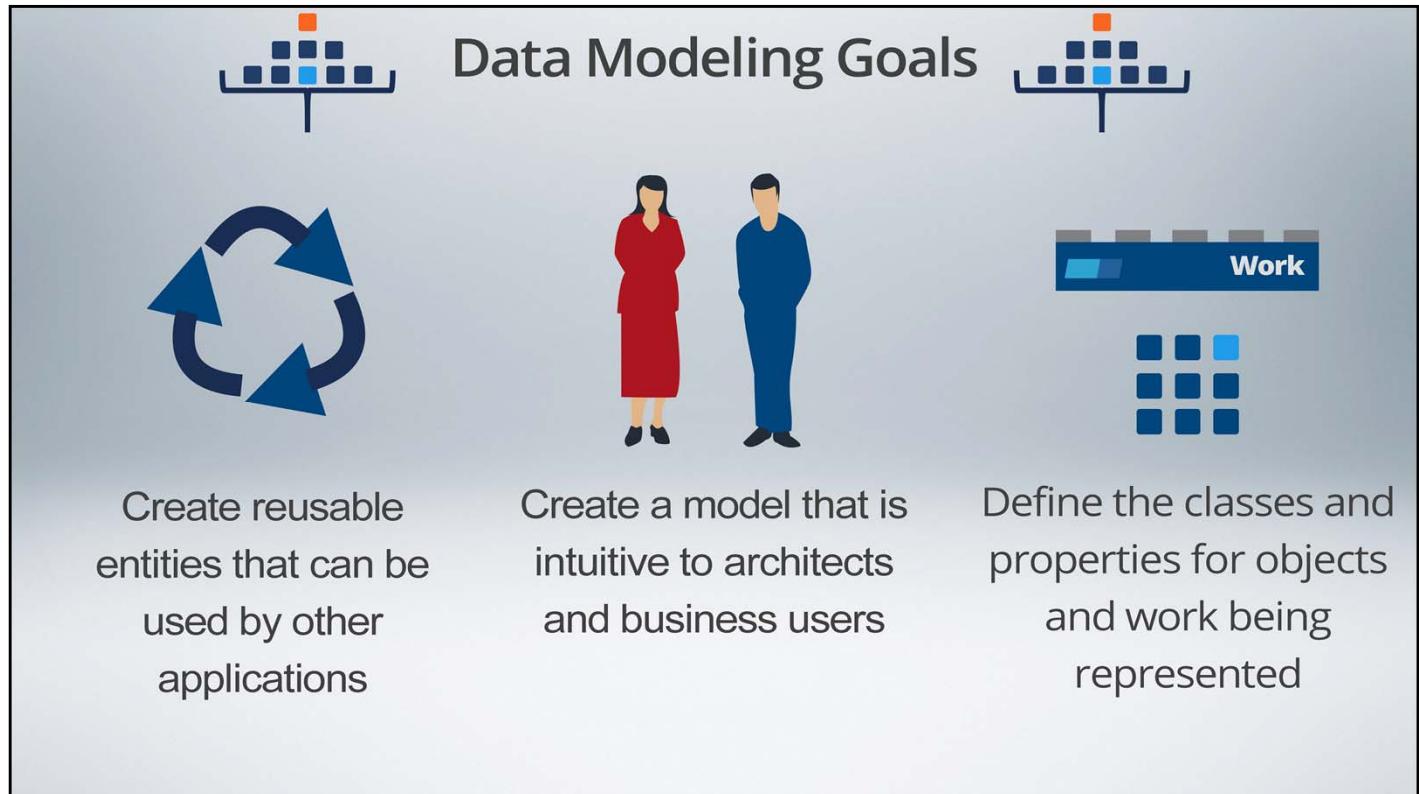


# Guardrails for Data Models

The data model represents what and where our data is in our application. When implementing the data model there are best practices to keep in mind to ensure a successful application.

At the end of this lesson, you should be able to:

- Name two best practices when creating a data model
- Describe the trade offs of performance vs. data accuracy when choosing a data page refresh strategy



Let's discuss some best practices. The process of defining the data model is critical to a successful application. A good data model should define entities that can represent business objects in a way that can be reused by other applications. The model should be understood by both architects and business users. In the end, the model will be the roadmap for the classes and properties that will be used by the rest of the application.

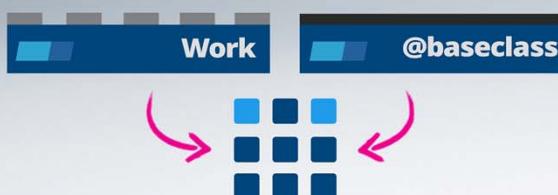
# Tips for Data Modeling



Keep the class structure, data model and the runtime structure clear in your head



Focus on class relationships first



Use inherited properties  
from PRPC when possible



Try to group common properties into data structures rather than work level properties

When modeling data keep these tips in mind. Keeping the concepts of class structure and the data model clear in your head is critical to completely mastering PRPC. It is also important to understand how the runtime data structure can differ from the design time structure since pages can be instantiated as sub classes of their definition. For example the CreditEvaluation page property can be instantiated as either a personal or corporate subclass.

Another tip is that when designing a new application focus initially on the relationships of the classes, which classes will use inheritance and which will become embedded pages, page lists or groups.

Remember to review inherited properties and out of the box classes to ensure that any already defined properties are used.

When designing your work and data structures try to group your properties into logical data classes rather than as direct work properties. This helps promote reuse by allowing entities (such as an account, product or person) to be reused outside of the specific work being performed.

## When not to use data tables...

### Rule-Like Functionality

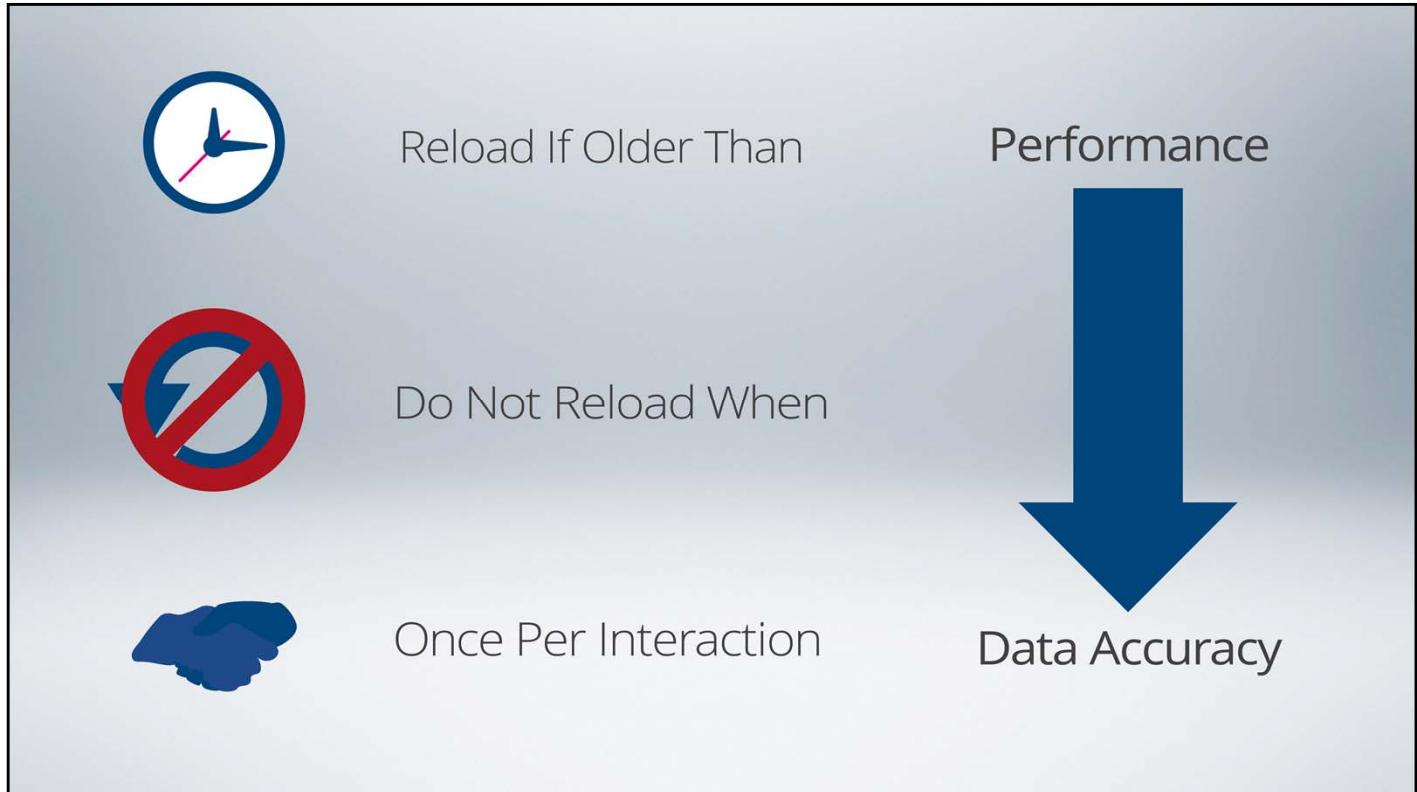
Date Range	Customer Type	Suggest Plan
April – September	Silver	3yr Flex
October – March	Silver	2yr Flex
April – September	Gold	3yr Fixed
October – March	Gold	5yr Fixed

### Large Quantities of Data

Record Number	Government ID
0000001	036-00-3389
0000002	493-00-2737
...	
9999998	430-00-4384
9999999	262-00-2259

Data tables should only be used for, well, data. Storing business logic in a Data Table is not right: that's what we have rules for, including all the wonderful process-routing functionality that rules provide, such as circumstance and decision trees and decision tables. In the example a decision table would be a better way to implement the logic presented in the table.

Data tables are also not intended for large quantities of data. The rule of thumb for sizing a data table is about 100 rows.



Some important things to keep in mind when working with data pages. When working with data pages you will always have to determine the tradeoff of performance vs. accuracy of data. Remember the Reload If Older Than refresh strategy is a countdown for inactivity of a data page. For example, if you set the time to one week you would need one week between the page being accessed before the data would be refreshed. This is more of an ideal strategy for data that will change very rarely. It should perform better because the more often a page is accessed the less the data will be refreshed and therefore save a trip to where the data located. At the same time this strategy can easily result in data that is not accurate. The once per interaction strategy is on the opposite end of that spectrum. For each interaction the data in the data page is reloaded which could impact performance but you have a very high level of data accuracy. The Do Not Reload When strategy lies somewhere in between, it will all depend on the conditions you create that determine if the page should be refreshed. This refresh strategy gives you the most flexibility in determining how often the data should be refreshed.

## **Exercise: Creating an Effective Data Model**

### **Exercise Verification**



## Module 05: Integrating with External Data Sources

This lesson group includes the following lessons:

- Integrating with Databases
- Guardrails for Integrating with External Data Sources

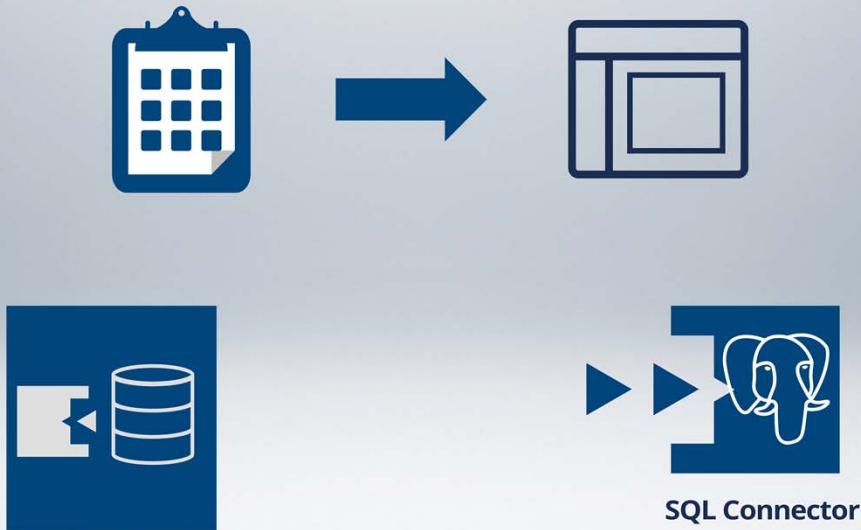
# Integrating with Databases

This lesson discusses how to use the Database Table Class Mapping tool is used to access an external database.

At the end of this lesson, you should be able to:

- Define the two ways to connect to an external database
- List the artifacts the Database Table Class Mapping tool creates
- Use the Database Table Class Mapping tool

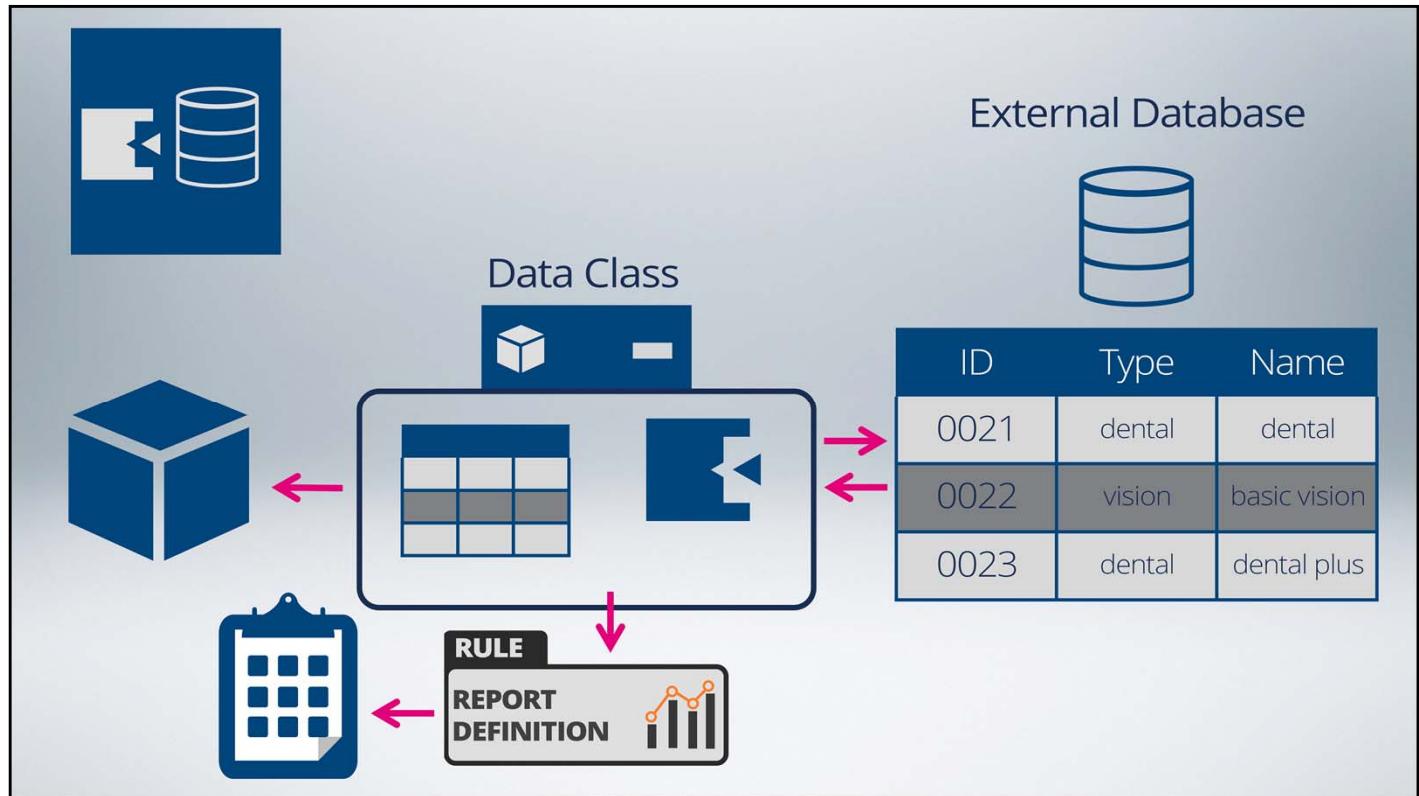
## Getting data from external data sources



We've previously discussed the need to get data from external data sources for our applications. Remember, data is represented in memory using a data page and is accessed via different components such as sections or decision tables.

There are two ways to accomplish this task. The first is to use the Database Table Class Mapping tool. The Database Table Class Mapping tool provides a wizard that generates all the artifacts needed to interact with reference data in an external database. These artifacts include a data class, a database table instance and a link between those two artifacts.

The other way is to use an SQL Connector to connect to the database. We use a Connector when we need to do more advanced queries with the database table. Setting up a Connector is a Senior System Architect or a Lead System Architect task, we will be focusing on using the Database Table Class Mapping tool to setup our integration..



We are going to focus on the Database Table Class Mapping tool. Let's say we have a table in an external database and we want to access the data contained within it.

By using the Database Table Class Mapping tool we create a data class that contains a data mapping and database table instance that references the external table. By creating a data mapping we create a pass-through from our application to a database table of our choosing. We are then able to use the data contained in that external database as if it were within our application.

Now that we have a data class and we have created a mapping, we can build a report definition to get some data. That report definition uses the data class to specify what data is needed.

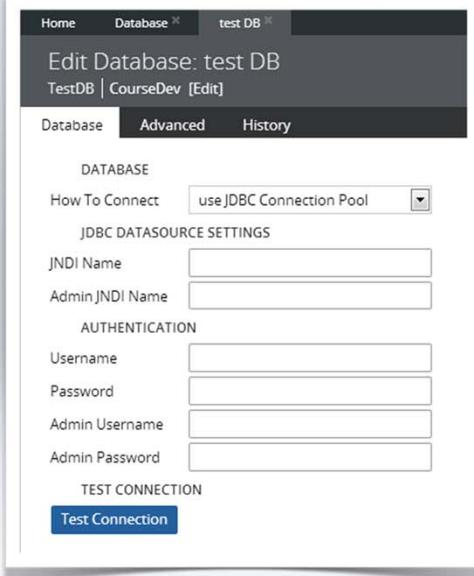
When the report is run the external database is queried to get results.

We then use the results of the report definition to populate a data page.

Now let's look at each of these steps in a little more detail...

## External Database







Which database to use

Which table in that database to use

Name of the data class to create

Which properties to map from database table to data class

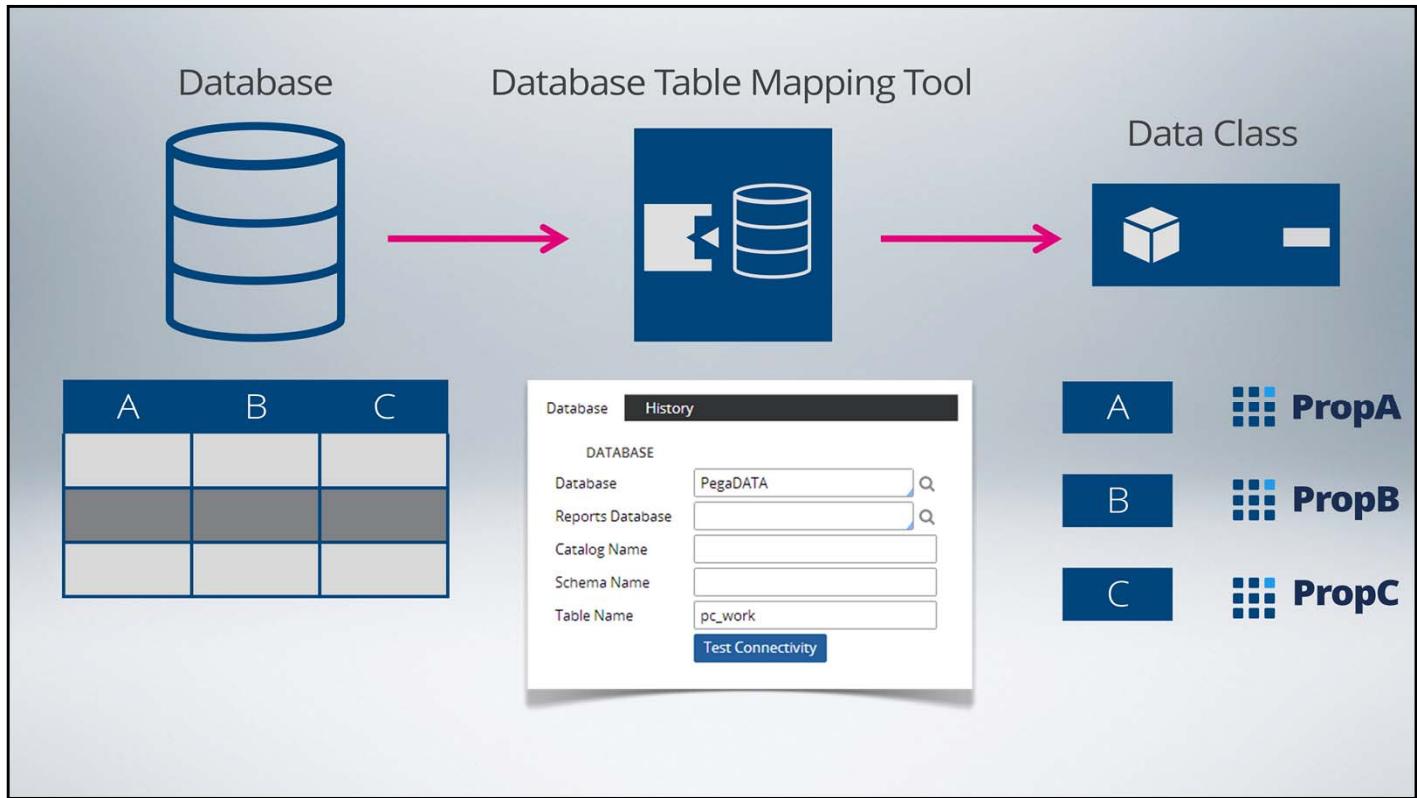
Before we do anything we first need to register the external database within PRPC.

To do this a Senior System Architect or Lead System Architect would register a database instance and provide the appropriate credentials and information on how to connect to the database.

It is important to note the name of the database instance, in this case it is test DB. This is the name we use to refer to this database in the Database Table Class Mapping tool.

Once we have a database configured we run the Database Table Class Mapping tool. The Database Table Class Mapping tool gathers information about:

- which database to use
- which table in that database to use
- the name of the data class to create
- which properties to map from the database table to your data class.



As an example let's say we have a table that has three columns A, B, and C, we want to use the Database Table Class Mapping tool to create a data class that allows us to use that data in our application.

When we run the tool we specify the database and database table we want to use, this is stored in the database table instance.

We then select which columns of that table to use and then map those columns to properties for our data class. In our example we map the columns A, B and C to the properties PropA, PropB and PropC. This mapping is stored as part of the data class.

The Database Table Class Mapping tool also creates a database table instance where the database name and table name resides, this is used by the data class when it needs to get data.

## Demo



### Using the Database Table Mapping Tool

In this demo we will use the Database Table Class Mapping Tool to access a Shipping Carriers table in an external database. To access the Database Table Class Mapping tool we go the Database Class Mappings landing page.

The Database Class Mappings landing page shows us all of the existing database table mappings. To create a new one we click New External Database Table Class Mapping.

Then we choose the database where the table resides. The database name has to already have been registered. For this example our table is actually in the Pega database, which is named PegaDATA.

Now we need to enter the name of the table. Our table is named SAE2\_SHIPPING\_CARRIERS.

Next we enter a RuleSet name and a version. Then we pick the name of the data class we want created. The class name we use cannot exist as the Database Table Class Mapping tool generates a new class.

Once you enter the table name, Pega queries the table and automatically discovers the different columns that exist. We then determine which columns we want to access and the corresponding property names we want to use in the data class. Once everything is configured we click Save and our new class and database table instance is generated.

Let's take a look at what is created. First we can see the new mapping entry, from that we can actually take a look at the columns from that table.

Here we get a list of all the columns that exist in the table we mapped to and additional metadata information like the column type and column size.

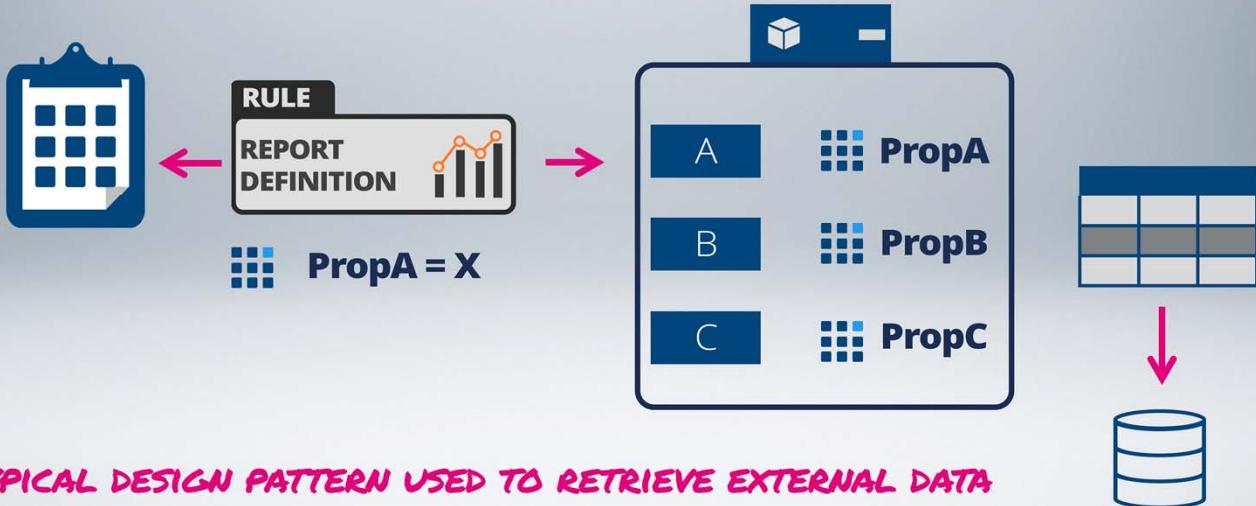
Next we can click the database icon and see the database table instance which shows us the database and table we used for this mapping.

This shows us the database and database table names we entered when going through the wizard.

Lastly we can take a look at the data class we have built. This is just like other data classes we built except that we can also examine the column to property mapping we created by clicking the External Mapping tab.

Now when we use our data class its data comes from the external table.

## Using Data class to access external data



Now that we have created a data class using the Database Table Class Mapping tool, we want to be able to leverage that class in our application.

Remember we want to store reference data in a data page, and one way to do that is with a report definition.

Let's say we create a report definition and we want to retrieve all the records where PropA = X. How does this work?

When the report definition is run the data class looks at the properties used in the report and maps the properties with the column names.

The data class also uses the database table instance to determine where the data comes from.

The report definition can then query the database to get the correct data.

## Exercise: Using the Database Table Class Mapping Tool



# Guardrails for Integrating with External Data Sources

When creating an application you will undoubtedly need to access an external database for data. When doing so it is important to follow Pega best practices for integration.

At the end of this lesson, you should be able to:

- State at least two best practices to follow when working with the Database Table Class Mapping Tool

## When using the Database Table Class Mapping Tool



You always want to minimize the amount of data that your application has to pull from an external resource.

2 ways to accomplish this task and they can be used separately or together.

Let's discuss some best practices when integrating with databases using the Database Table Class Mapping tool. You always want to minimize the amount of data that your application has to pull from an external resource. There are two ways to accomplish this task and they can be used separately or together.

## Map only required properties

Key	Column Name	Data Type	Property Name	Property Type	Map All/None <input type="checkbox"/>
✓	ID	VARCHAR2	ID	Text	<input checked="" type="checkbox"/>
✗	NAME	VARCHAR2	NAME	Text	<input checked="" type="checkbox"/>
✗	TYPE	VARCHAR2	TYPE	Text	<input checked="" type="checkbox"/>
✗	DESCRIPTION	VARCHAR2	DESCRIPTION	Text	<input type="checkbox"/>
✗	EMPLOYEE_COST	FLOAT	EMPLOYEE_COST	Decimal	<input checked="" type="checkbox"/>

The first way is when you use the Database Table Class Mapping tool. When going through the wizard you may be tempted to map all the columns to properties, you should only map the properties that are actually needed for your application. By mapping extra columns you are adding unnecessary data to your application.

## Use filters to limit data retrieved



Filter data in a report definition

The other way to limit the amount of data is to make use of filters when using a report definition. By filtering the amount of data that is returned we help to optimize the performance of our application. We will discuss report definitions and filtering in a future lesson.



Key	Column Name	Data Type	Property Name	Property Type	Map All/None
✓	ID	VARCHAR2	ID	Text	✓
□	NAME	VARCHAR2	NAME	Text	✓
□	TYPE	VARCHAR2	TYPE	Text	✓
□	DESCRIPTION	VARCHAR2	DESCRIPTION	Text	□
□	EMPLOYEE_COST	FLOAT	EMPLOYEE_COST	Decimal	✓

Key	Column Name	Data Type	Property Name	Property Type	Map All/None
✓	ID	VARCHAR2	ID	Text	✓
□	NAME	VARCHAR2	Name	Text	✓
□	TYPE	VARCHAR2	Type	Text	✓
□	DESCRIPTION	VARCHAR2	DESCRIPTION	Text	□
□	EMPLOYEE_COST	FLOAT	EmployeeCost	Decimal	✓

Remember that when you use the Database Class Mapper tool that one of the artifacts it creates is a data class with properties for the columns that you mapped. The mapper queries for the column names in the table you specified and the default property names are the column names. These may violate our property naming best practices that we talked about earlier. For instance the column name might be all in capital letters or there may be an underscore in a name. It is a best practice to rename all the property names using the guardrails we discussed before.



## Database Table Class Mapping = Best Practice



**Connectors** = advanced SQL queries

- ▼ foreign key joins
- ▼ accessing stored procedures
- ▼ managing two phase commit rollbacks

When accessing databases it is the best practice to use the Database Table Class Mapping tool to configure access. The artifacts built by the Database Class Mapping tool will be all that you need for almost all use cases. There are a few instances though where a SQL Connector is used to access a database. Connectors can be used for more complex SQL query needs for instance using foreign key joins, accessing stored procedures and managing two phase commit rollbacks.

## Exercise: Integrating with External Data Sources Verification



## Module 06: Creating Engaging User Experiences

This lesson group includes the following lessons:

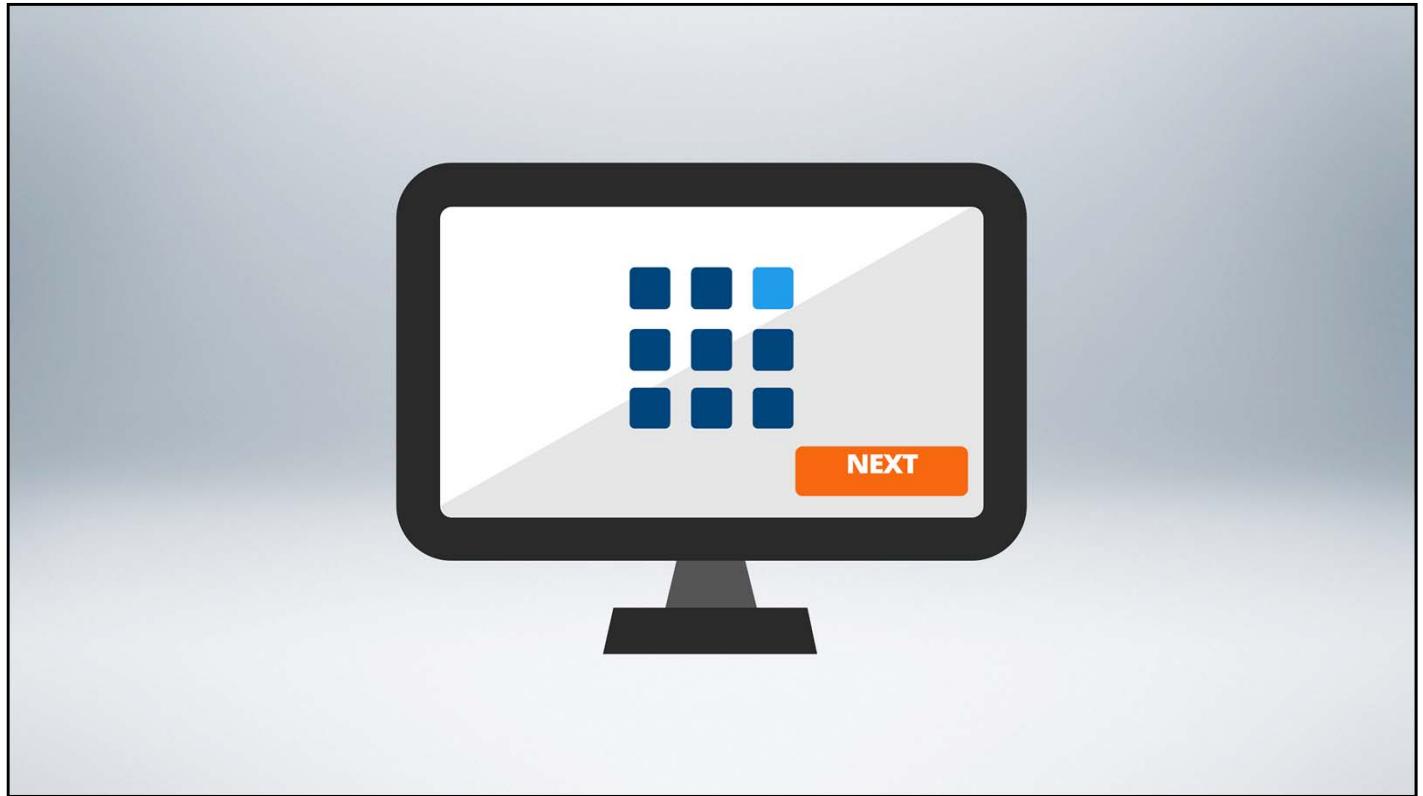
- Designing the User Interface for Reuse and Maintainability
- Building Assignment Focused (Intent-Driven) User Interfaces
- Best Practices for Designing the User Interface
- Using Advanced User Interface Controls
- Managing Data for Selectable List Controls
- Building Dynamic User Interfaces
- Validating User Input
- Guardrails for Creating Engaging User Experiences

# Designing the User Interface for Reuse and Maintainability

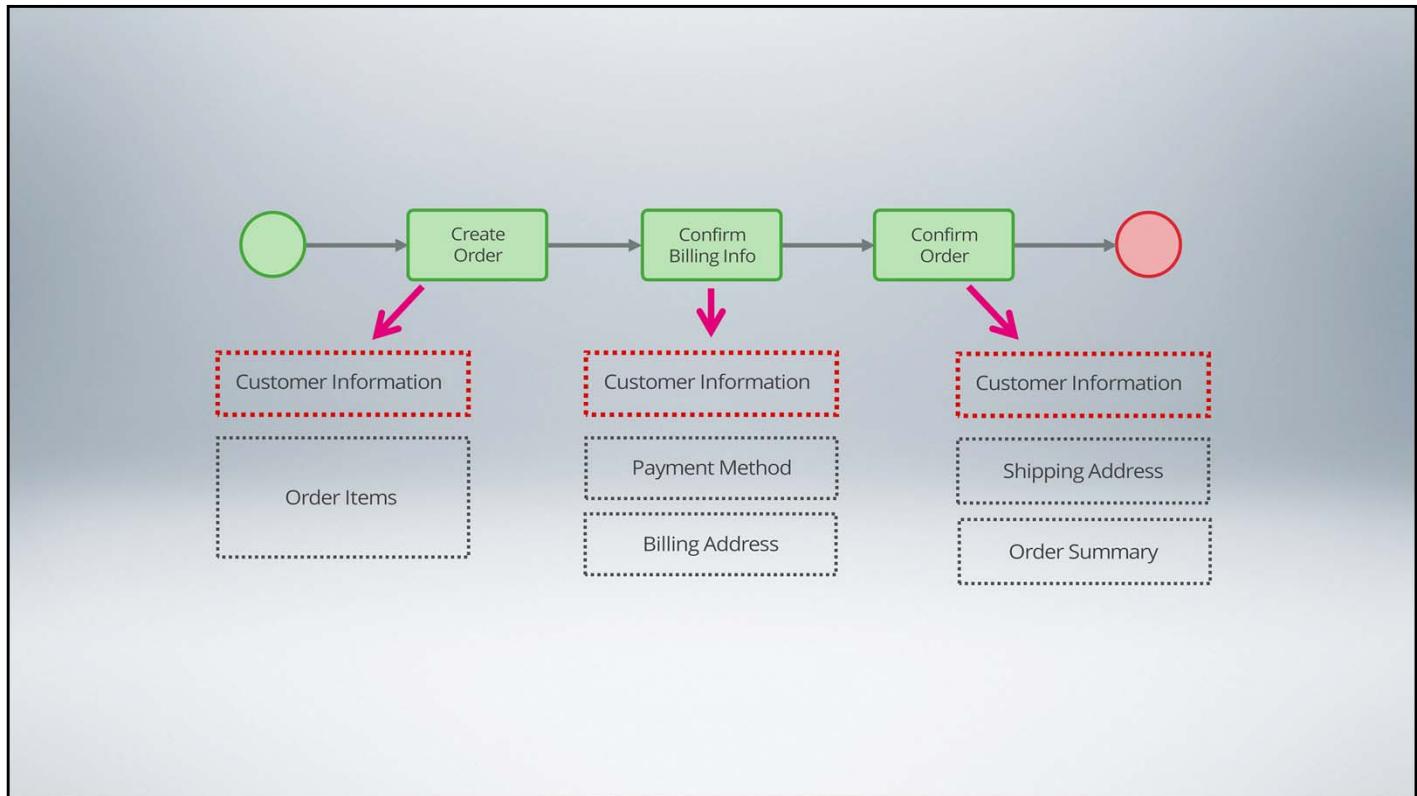
In this lesson, we will explore the best practices for maximizing the reusability of user interface components

At the end of this lesson, you should be able to:

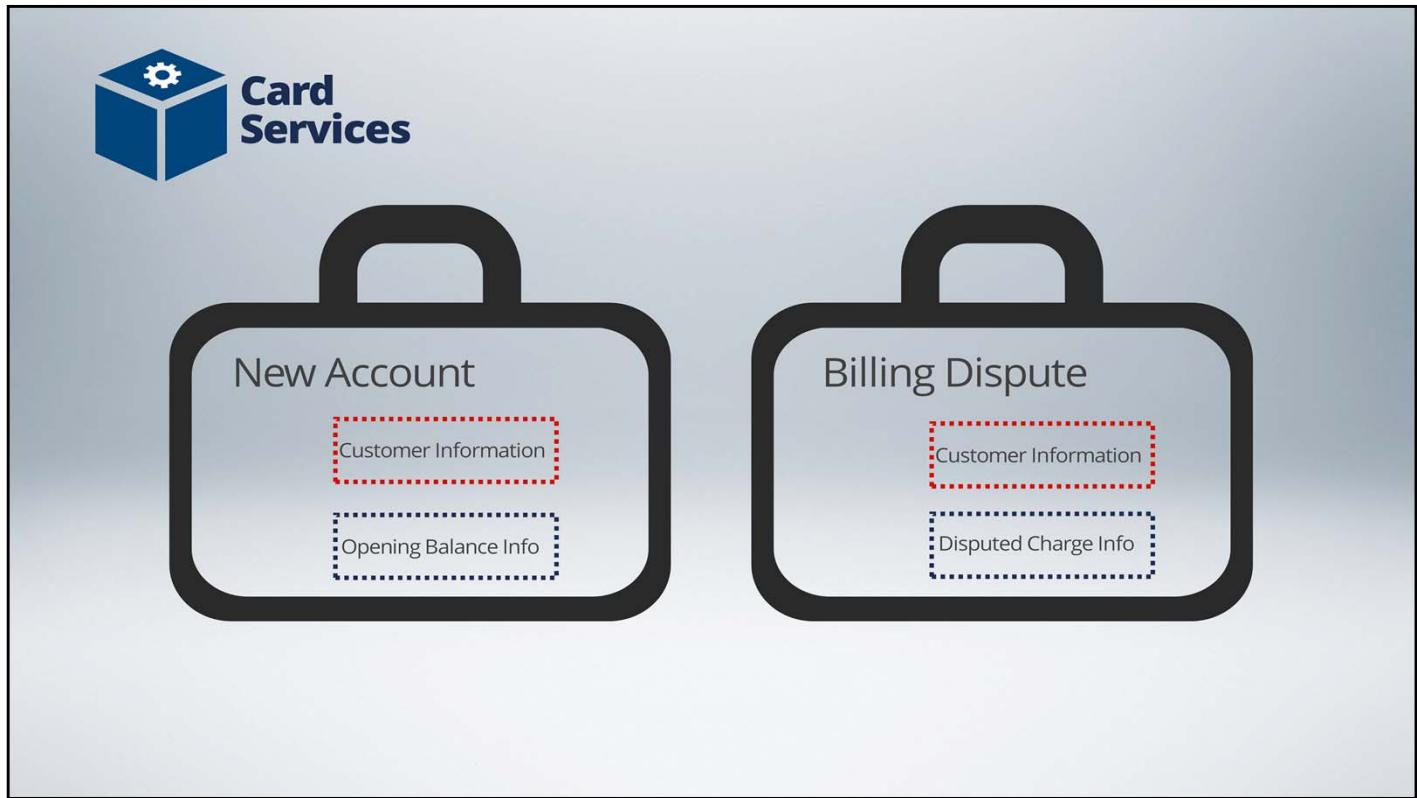
- State the importance of reusable UI components
- State the importance of naming conventions and documentation when creating reusable UI components
- Choose the most appropriate “Applies To” class for user interface components
- Include a section applied in a data class in a section in the case type class



When designing user interfaces, one requirement most often encountered is the need to display a common set of data elements across multiple screens.



For example, regardless of where a customer is at in the ordering process, we must display their contact information until the ordering process is completed.



Another requirement might be that a common set of data elements be presented across multiple screens in different case types in a single application.

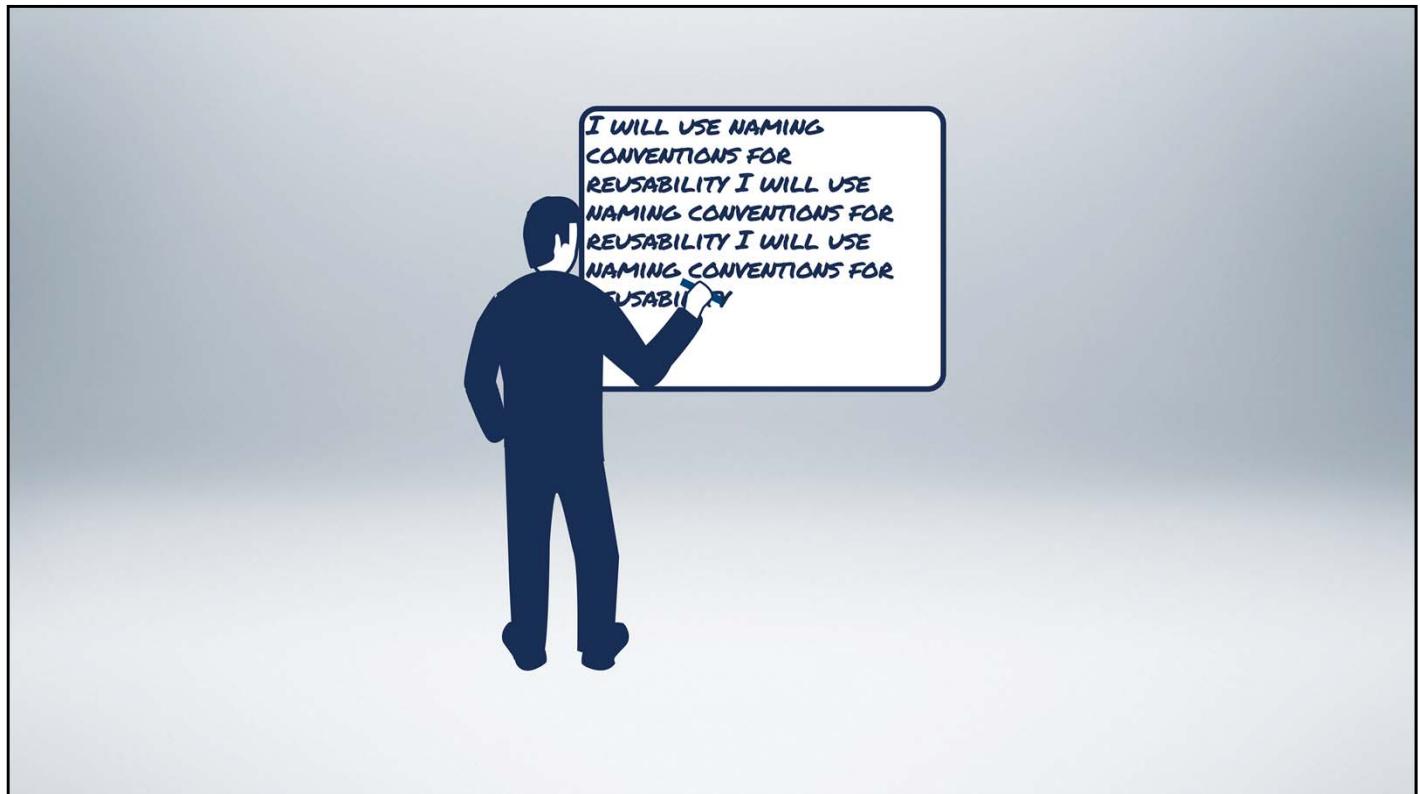
For example, in a Card Services application,

we might have an New Account case type and a Billing Dispute case type,  
both of which need to display the same information about the customer,  
along with unique information for each case type.

Building and maintaining all these pages, and making sure they are consistent in structure and layout, can become increasingly time consuming and prone to error - if not done correctly.

As with every other aspect of building our application, it's important to take a step back and broadly identify what UI components are generic  
versus which are unique to a given case type.

By creating - and using - reusable UI components in our application, we can build user interfaces that are always consistent in structure and layout, and an application that is scalable and extensible.



When designing for reusability, the importance of naming conventions cannot be overstated – which is why it keeps coming up.

Harness	New_2	New (01:02:03)
Flow Action	Orders_2	CreateOrder
Section	InsurancePolicy	CoverageList

PURPOSE IS CLEAR THROUGH NAME ALONE

Provide meaningful and relevant names for all user interface components.

This may seem obvious, but nonetheless, in the heat of the moment of a fast-paced design effort, we may be tempted to let slip the occasional “New underscore two” harness.

Or deploy the all new - and hotly anticipated - “Orders underscore Two” flow action.

Or name a section “Insurance Policy” - which could actually describe ANY section in an insurance policy application.

When building UI components,

the short description should indicate what the component does or contains.

For example, if the flow action is used to create an order, the short description should be just that - “Create Order.”

If the section is used to show a specific data set – such as a policy holder’s insurance coverage – then the short description should read “Coverage List”.

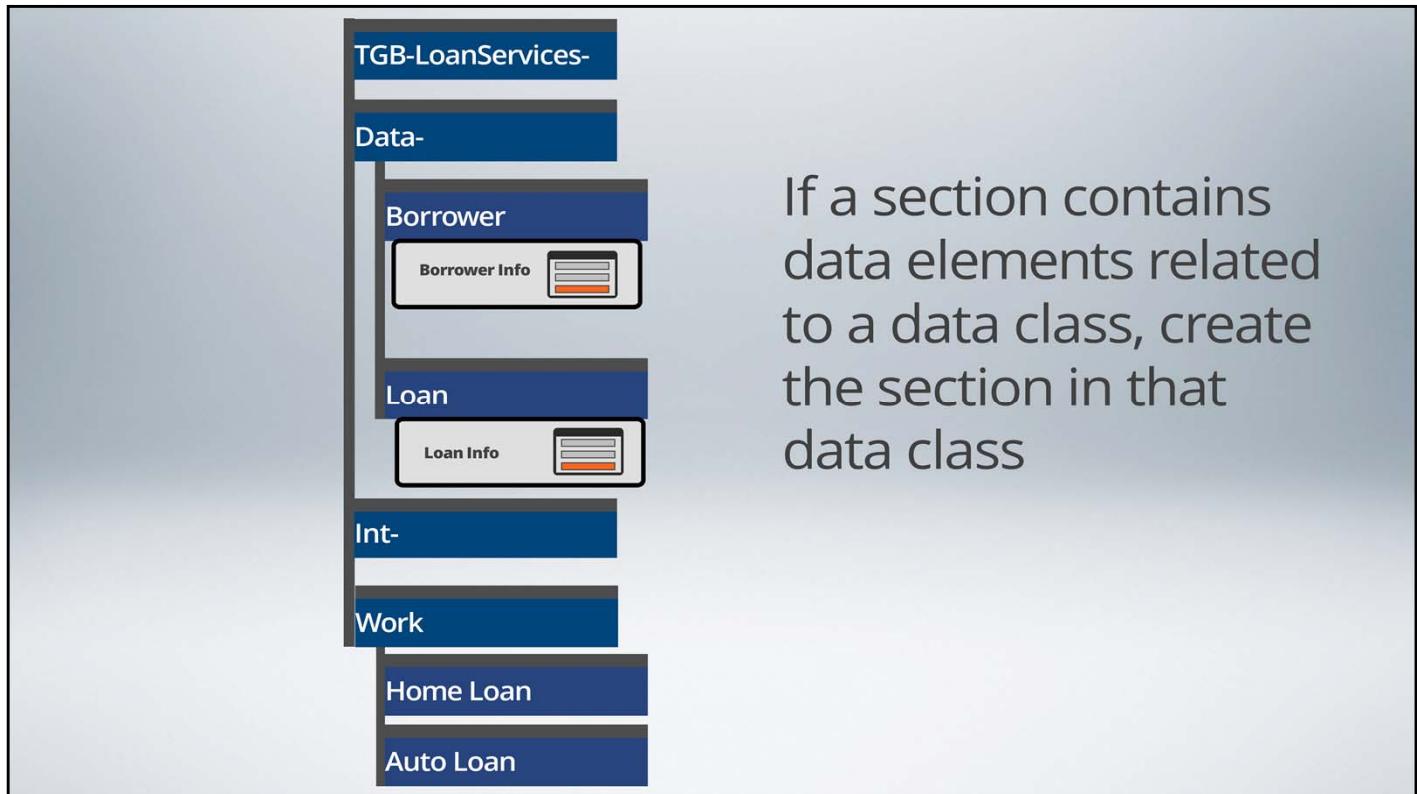
And...when it comes to different versions of the same component, always – always – use a new RuleSet. Never stamp a version of any component into the short description.

In short, use short descriptions that allow another architect to clearly understand the purpose of any given component.

This is our primary way to communicate with others who may wish to extend the application or support it in the future.

Another important factor for ensuring a successful reusability strategy is to complete the fields on the History tab.

This is where people will look when they need more descriptive information as to what a rule does. Well documented rules allows them to be more easily reused as they provide other developers a clear description of how and why to use a rule.



If a section contains data elements related to a data class, create the section in that data class

As we know, the “Applies To” class for any given rule determines its scope of reusability.

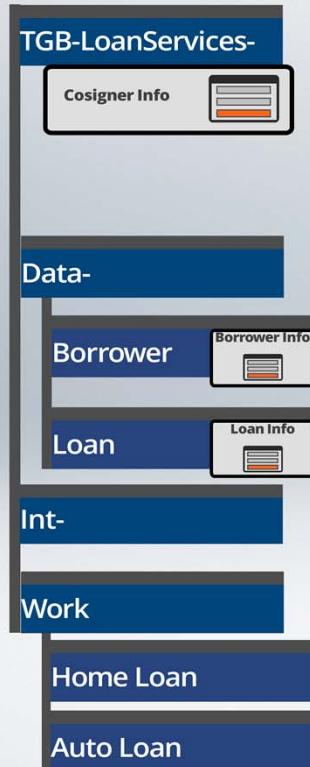
There are many different layers we can use, however let's focus our attention on a single application's implementation layer.

Every application will have three main classes: An abstract class named Data dash, where we store all rules associated with our data model, for example, our data objects; an abstract class named Int dash, where we would store all rules associated with integrations; and, a concrete class named Work.

This class is where we store all rules associated with the individual case types.

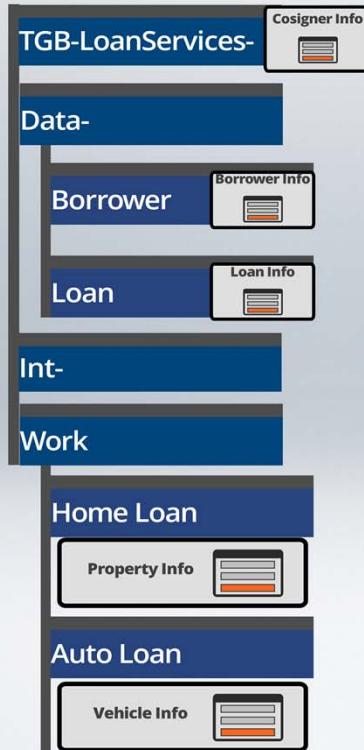
When deciding on the appropriate “Applies To” class for sections, consider whether the section contains data elements related to a data class.

If it does, use the data class as the “Applies To” class.



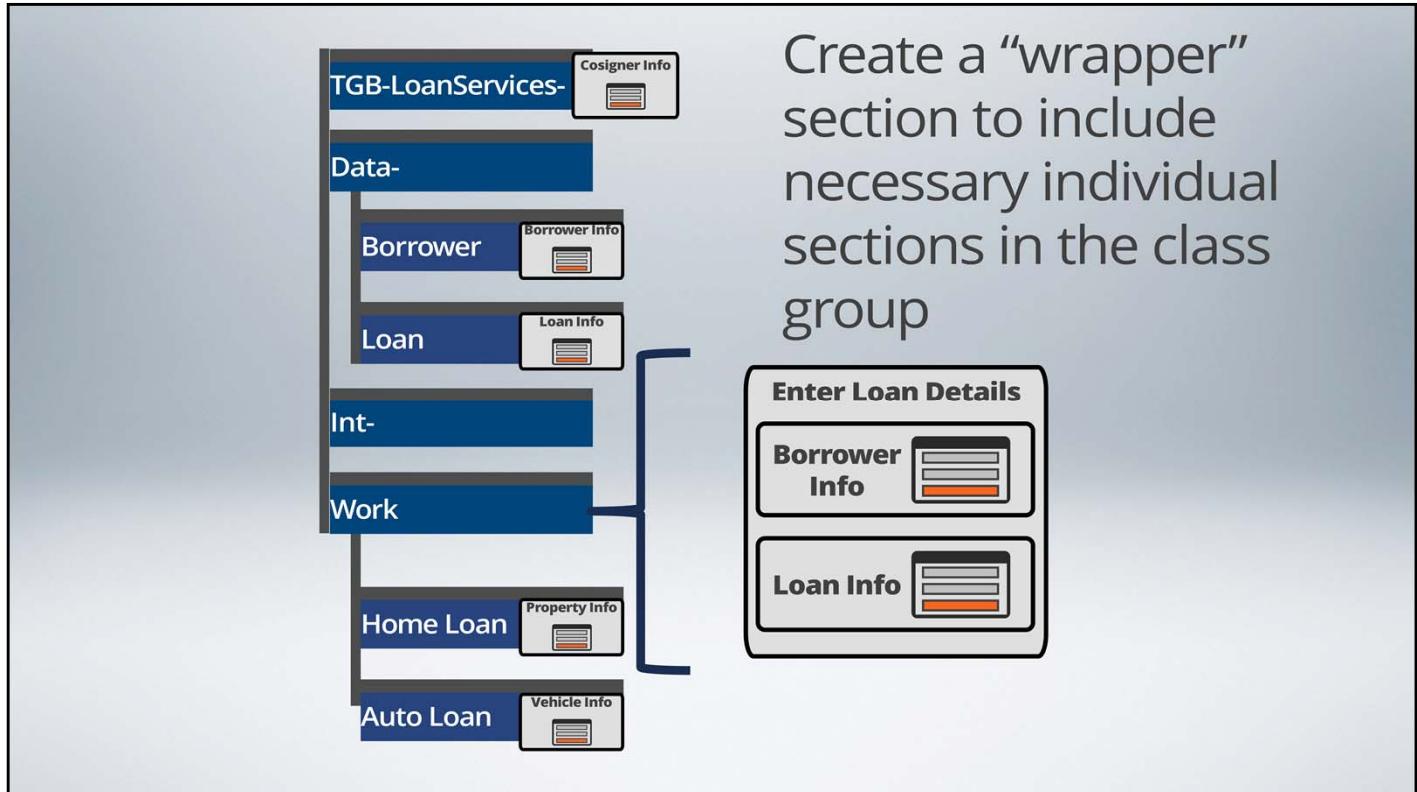
If a section contains data elements relevant to more than one case type, apply to the application abstract class

If the section contains data elements relevant to more than one case type, then apply that section to the ORG dash application dash work class.

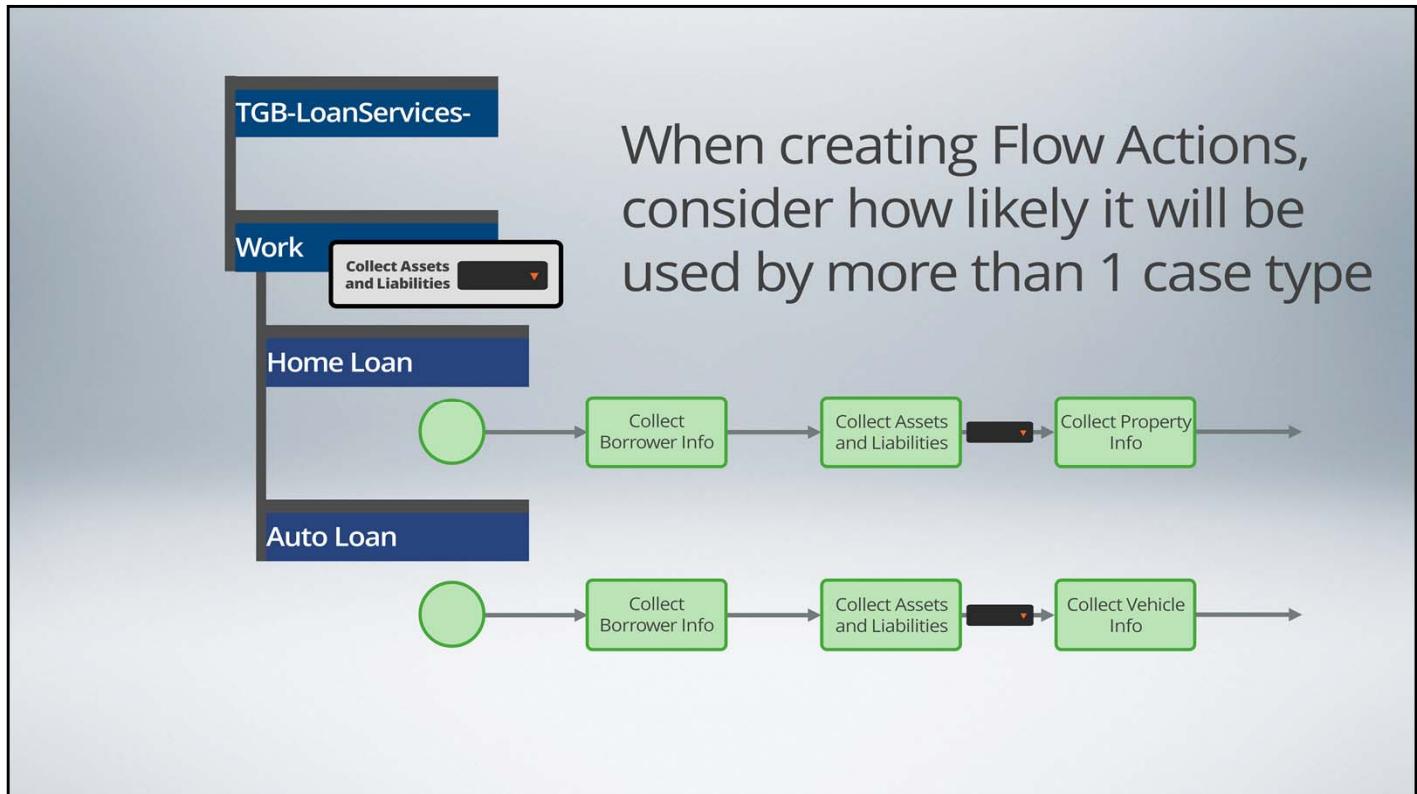


If a section contains data elements unique to a given case type, create the section in the case type class

Finally, if the section contains data elements unique to a given case type, then apply that section to the ORG dash application dash work dash case type class.



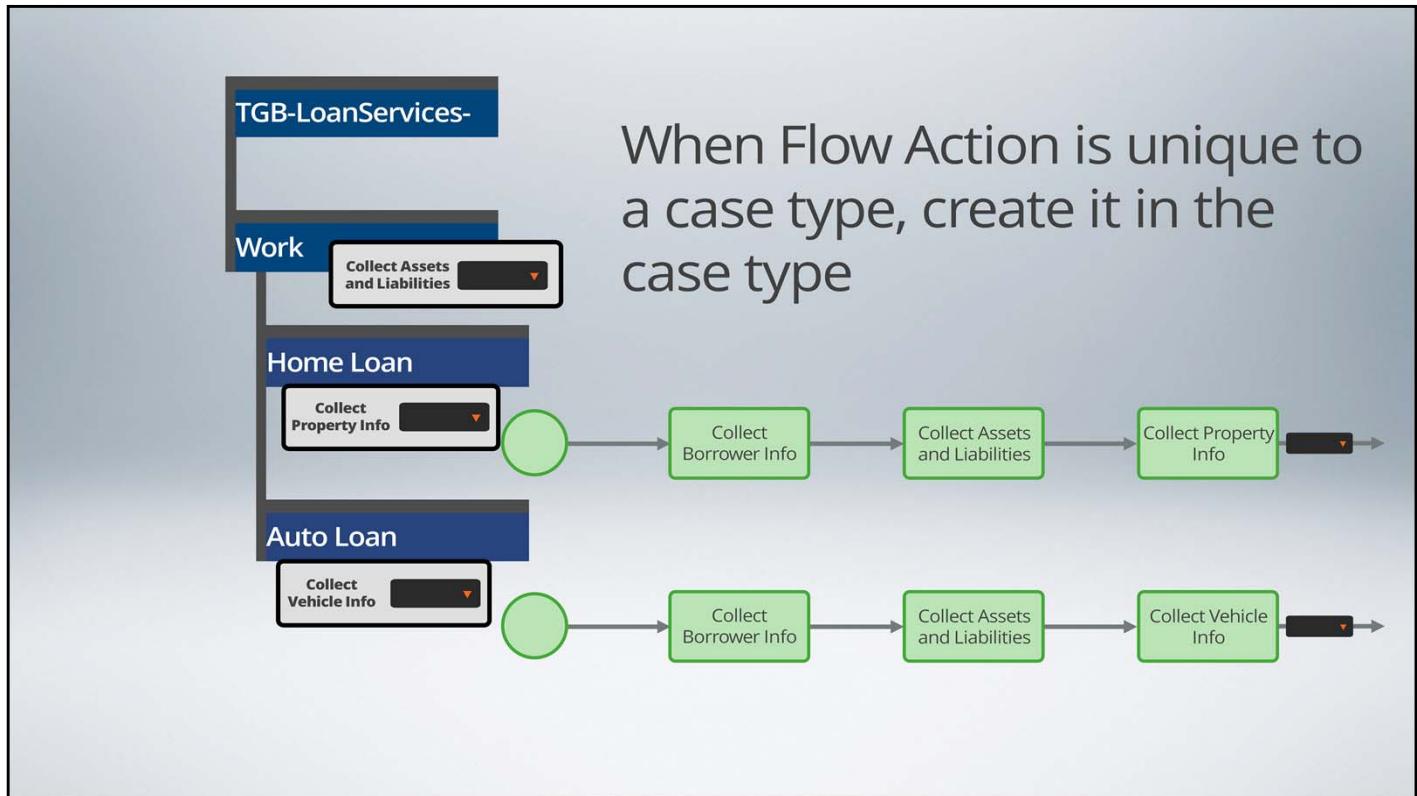
To assemble these widely dispersed sections into a single UI, create a “wrapping” section in the appropriate work dash class, then “include” the necessary individual sections.



When creating flow actions, consider how likely it is to be used by more than one case type.

If the action needed to be taken – in this example, collecting the assets and liabilities information for a loan – is the same regardless of the case type, then we would apply the flow action to the work dash class.

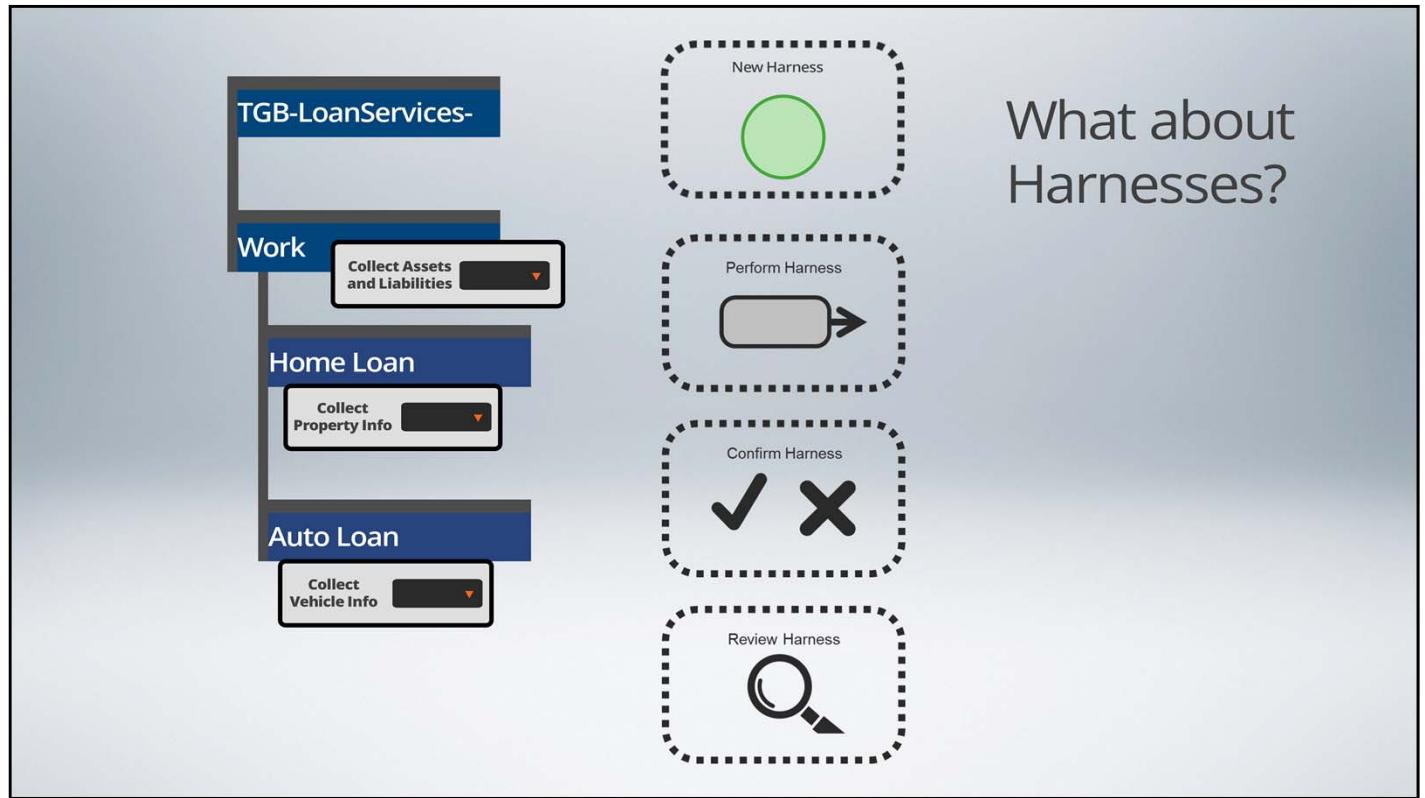
**Then, we would reference the common flow action at the appropriate point in the individual flows.**



If the action needed to be taken is unique to a case type,  
then apply the flow action to the appropriate case type.

Once again, we would reference the unique flow actions at the appropriate point in the individual flows.

There are a good number of out-of-the-box flow actions available for things like presenting standard approval or rejection screens. We won't go through those now, but consider taking the time to become familiar with these components over time.



That leaves us with harnesses.

Remember the four “standard” harness types?



What about Harnesses?

Harnesses are automatically created in the application class.

Simply refer and use.

To customize, save a copy to the appropriate case type class.

When the application is first created, one of each of these standard harness types – named after the type of harness it represents – is automatically created in the ORG dash application class.

We simply refer to these standard harnesses in any given flow in any given case type. If we have a requirement that requires a unique, customized harness, we would save a copy of that harness and apply it to the appropriate case type class.

## Demo



### Use A Section From A Data Class In The Case Type

Sections related to a data class are preferably placed in the data class itself to increase reusability. For example, in the data class SAE-HRSvcs-Data-HRPlan we have a section called SelectPlan.

This section is used when selecting the benefit plans. In other words, it is reused across all benefit types.

As you can see, the medical and dental plans are already in place. Let's add a section for the vision plan as well. To do that we need to add a section.

The page context is a clipboard page and the data class is SAE-HRSvcs-Data-HRPlan.

The clipboard page is the DentalPlan property and the section is called SelectPlan.

The SelectPlan section has a parameter called type used to filter the plans in the dropdown. We will configure this control with a source in a later exercise.

In this case we want to show only vision plans so we set the value to vision.

Let's add a header so that it fits the medical and dental sections.

This is how easy it was to add a benefit type. You'll see how the SelectPlan section is configured later.

## Exercise: Creating Reusable User Interface Components

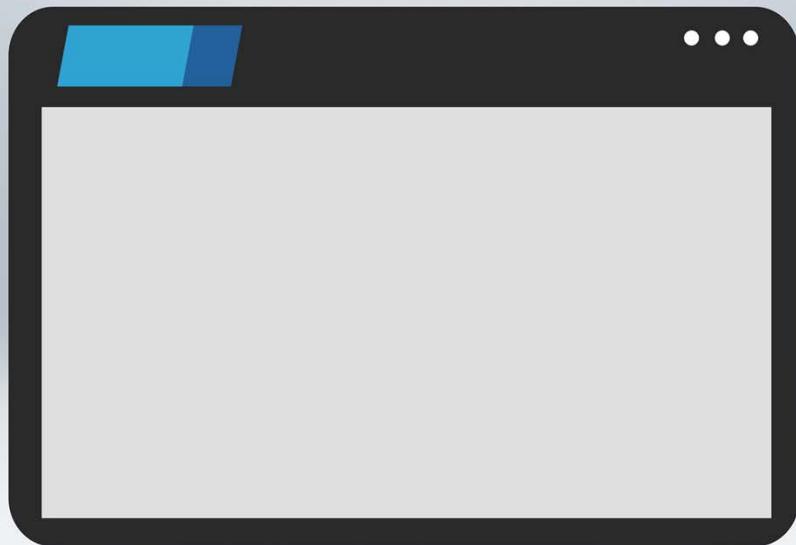


# Building Assignment Focused (Intent-Driven) User Interfaces

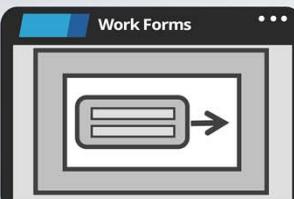
This lesson provides a review of the core user interface components and how to build and organize user screens so they are built for change.

At the end of this lesson, you should be able to:

- Name the three core User Interface components and define their relationship to one another
- Design intent-driven work flows
- Identify three simple rules which will always improve the user interface



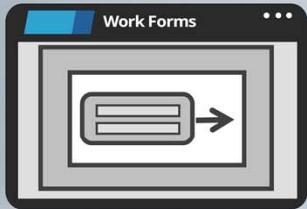
Let's begin this lesson with a review of the core components that make up the user interface in a Pega application.

	Creating, viewing and working on cases
	Designing, and modifying, case management applications Built and maintained by Pega
	Contains the information a user needs to work on a case

User interface components are used for three purposes. There is the “Case Manager” dashboard and portal, which controls and displays a workspace for business users.

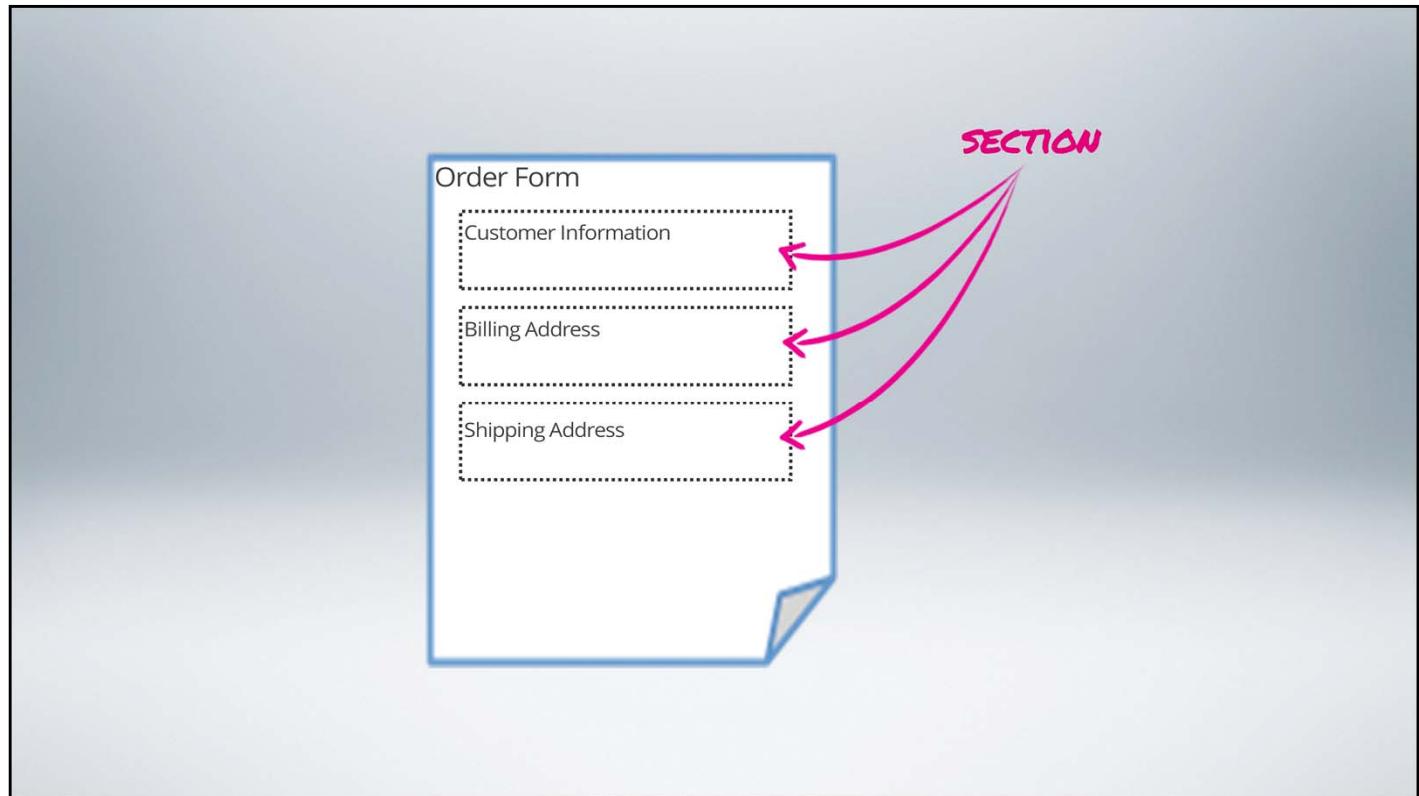
There is also the “Designer Studio” portal, which is used by Business and System Architects – us – to design and modify case management applications. However, this particular portal is built and maintained by Pega as part of the product so there will never be a need for you to build UI components for it; unless, of course, you hire on at Pega as an engineer ;)

Finally, there are the actual “work forms,” which are used to create, update, or resolve cases.



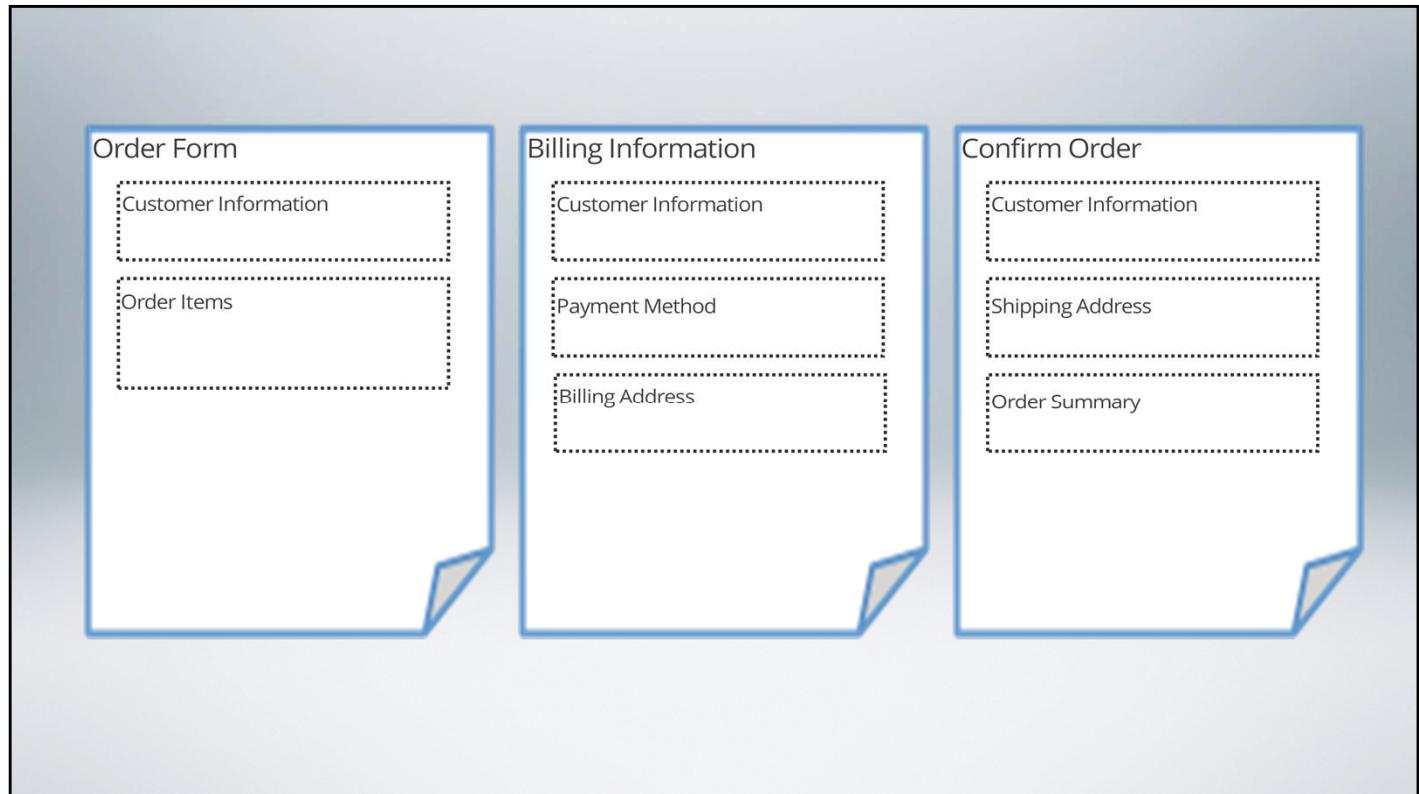
Contains the information a user needs to work on a case

In this lesson, let's focus our attention on the "work forms." We don't want to imply the case manager portal is not important; it's just that customizing the case manager portal is a separate development effort – and outside the scope of this course.

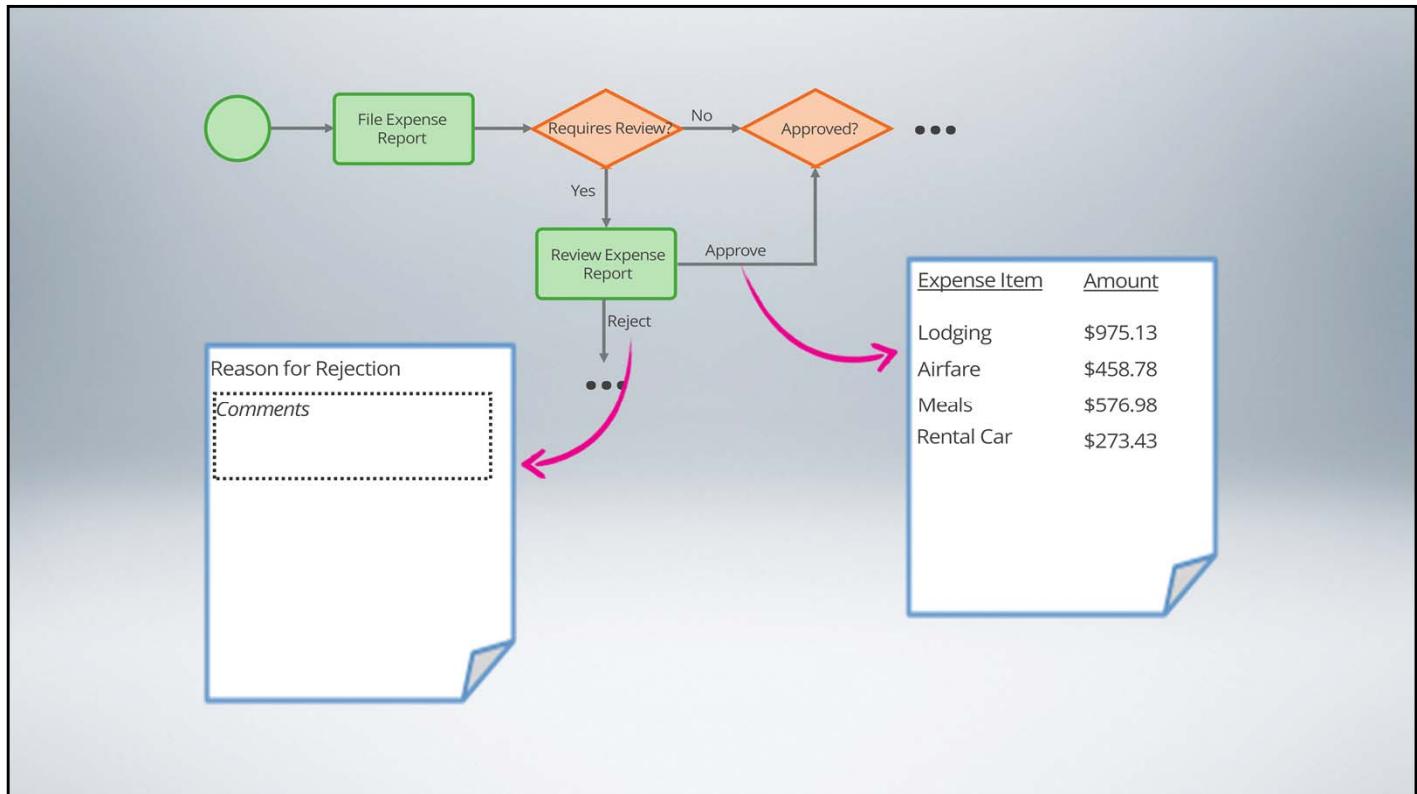


There are three core components used to present the work forms to the business users. Let's start with the inner-most component – the section. A *section* contains content for a portion of a work form.

A section should contain related bits of data, such as a customer's personal information, maybe a billing address, and maybe a shipping address.

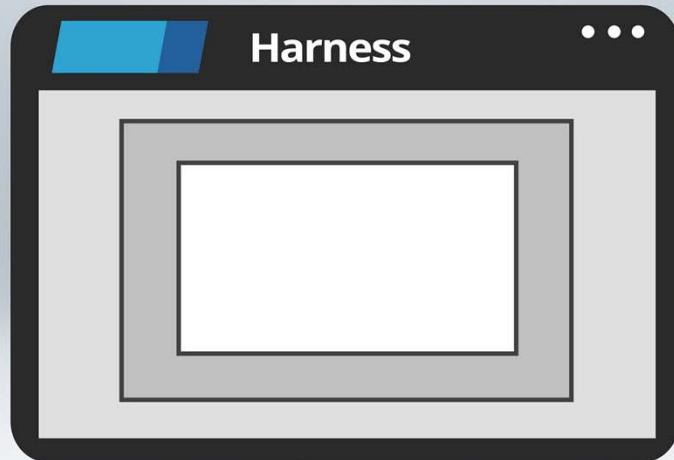


The goal when creating sections is to build, and organize, them in such a way that they can be reassembled in any combination based on the information a user needs to complete some portion of a task.

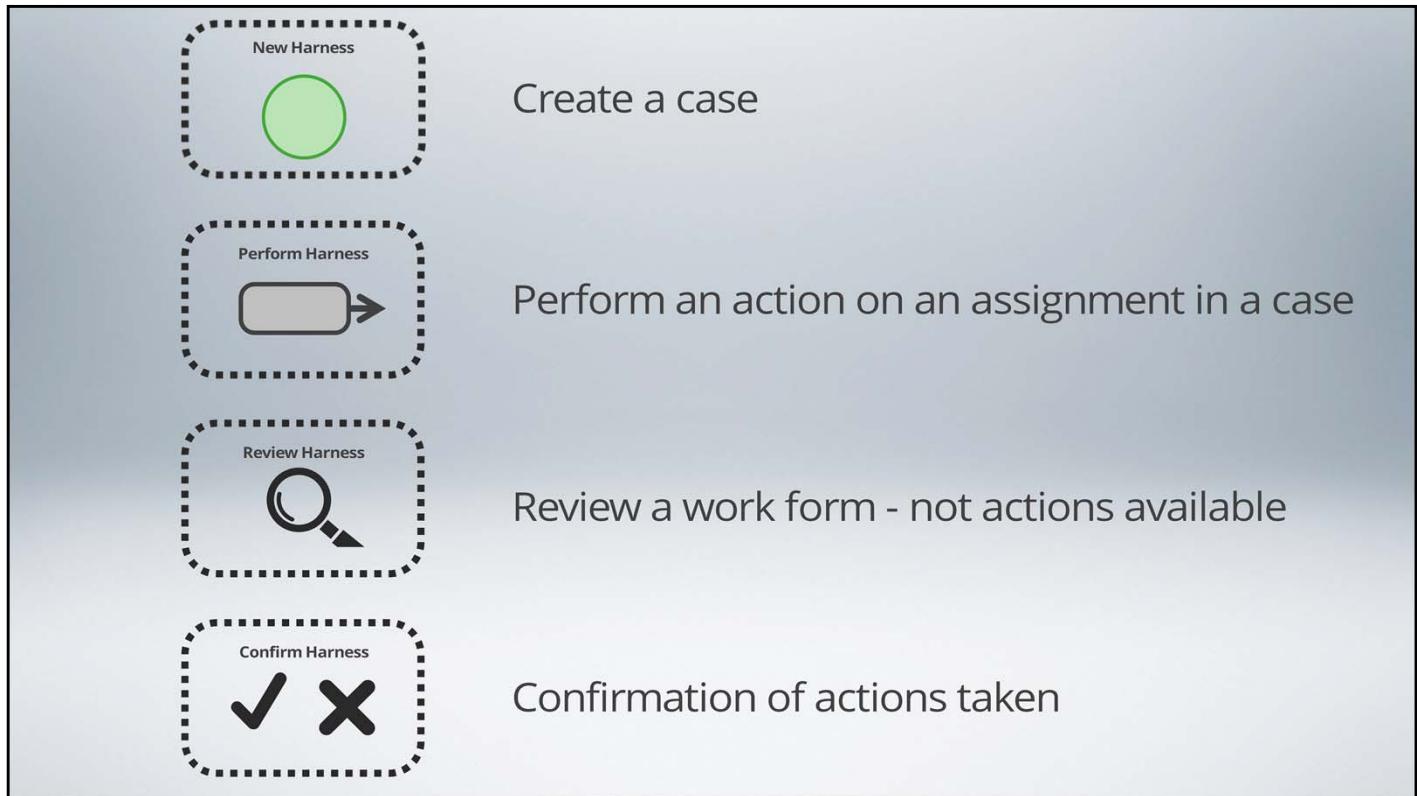


That takes us to flow actions. A flow action does two things. One, it defines a specific action a user can take while processing a case.

Two, a flow action presents the relevant portion of a work form – a “section” - to the end user for each specific action.



Finally, there is the “Harness.” The harness sets the context for processing an action - it defines the behavior of a work form.



There are four types of harnesses: a “New” type, which is used when creating a case.

There is a “Perform” type harness, which allows a user to select a flow action to complete an assignment

There is a “Review” type harness, which presents a work form in read-only mode.

And, finally, there is a “Confirm” type harness, which presents a pre-defined confirmation screen to the user.

Harnesses are tightly integrated with the shapes in a flow diagram and are set by default.

## Create Intent-Driven user interface



Review Shopping Cart

Items to Buy    Quantity



Price

Confirm Billing Information

Payment Method

Billing Address

Pega strongly encourages creating intent-driven User Interfaces.

An intent-driven user interface is a screen where the user has no problem understanding what they need to do. For example, the form used to review a shopping cart for a purchase should only have the relevant information inside it like the items to purchase, how many and the price of each item.

The form should not contain information for things like billing address; these data elements are extraneous for reviewing a shopping cart and could distract the user from the task at hand.

## Create Model-Driven user interface



Review Shopping Cart

<u>Items to Buy</u>	<u>Price</u>	<u>Quantity</u>

Confirm Billing Information

<u>Payment Method</u>

Billing Address

The user interface should also be model-driven. What this means is that user interfaces are tightly coupled with the process.

It is the process that we define that determines which user interface is rendered at each step in the process.

Everything we do UI-wise is connected to our core value of model-driven development.  
Remember, what we model is what we execute.

## Model-Driven approach allows us to keep up with rapidly changing UI needs



UI is connected to model driven development

Auto-generated using controls and no coding

UI dynamically incorporates process steps & business rules

A Model driven approach allows us to keep up with rapidly changing UI needs.

A model-driven approach has a number of benefits, including speedier application development, a UI that is contextually sensitive to the type of business process we are mapping out, as well as a UI that responds rapidly to changes in business rules.

If we divorce the creation of the UI from the creation of the business processes we lose “Build for Change®” at the UI layer.

## Exercise: Building the Screens for Implementing the Assignment



# Best Practices for Designing the User Interface

In this lesson, we study the importance of, and best practices for, UI design.

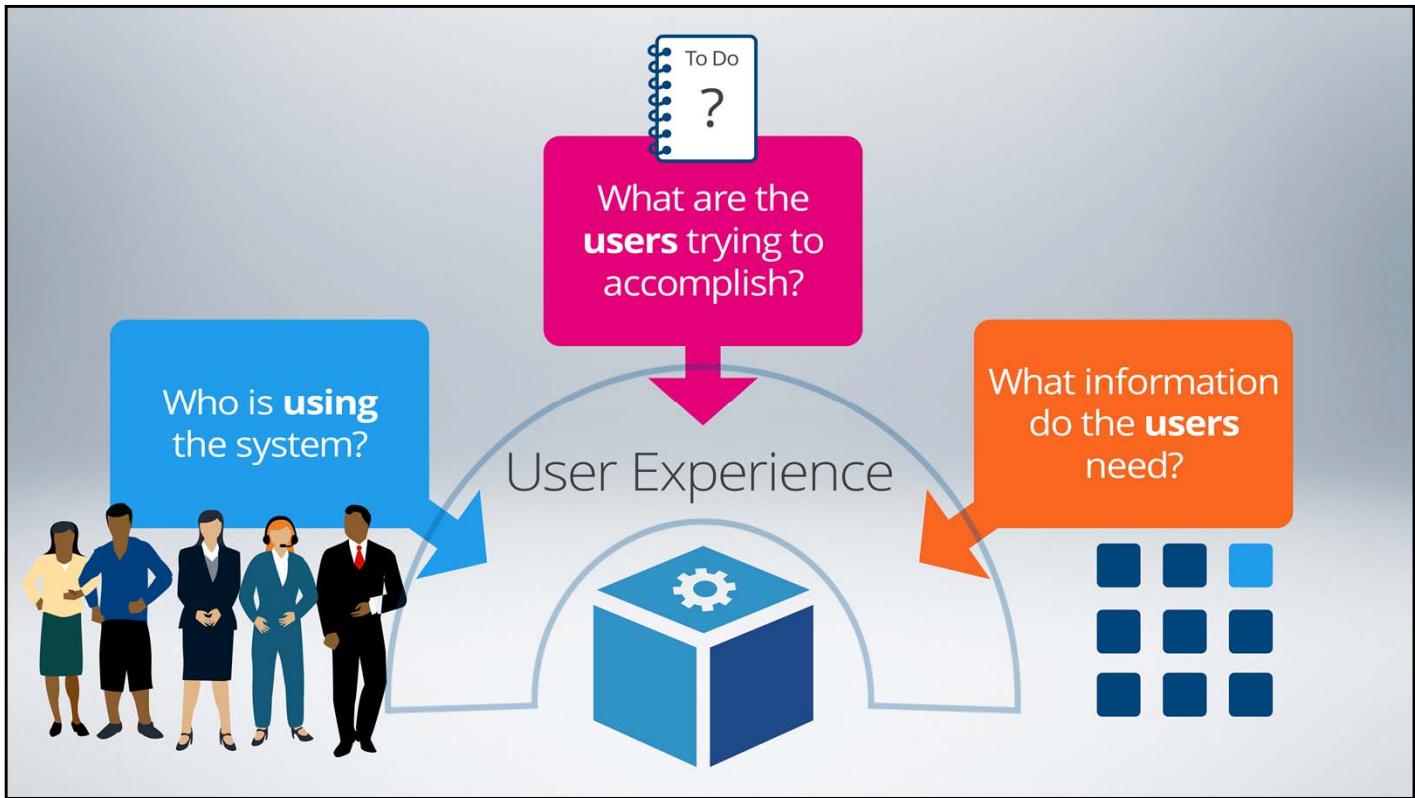
At the end of this lesson, you should be able to:

- Answer the question "Why is the user experience important?"
- Answer the question "What is usability?"
- State at least three best practices for designing user interfaces



The user interface is the most broadly visible aspect of your application.

Its design affects a users' productivity, acceptance, accuracy, and satisfaction and so is a critical factor in the success of an implementation.

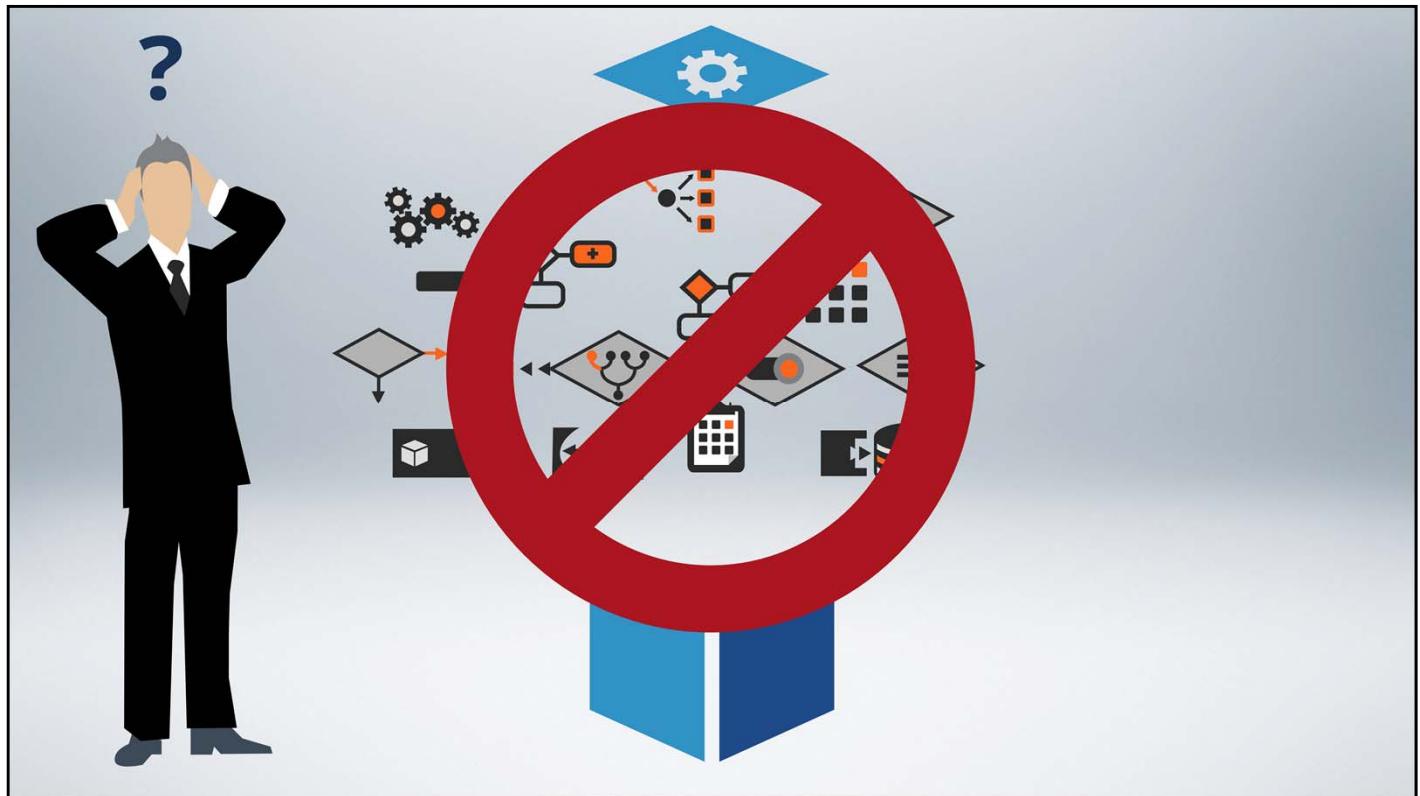


Instead of focusing solely on the technology or the process, a great case lifecycle management application development effort should also focus on the user. User Experience should focus on three main areas:

Who is using the system?

What are users trying to accomplish through the system?

What information do users need?



A good UI should never expose the complexity of the application to a user so that they have to struggle to get their jobs done.



When used by employees, time wasted being lost in application is money wasted paying them to be at work without getting work done.

May eventually lose interest and leave.

If the system is being used by employees, the time they waste being lost in the application is money wasted by paying them to be at work without getting work done. They might eventually lose interest in working for the organization.



If used directly by customers, spending too much time to figure out what and how to do things increases their frustration.

May eventually look and move elsewhere, probably to competitors.

If, on the other hand, the system is being used directly by customers through the web, spending too much time trying to figure out what to do and how to do it quickly increases their level of frustration. When this happens, you can be sure that they will eventually look and move elsewhere, probably to the competitors.

**Usability** = Key attribute to make an application useful



Usability is a key attribute needed to make an application useful. It matters little that an application can hypothetically do what users want when they can't make it happen because the user interface is too difficult.

## The User Interface is the entire system from a user's perspective



The user interface, while it's not the entire system in a technical sense, is indeed the entire system from a user's perspective – it is the window into the application but it is also a layer that protects users from the technical complexity of the entire application. From the beginning of your project, try to set some standards or define conventions regarding the user interface.

# Key Principles of UI Design and Usability

## Set some standards or define conventions

[Submit](#)[Submit](#)

- Actions, messages, icons, screen layouts and navigation patterns should be consistent throughout the application

## Use words, phrases and concepts familiar to end-users

- Simple, concise and very clear
- Follow natural and logical order

## Use words rather than icons

### Make icons meaningful

- Design for usability



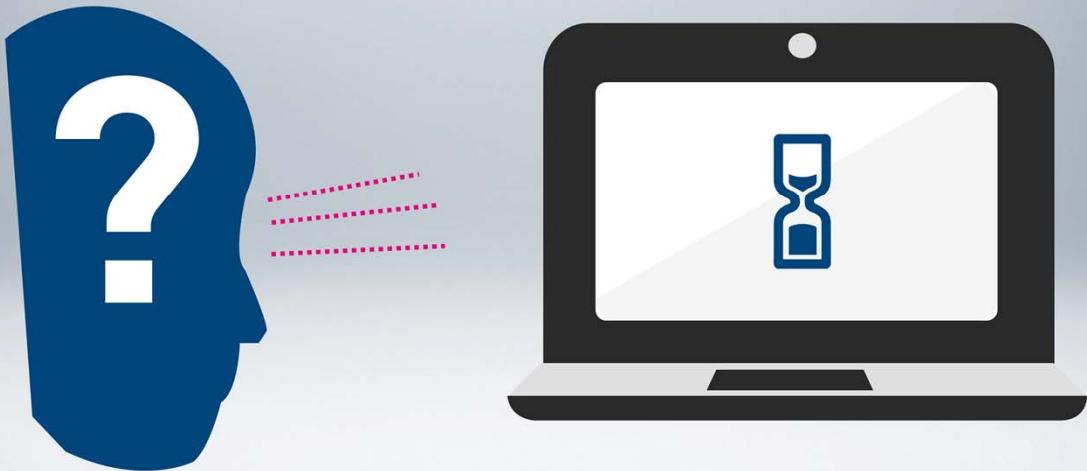
First and foremost, be consistent Actions, messages, icons, screen layouts, and navigation patterns should be consistent throughout the application.

As much as possible, try to use words, phrases and concepts familiar to the end users rather than system technical terms. Keep those words simple, concise and very clear. Make it easy by having information appear in a natural and logical order.

Even though words are better than icons, when using icons, make sure they are a meaningful and real-world visual counterpart like a trash can for delete.

These icons for example may look very cool but they may not mean anything to the end-users. It is not easy to know what they represent and what might happen if we click on them. Even though they are cool looking, they are not designed for better usability. Simply avoid such icons.

# Visibility of System Status



Another key principle is the visibility of system status. Users should not be left wondering what the system is doing.

For instance, suppose a user clicks on a button and nothing happens on the screen or the screen just goes blank. Without any sign or a message, the users may not know that the application is still up and running in the background but is just slow.

Always keep users informed about what is going on through appropriate feedback such as an hour-glass, progress bar or other control.

# Recognition rather than recall

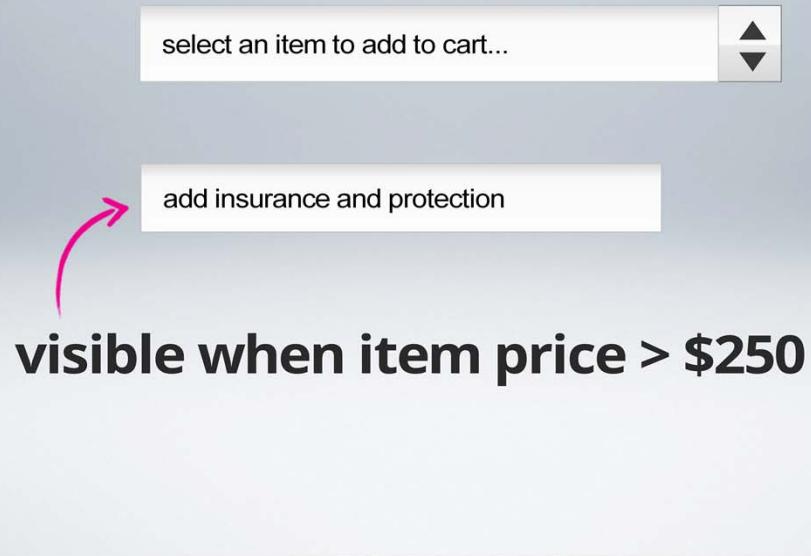


Recognition rather than recall is another key principle to try to enforce.

The basis of this principle is NOT to have users remember, or memorize, information from one part of the interaction to another.

So, make actions to take and instructions for those actions visible or easily retrievable whenever appropriate.

# Aesthetic and minimalist design

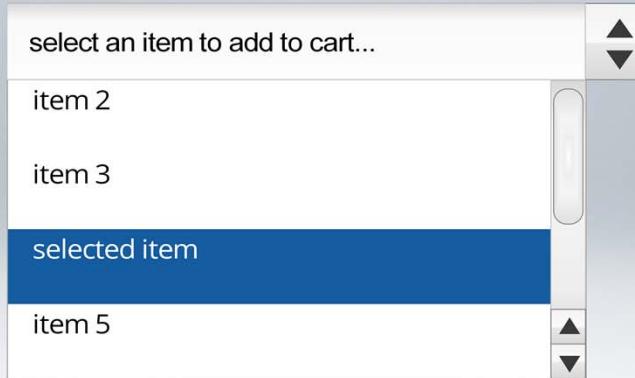


However, aesthetic and minimalist a design may be it should also be enforced by making interactions contain information that is relevant.

Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

So, use the visible when feature whenever appropriate so that users are not inundated with information until needed.

# Aesthetic and minimalist design



**USE DROPODOWN IF LIST CONTAINS > 10 ITEMS  
AVOID DROPODOWN IF LIST CONTAINS < 6 ITEMS**

A drop-down is a common control used to display a list of items from which users select.

For better usability, always use a drop-down if the list contains more than 10 items. However, if there are less than six items, avoid using drop-downs. Other controls such as checkboxes may be more suitable.

# Primary action emphasis



select an item to add to cart...



**ADD TO  
CART**

**ADD TO  
WISHLIST**

**BUY AS  
A GIFT**

Whenever there is a likely primary action, use a bright and obvious button to emphasize it on the screen.

Use a verb and noun if possible when labeling your buttons and links.

Make sure the label is clear not something like "click here."

# Error prevention



**Are you sure you want to leave beneficiary blank?**

Error prevention is another key principle which calls for a careful design that prevents a problem from occurring in the first place.

Either eliminate error-prone conditions or check for them and present users with a confirmation message before they commit to the action. This is even better than good error messages.

# Error prevention



**Error: Beneficiary field cannot be blank.  
Please designate a beneficiary.**



Indeed, you cannot totally eliminate error messages from your application.

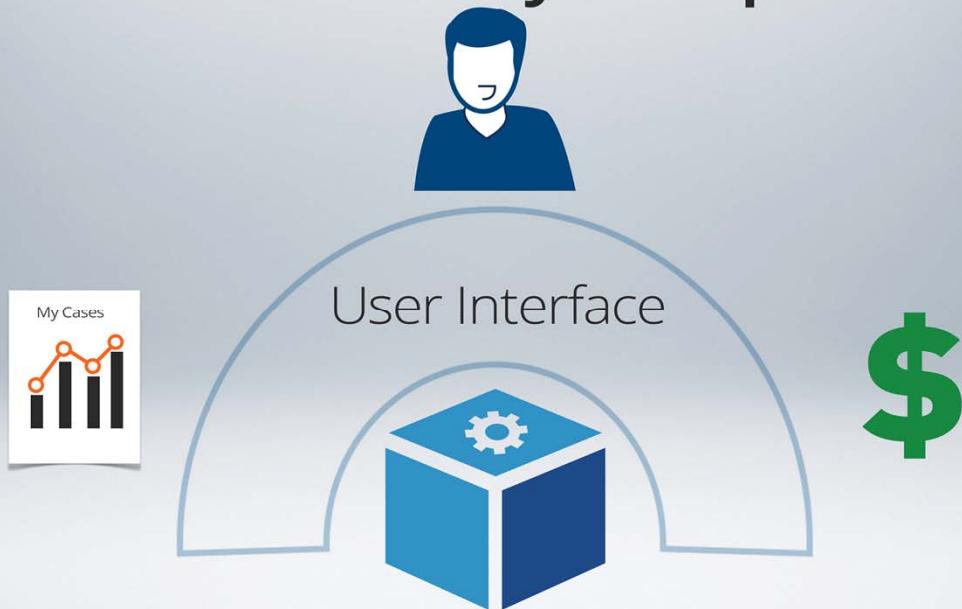
However, you can and must make every effort necessary to use plain language to express the error messages, clearly indicate the problem and precisely suggest a remedy.

This helps users recognize, diagnose and recover from errors.

Also, it is most likely that users will choose some application functions by mistake.

For such situations, support operations like “undo” to leave the unwanted state without having to go through an extended dialogue.

# Application usability is important for:



**think: Simple, think: Clean, think: Easy to understand**

This was a rather short lesson, but...we have covered quite a few UI design and usability best practices. Let's see if we can summarize them into a single frame.

We have discussed what usability is about and why it is so important.

Employee productivity, customer satisfaction and retention and ultimately the most efficient use of your organization's money relies on the level of effort you put on the usability of the system. Always remember that a good and powerful application is only as good as its user interface.

Keeping things simple, clean and easy to understand is your main road to usability greatness.

# Using Advanced User Interface Controls

In this lesson, we will explore best practices for choosing the most appropriate UI control, then look at some of Pega 7's advanced UI controls.

At the end of this lesson, you should be able to:

- Choose the most appropriate UI control
- Configure a Dropdown Control
- Configure an Autocomplete Control
- Configure Radio Buttons
- Configure Modal Dialogs and Overlays in controls
- Use the UI Gallery

Element	Description	Examples
Check Boxes	Select one or more options from a set	Vehicle Options <input type="checkbox"/> Sunroof <input type="checkbox"/> Premium Audio <input type="checkbox"/> Parking Assist <input type="checkbox"/> Backup Camera <input type="checkbox"/> Mirror Memory <input type="checkbox"/> Trip Computer
Radio Buttons	Select only one option from a set	Vehicle Color <input checked="" type="radio"/> Pearl Grey <input type="radio"/> Blue Dazzle <input checked="" type="radio"/> Saddle Tan <input type="radio"/> Black
Drop Down Lists	Select only one option from a set	select an item... Home Phone Mobile <b>Email</b> Fax
Auto Complete	Select only one option from a large set	New New Yorker New York Square <b>New YorkTimes</b> New York and Company

User interface design – or..., UX - focuses on understanding what users might need to do and ensuring that the user interface has elements that are easy to access, easy to understand, and easy to use.

So, in no particular order....

If our business requirements specify users need to be able to make multiple selections, we would use check boxes.

Checkboxes allow users to select one or more options from a set. It is usually best to present checkboxes in a vertical list. More than one column is acceptable as well if the list is long enough that it might require scrolling or if a comparison of terms might be necessary.

If our business requirements specify the need to make a single selection from a set, we could use radio buttons. Depending on the number of options, radio buttons are usually best presented in one or more rows.

Dropdown lists also allow users to select only one item at a time, but are more compact allowing us to save space. Consider adding text to the field, such as 'Select a State' to help the user recognize the necessary action.

If the list of options to choose from is rather large, an AutoComplete control can help limit the number of available options that appear. As a user types one or a few characters, a filtered list of matching text values appears below the input field and users can select a value from the list.

Users have become familiar with interface elements acting in a certain way, so try to be consistent and predictable in your choices and their layout. Doing so helps with task completion, efficiency, and satisfaction.

## Demo



## Configure a Dropdown Control

Use a dropdown control to present users with a list of items, from which a single item can be selected. In the Purchase Application we use a dropdown to select the program for a purchase request. Let's have a look at how the Program dropdown is configured. The General tab lets us specify the associated property, whether or not to use the property label, the format for the label.

and a default value. We can also control the visibility, enabled state and whether or not this is required. Select Include placeholder and provide the text to be displayed in the dropdown if the property has no value. A dropdown can be sourced in four different ways. The configuration differs slightly depending on how we source the dropdown. We can choose As defined on property if we have defined the option on the property rule form.

There are five ways to define the option list on the property. Use the method most suitable for the application. It is also possible to source the dropdown from a data page. We need to specify the name of the data page. The property we entered for the value is also taken as the default for the display text. This can be overridden if we want to display text that is more user friendly, but keep the system friendly value in the clipboard.

We can also enter a property to be displayed as a tooltip when users hover over the item in the list. We can categorize items within the dropdown list by grouping items by a property. We could for example group by cost center code. A dropdown can also be sourced from a clipboard property. The clipboard page must be identified on the Pages & Classes tab. It is possible to specify a data transform or activity that is run before the dropdown is populated.

The rest of the properties are the same for data pages. Lastly we can source the data page from a report

definition. Specify the “Applies To” class of the Report definition, then select the Report definition from which to source the dropdown. The rest of the properties are the same of for data pages and clipboard page. Down here at the very bottom, we can configure the Load behavior. There are three choices available. To load when the screen renders, to defer loading until the user hovers over the dropdown, and to defer the load until all the other UI elements are rendered.

On the Presentation tab, we can specify the edit and read only settings, whether to use auto or fixed sizing, a tooltip if desired and the style of the dropdown. Using styles defined in the skin allows us to avoid specifying inline styles. The last tab is used to specify actions.

## Demo



### Configure an Autocomplete Control

Use an Autocomplete control to allow users to select a value from a possibly large set of text values based upon characters they type. In the Purchase Application we use an autocomplete to select the user for whom the purchase request is for. The user must type at least one or a few characters to activate the control.

Let's have a look at how the Requested For autocomplete is configured. The upper part of the General tab for autocomplete looks just as dropdown. An autocomplete can be sourced and configured the same way as a dropdown. The autocomplete search result configuration is independent of how the list is sourced. At least one data source property has to be configured.

The mandatory property is always shown and is the one the associated property of the autocomplete is set to. It can but does not have to be used for the search.

Additional properties can be added. These can optionally be shown and used in the search. It is possible to use the value of a data source property to set a specified property. Select Display best bets to add a Best Bets heading displaying the results that application users select most often. We can specify a custom best bets property. It is possible to categorize results in the list. Select Categorize search results and provide a Category property. We can use Max results displayed to limit the size of the returned list. We can also specify the number of search characters the user has to enter before the search is triggered. Select Highlight match to highlight the search string in the displayed result. Match starting of string specifies whether or not the search is on the beginning only or anywhere in the field.

Let's have a look at the Presentation tab. The Presentation tab contains the option to specify a placeholder in addition to the same standard options that were available for the dropdown.

## Demo



## Configure Radio Buttons

Use radio buttons to display a small number of options, from which users can select one. The advantage of radio buttons over dropdown and autocomplete is that all options are visible at all times. Here we use radio buttons to select the urgency for the purchase request.

Let's have a look at how the Urgency radio buttons are configured. The General tab for radio button looks just as dropdown and autocomplete. Radio buttons can be sourced in the same four ways. On the Presentation tab, we can specify if we want to display the radio buttons vertically or horizontally and how many items we want in each column or row.

## Demo



### Configure Modal Dialogs and Overlays in Controls

A modal dialog is a form to be filled in or a statement to be acknowledged that pops up in its own window on top of the application. The user cannot continue in the application without processing or dismissing the modal dialog. In the Purchase Application we use a modal dialog to enter new vendors.

An Overlay is similar but overlays the current window instead, allowing users to dismiss it by clicking outside the overlay area. Overlays are commonly used to show detailed information.

Modal Dialogs and Overlays can be launched from controls. Let's have a look at how the add vendor modal dialog is configured.

The icon control is used in this case, but a modal dialog can be launched from any control.

Let's have a look at the Actions tab. A click event has been defined for the icon. A Local action is called on click. The local action called is AddVendor. The target is set to Modal Dialog in this case.

However, we could show the local action in an Overlay instead if we wanted, or in the current window by selecting Replace Current.

## Demo



## Using the UI Gallery

Pega 7 provides numerous user interface components. Use the UI Gallery to get an overview of available components. Launch the UI Gallery by selecting DesignerStudio > User Interface > UI Gallery.

The UI Gallery has two parts. The first is the Showcase, which contains some samples of complete User Interfaces. The second is the Available Components, which contains samples of individual controls. These are further grouped into the four categories shown here; Components, Layouts & Containers, Tables & Grids and Samples & Combinations.

To examine a control, we just click on the name. Let's have a look at the dropdown. The main part of the screen shows us samples of this control in use. On the right other controls that share similar functionality to the one we're currently examining are shown. Use Navigate to jump directly to any other control in the gallery. Or, return to the main UI Gallery.

The View design-time configuration link opens the section rule so that we can examine how these examples have been configured.

## Exercise: Working with modal dialogs



# Managing Data for Selectable List Controls

In this lesson, we explore the options available for sourcing list controls

At the end of this lesson, you should be able to:

- Identify two of the most common ways to manage data for selection lists
- Configure a drop down list to use a data page as the source
- Source a property from a data page

The screenshot shows the Pega Platform's property rule form for an 'Expense Report' object. The 'Expense Type' field is configured as a dropdown list ('pxDropdown'). The 'Table Type' is set to 'Local List', and the 'Table Values' section displays a static list of expense types: Airfare, Car, Lodging, Meals, and Misc. The 'Lodging' option is currently selected.

## Local Lists

- Useful when values rarely change
- Out-of-the-box lists available
- Can be labor intensive

There are several ways to source the data for selectable lists. Let's look at two of the most common options.

One way is to use a static – or...local list,

as defined on a property. Choose this option when the contents of the list rarely change, and are the same for all users.

To implement this option, edit the property rule form

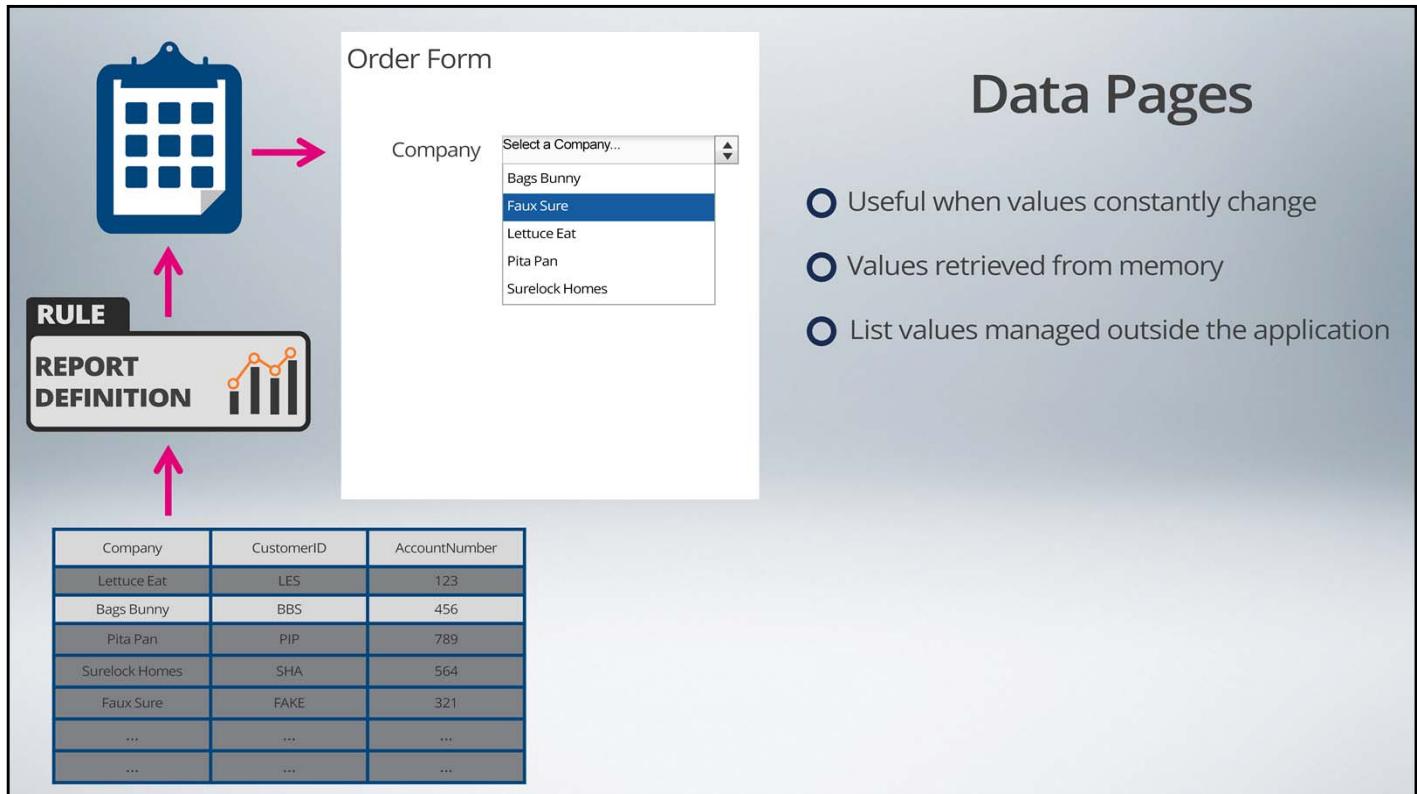
and set the UI Control option to pxDropdown or pxAutoComplete.

Then, set the Table Type to Local List in the Display and Validation section of the property.

There are some useful static lists, such as a state list, provided out of the box in Pega 7.

The downside is the property needs to be changed when the list values need to be changed. This can be labor intensive,

especially if it is the only rule that requires changing. A full development-to-production cycle is not something we would want to do for a single rule change.



## Data Pages

- Useful when values constantly change
- Values retrieved from memory
- List values managed outside the application

Another, very common approach, is to maintain the list values in a database table.

This is useful when the values in the list are subject to frequent change.

To implement this option, we create a data page.

This data page can be sourced using a report definition.

When the UI is accessed, the values are retrieved directly from the data page.

And..., changes to the values in the database table do not impact our application.

## Demo



### Source a Dropdown from a Data Page

We want to present the user with a dropdown containing the list of plans to choose from for each benefit type on the select benefits screen.

The plans are maintained in an external database table, which we mapped using the Database Table Class Mapping tool in a previous lesson. We want to source the dropdown from this table.

For this purpose we need a data page that provides us with the list of plans.

The structure is List and the list entries are of type SAE-HRSvcs-Data-HRPlan, which is the mapped class.

The data page takes the benefit type as a parameter allowing us to return a list of plans for a specific benefit type, such as medical, dental or vision.

We use a Report Definition to retrieve the data from the external table.

The benefits type parameter is passed into the report definition.

The report definition is in class SAE-HRSvcs-Data-HRPlan class, which is the class of the returned list entries.

The type is defined as an input parameter.

The report definition returns the necessary columns and filters the results on the type.

The section for selecting the plan is also in the SAE-HRSvcs-Data-HRPlan class and is reused across all benefit types.

The section takes the benefit type as an input parameter.

Let's have a look at the configuration for the medical benefits dropdown.

The list source type is data page.

Our data page D\_HRPlanList is used.

The benefits type parameter is set to the parameter passed in to the section.

We want to store the ID in the associated property, display the name in the dropdown and the description in

the tooltip.

## Demo



### Source a Property from a Data Page

We want to display the cost and description for the plan selected.

To accomplish this we have configured the data access for the benefit properties to refer to the D\_HRPlanLookup data page.

The selected value in the dropdown is associated with the ID field and is handed over to the data page as a parameter.

Let's have a look at the data page.

The structure is page and the object type is SAE-HRSvcs-Data-HRPlan.

We use a Lookup to retrieve the data from the external table.

The key to the record, in this case the ID, is provided as a parameter.

In this case the page property is of the same class as the external data, but it could be different. If it is different, we could map the data in the response data transform.

We have configured an action set on the dropdown to refresh the section when the value changes.

The refresh causes the data to be fetched from the table and loaded in the data page which is referenced by the property.

## Exercise: Managing Data for Selectable List Controls

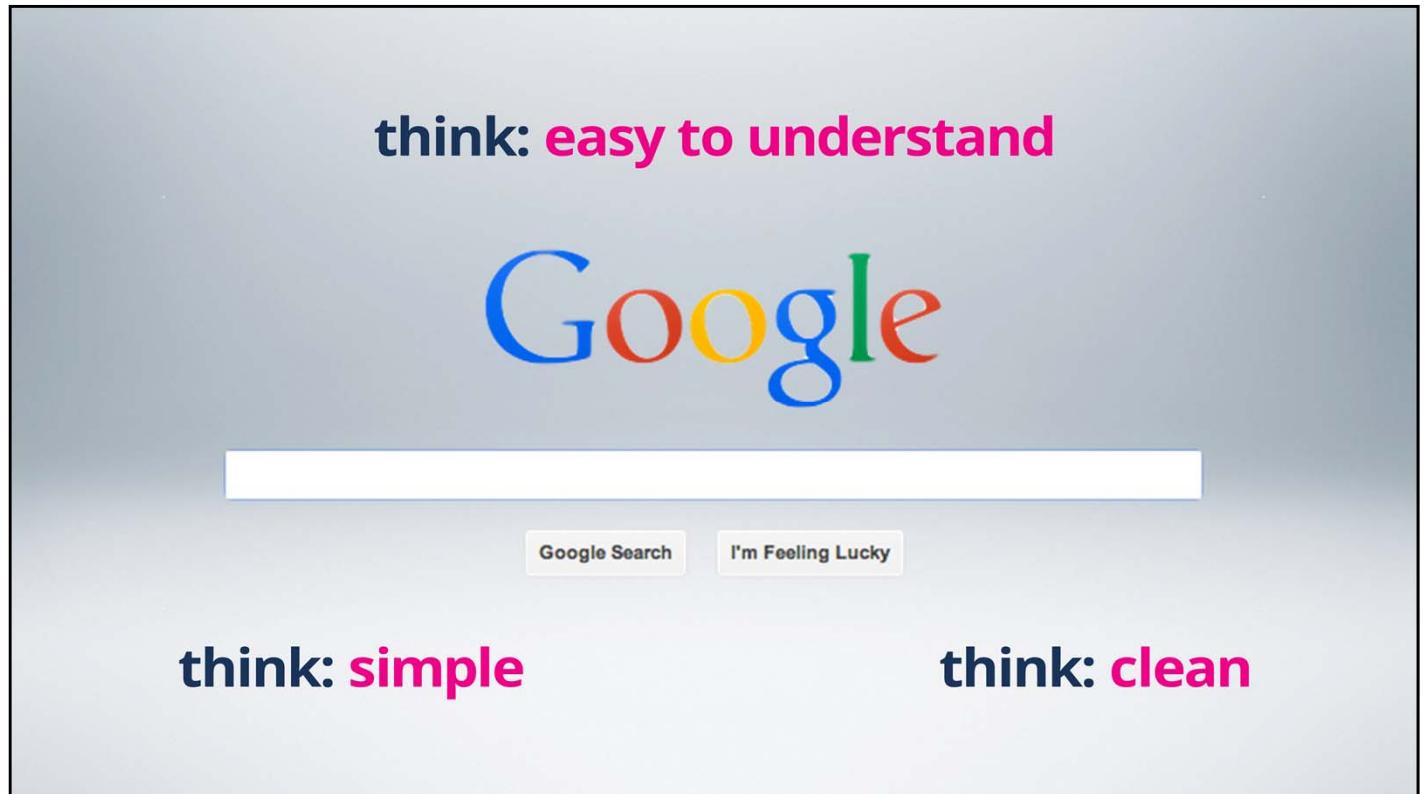


# Building Dynamic User Interfaces

In this lesson, we learn best practices for keeping complex user interfaces simple and clean.

At the end of this lesson, you should be able to:

- State at three reasons why dynamic UIs are useful
- Name two types of available dynamic events
- Dynamically hide and disable components



Dynamic user interfaces are inextricably tied to best practices for designing user interfaces.

The best UI is one that is immediately intuitive – like Google's home page.

You know exactly what to do - because there is nothing else left to do on any given screen.

Keeping a screen simple and focused is critical – and building dynamic UIs is a key component of that simplicity.

# Dynamic UIs

More functionality without overwhelming the end user



Building dynamic user interfaces  
allow us to combine more functionality on a single page.  
We start with collecting the most relevant information,  
then respond in real-time – dynamically - to what the end user is doing.

# Dynamic UIs

**Borrower Information**

First Name      Last Name

Marital Status     Single     Married

**Partner Information**

First Name      Last Name

- More functionality without overwhelming the end user
- Respond in Real-time to end user behavior
- Eliminate full-page refreshes
- Increase application performance
- More compelling, modern user experience

By making some parts of the screen dynamic, we can eliminate full-page refreshes on the browser.

When we do that, the "perceived performance" of the application improves.

And, at the end of the day, you will be creating a much more modern user experience and matching the kind of experience that users have daily on the web.

<b>Event</b> = something happens	<b>Action</b> = something changes
<b>Property-based event</b>  Order Total: <input type="text" value="£501.98"/>	Display message "Purchase orders limited to £500"
<b>User action event</b>  Marital Status: <input type="radio"/> Single <input checked="" type="radio"/> Married	Display partner information section

Conceptually, when we talk about “dynamic UI,” we’re really talking about using an event and action model in the browser-based application.

Today we’ll be talking mainly about two types of events, property-based events and user events.

Property-based events are fired either when a data value changes or specifically when a value meets a specific criteria.

A user event fires when an end-user actually takes some action on the page like selecting an option or clicking on a link.

## Demo



### Dynamically Hide And Disable Components

Let's have a look at how we can dynamically hide and disable fields. If we select a program in our purchase application the GL Entity and Cost Center fields are hidden since those are given by the selected program and can't be changed.

Let's have a look at the configuration.

There are five options for the visibility. We can hide the field if it is blank or zero or alternatively define a condition.

If we define an expression which can be run on the client the Run visibility condition on client checkbox is shown.

Alternatively, we can provide a when rule. In this case we use the when rule called IsNotProgramFunded.

If the condition is not run on the client we need to refresh the section if any of the fields that are part of the condition are updated. In this case the only property involved in the condition is the program name.

On the Actions tab we can define an event if the property is changed.

and the action is refresh this section.

But we could refresh another section or the harness if required.

We can also run a data transform or activity before the refresh.

If a program is selected we want to set the GL Entity and Cost Center to that of the program so we have a data transform that does that for us.

Disabling fields works very similar.

A condition can be specified in the same way.

This is how it will look for if the fields are disabled instead.

Instead of hiding the individual fields we can hide the entire layout.

In the visibility dropdown we have the option to specify a condition.

## Exercise: Creating Dynamic UIs

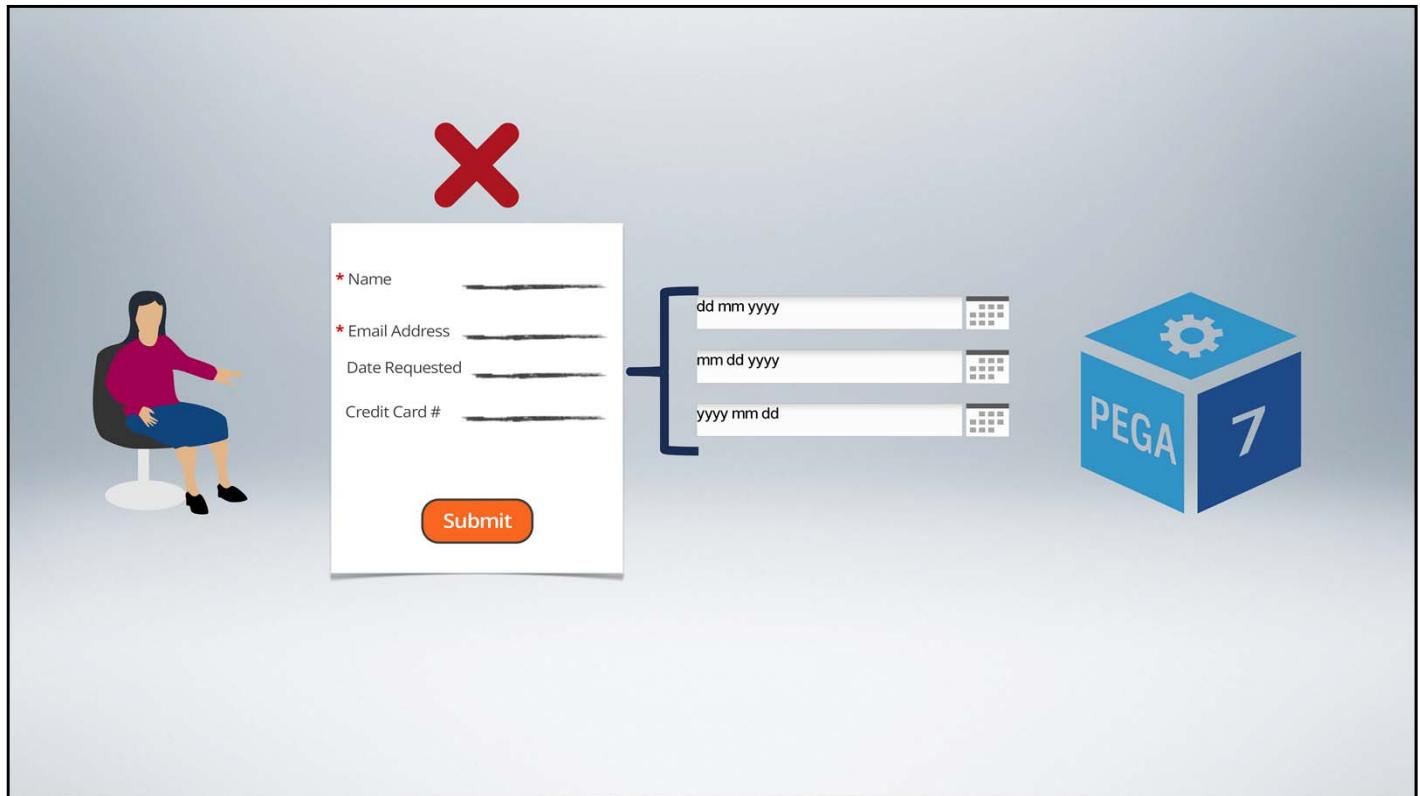


# Validating User Input

In this lesson, we learn the most appropriate methods for validating user input.

At the end of this lesson, you should be able to:

- Choose the most appropriate user validation method for user input
- Configure a Validate rule
- Customize Error Messages



Data validation is important in all web applications.

Before accepting data,  
we need to ensure that all required information is submitted,  
and that the submitted data conforms to our formats  
and business rules.

## Server-side & Client-side validation



In a broader context there are two types of validation. There is server-side validation; and there is client-side validation.

In this lesson we will be talking more about client-side validation.

## Validation using Form Controls

The screenshot shows the 'Cell Properties' dialog for a 'Text input' control. The 'General' tab is selected. The 'Label' field contains 'Last Name'. The 'Required' checkbox at the bottom is checked and highlighted with a red border.

There are several different types of client-side validation available in Pega 7. The most common – and probably the easiest to implement – is using the form controls.

For example, marking a form control as “required”

prevents the users from submitting the form until all fields using a control with this check box selected contains at least some data.

## Validation using Form Controls

Cell Properties

Date time [\(change\)](#)

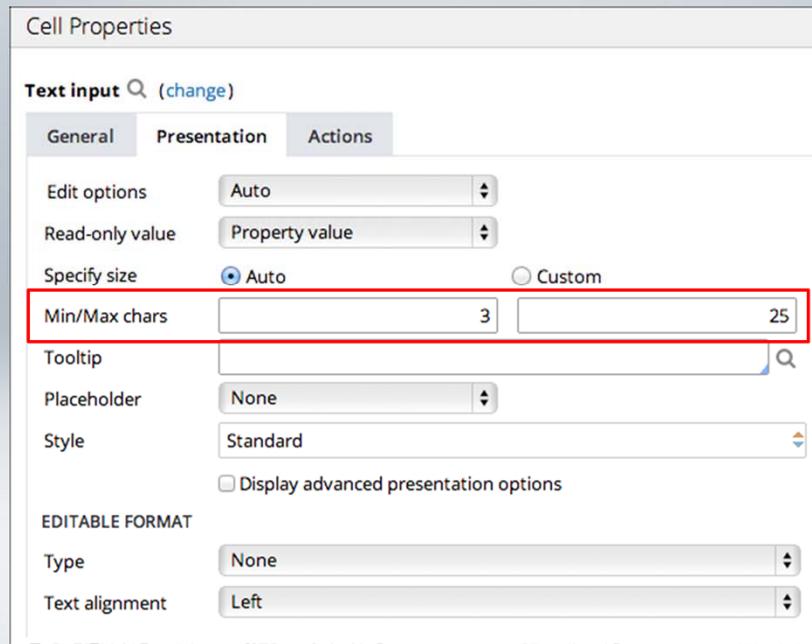
General		Presentation	Actions
Edit options	Auto		
Read-only value	Property value		
Specify size	<input type="radio"/> Auto	<input checked="" type="radio"/> Custom	
Width	100	%	
Placeholder	None		
Date/Time	Auto		
Display mode	Text Input + Calendar		
Allow text entry	<input type="radio"/> Yes	<input checked="" type="radio"/> No	
<input type="checkbox"/> Display advanced presentation options			
READ-ONLY FORMAT			
Type	Date/Time		
Date-time format	1/1/01 1:00 AM		

On the presentation tab of the cell properties for any form control, we can specify criteria that can be used to restrict users input, so they can only enter data in a way that we specify.

For example, when using a “Data/Time” control, we can select to not allow text entry.

Then, only the calendar can be used to select values – and as the name suggests – it does not allow users to manually enter the date.

## Validation using Form Controls



When using a "Text Input" control,

we can limit the length of the field. The field will not accept entries beyond the maximum number of characters allowed, and will not allow the form to be submitted unless at least the minimum number of characters is present.

We won't go through all of the controls in this lesson.

Type	Description
Date	For date with no time
DateTime	For date and time of day
Integer	Integer, may be zero, positive or negative
Decimal	Fixed decimal number, positive or negative
Text	Unedited, alpha-numeric text, which may contain spaces, tabs, and line break characters
TrueFalse	Boolean, two-valued

The point is to be aware that one option for data validation is using form controls. We can also use “data type” validation.

Remember, when we define a property we specify its type.

The data type of the property decides the valid values that it can accept. For Example, a date field can allow only date values,

and a decimal type will only allow numeric digits for which we can specify the fixed number of decimal places.

## Use Validate Rule for more complex validation

Credit score must be a valid FICO range

.CreditScore      IF value is < 300  
                  OR value is >850  
                  THEN display message:

Please enter a valid credit score. Credit scores should be between 300 and 850.

As useful as form control validations are, they are not always enough. If the business logic which governs valid input is more complex, say....for example a FICO credit score,

we can use a “Validate” rule. With a “Validate” rule, we can test property values against specified criteria.

If the values violate the specified condition, we can display a custom – and, hopefully, meaningful – message to our users.

Here as well, users cannot submit the form until the property value meets the specified criteria.

## Demo



### Configure a Validate Rule

We want to present the user with a dropdown containing the list of plans to choose from for each benefit type on the select benefits screen.

The plans are maintained in an external database table, which we mapped using the Database Table Class Mapping tool in a previous lesson. We want to source the dropdown from this table.

For this purpose we need a data page that provides us with the list of plans.

The structure is List and the list entries are of type SAE-HRSvcs-Data-HRPlan, which is the mapped class.

The data page takes the benefit type as a parameter allowing us to return a list of plans for a specific benefit type, such as medical, dental or vision.

We use a Report Definition to retrieve the data from the external table.

The benefits type parameter is passed into the report definition.

The report definition is in class SAE-HRSvcs-Data-HRPlan class, which is the class of the returned list entries.

The type is defined as an input parameter.

The report definition returns the necessary columns and filters the results on the type.

The section for selecting the plan is also in the SAE-HRSvcs-Data-HRPlan class and is reused across all benefit types.

The section takes the benefit type as an input parameter.

Let's have a look at the configuration for the medical benefits dropdown.

The list source type is data page.

Our data page D\_HRPlanList is used.

The benefits type parameter is set to the parameter passed in to the section.

We want to store the ID in the associated property, display the name in the dropdown and the description in

the tooltip.

## Demo



### Customize Error Messages

We want to present the user with a dropdown containing the list of plans to choose from for each benefit type on the select benefits screen.

The plans are maintained in an external database table, which we mapped using the Database Table Class Mapping tool in a previous lesson. We want to source the dropdown from this table.

For this purpose we need a data page that provides us with the list of plans.

The structure is List and the list entries are of type SAE-HRSvcs-Data-HRPlan, which is the mapped class.

The data page takes the benefit type as a parameter allowing us to return a list of plans for a specific benefit type, such as medical, dental or vision.

We use a Report Definition to retrieve the data from the external table.

The benefits type parameter is passed into the report definition.

The report definition is in class SAE-HRSvcs-Data-HRPlan class, which is the class of the returned list entries.

The type is defined as an input parameter.

The report definition returns the necessary columns and filters the results on the type.

The section for selecting the plan is also in the SAE-HRSvcs-Data-HRPlan class and is reused across all benefit types.

The section takes the benefit type as an input parameter.

Let's have a look at the configuration for the medical benefits dropdown.

The list source type is data page.

Our data page D\_HRPlanList is used.

The benefits type parameter is set to the parameter passed in to the section.

We want to store the ID in the associated property, display the name in the dropdown and the description in

the tooltip.

## Exercise: Validating Properties in a Repeating Layout



# Guardrails for Creating Engaging User Experiences

In this lesson you will explore the best practices of creating user experiences including practices to avoid and principles to follow.

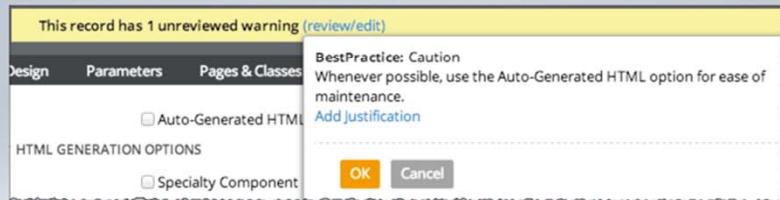
At the end of this lesson, you should be able to:

- Identify common guardrail warnings encountered when building user interfaces
- Identify motivations for establishing best practices as additional guardrails for designing user interfaces
- Provide a summary of the best practices for designing user interfaces



# Not Auto-Generated

**AVOID JUSTIFYING WARNING!**



- ⌚ Hand-coding is time consuming and prone to errors
- 🔨 Increases maintenance burden
- 🌐 Limits browser independence and compatibility

Let's look at two of the most common guardrail warnings you will encounter during UI design.

The “Not Auto-Generated” warning appears when the “Auto-generated HTML” checkbox on the HTML tab of a Section or Control is not selected.

Allowing auto-generation eliminates hand coding,  
which simplifies design and maintenance.

It can also makes cross-browser compatibility a bigger issue than it needs to be.

There are very few, if any, reasons for not using Auto-Generated HTML so avoid justifying this warning.



# Avoid Inline Styles

**AVOID JUSTIFYING WARNING**

This record has 1 unreviewed warning (review/edit)

Maintainability: Caution  
Whenever possible, avoid using inline style for ease of maintenance.  
Add justification

- Inconsistent** look and feel
- Changes are time consuming
- Binds content and presentation

Caution - Best Practice

Cell Properties

**Label (change)**

General    Presentation

Format    Standard

Display advanced presentation options

ADVANCED OPTIONS

Cell read-write style:

Cell read-only style:

Cell inline style: color:red;

Adding custom inline styles to individual cells in a layout violates guardrails.

Including styles inline at the cell level makes it difficult to maintain consistency over time.

And maintenance becomes much more difficult and time consuming.

By defining presentation attributes in the skin, we separate the content from its presentation. This ensures greater consistency and promotes reuse.

With the new skinning capabilities in Pega 7, there should never be a reason to justify this warning.



The user interface is the most broadly visible aspect of your application.

The user interface you design and build affects productivity, acceptance, accuracy and satisfaction. This makes UI design a critical factor in implementation success. Using best practices –

whether they are defined in this course or in your company's center of excellence - helps ensure the application we build provides the best end user experience.

These best practices for UI design become additional guardrails, which help ensure a consistent development effort and end-user experience. Let's end this lesson by taking one last look at the best practices for user interface design.

# User Interface Design

Put end users first

Be consistent

Customize Thoughtfully

Test usability constantly

These best practices for UI design become additional guardrails, which help ensure a consistent development effort and end-user experience. Let's end this lesson by taking one last look at the best practices for user interface design.

First, and probably most important, design with the end user in mind. Business and user requirements should be the driver, not opinion. Involve the users in the design process. They will feel invested in the application.

Be consistent. The easiest way to achieve this is to let Pega do the heavy lifting. Use auto-generated forms – always. Something as simple as using one primary navigation method can make all the difference.

Customize thoughtfully. Don't use UI features simply because they are available. When choosing to customize UI components, make sure they support intent-driven processes and enhance the ability of a user to complete a task.

Finally, build and test, build and test, build and test. Set measurable usability goals early and then design to meet those goals. Usability testing early lets us fix the issues before they become a problem.

## **Exercise: Creating an Engaging User Experience for Adding Dependents during Benefits Enrollment**

**Exercise: Creating an Engaging User Experience for Creating the Employee Record Exercise Verification**

**Exercise: Creating an Engaging User Experience for Selecting Benefits Exercise Verification**



## Module 07: Enforcing Business Policies

This lesson group includes the following lessons:

- Designing Business Rules for the Business User
- Enforcing Business Policies Using Service Levels
- Notifying Users from Within a Process
- Enforcing Business Policies Using Decision Rules
- Enforcing Data Relationships with Declarative Rules
- Guardrails for Enforcing Business Policies

# Designing Business Rules for the Business User

In this lesson, we learn to ask the right questions to be considered when designing and planning for rule maintenance by business users.

At the end of this lesson, you should be able to:

- State at least one benefit of delegating business rules
- Identify at least three types of business rules best suited for delegation to business users
- State at least three best practices when planning and designing for business rule maintenance by business users



Business policies change - sometimes suddenly - in response to internal and external factors. Pega 7 allows us to design and implement applications that respond to change with agility and efficiency.

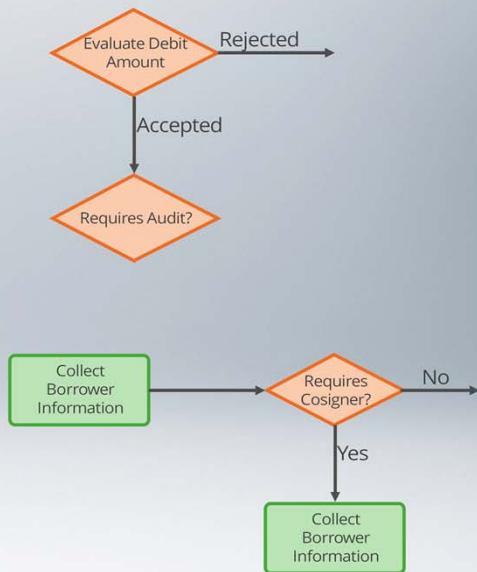
```

boolean bDoDecisionPath = (prop != null) ? true : false;
colVar1Set = prop.get("colVar1Set");
colVar2S = prop.get("colVar2S");
propString = prop.get("propString");
String str0 = new String(propString);

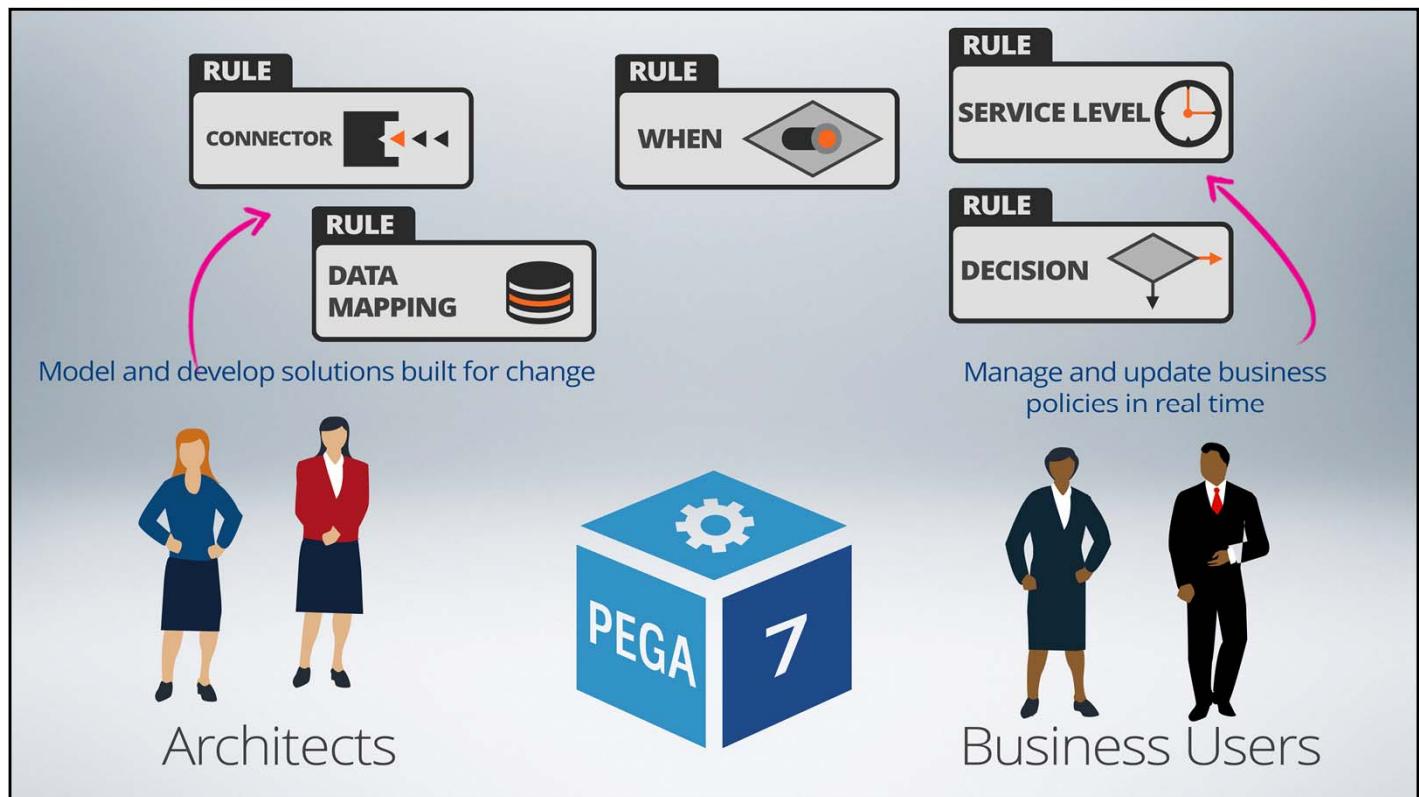
// loop through the rows and evaluate
// if the cells in a row are true
// then add to result
for (int i = 0; i < 1; i++) {
    for (int j = 0; j < 1000; j = j + 1000) {
        if (Blocks_circum0[i] == null) {
            //Conditions_circum0[i] = bDecisionPath, propString, runRuleProp);
            return;
        }
        if (Blocks_circum0[i].get(j) == null) {
            //Conditions_circum0[i].get(j) = "pyDefaultResult");
            Page runRulePage =
runRulePage.getPageValue(runRuleProp.getString("pyDecisionTable"));
            runRulePage.setPageValue("String("pyDecisionTable");
        }
    }
}

PerformProperty();
tools.putParamValue("param1", "sit", "sit");
//end of evaluateDecisionTable

```



By recording business policies in rules rather than in code – a model-driven approach - an application can provide a degree of modularity and transparency that can simplify maintenance.



However, to be truly effective, a Build for Change® strategy requires us to go beyond this basic benefit.

We – the application designers -  
can delegate responsibility for updating selected parts of each application  
to business users.

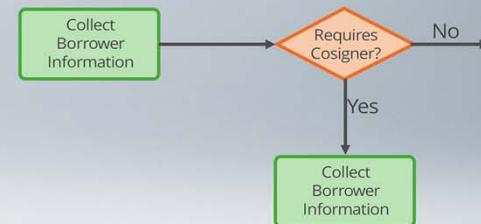
## Cosigner required when:

### TODAY

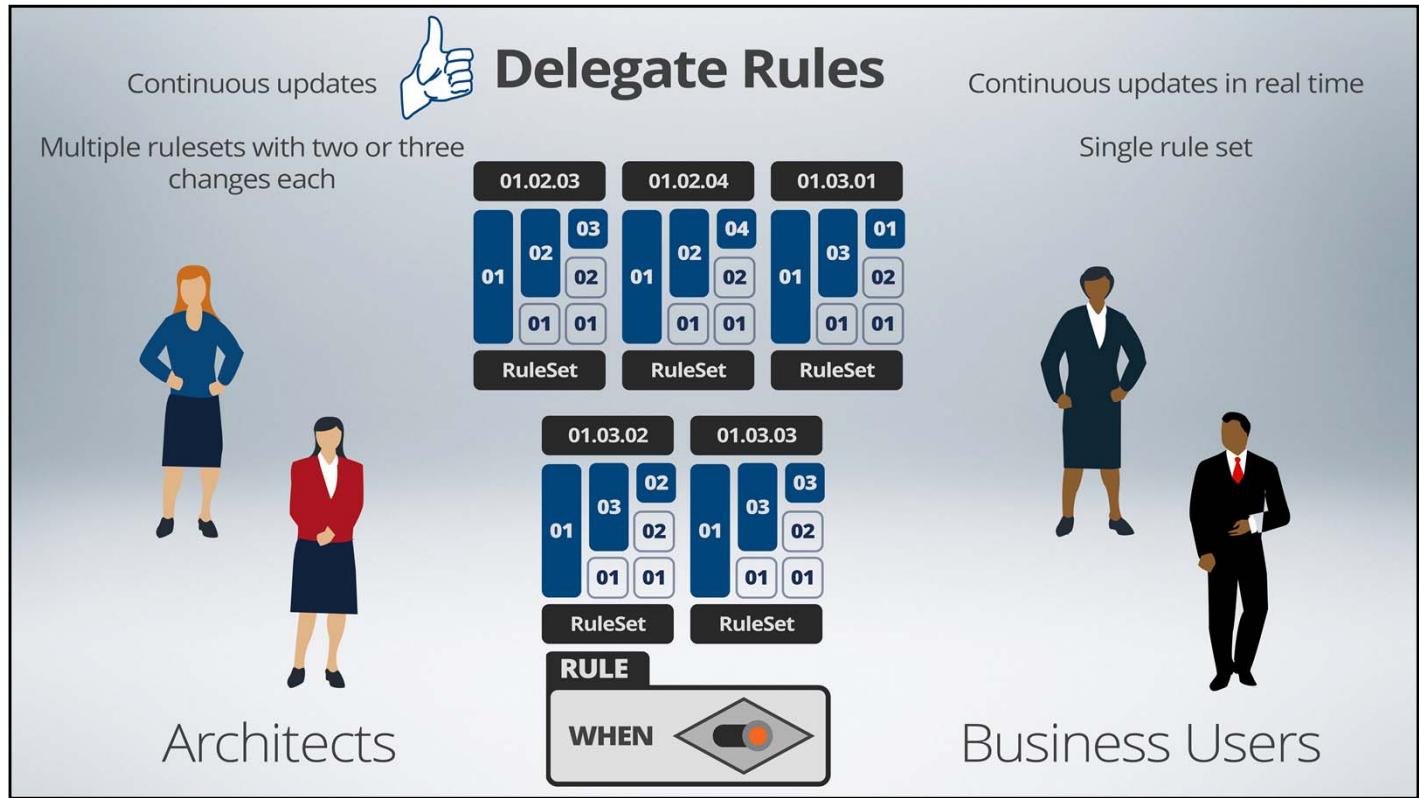
If borrower's income is < \$45,000  
and their credit score is < 649

### TOMORROW

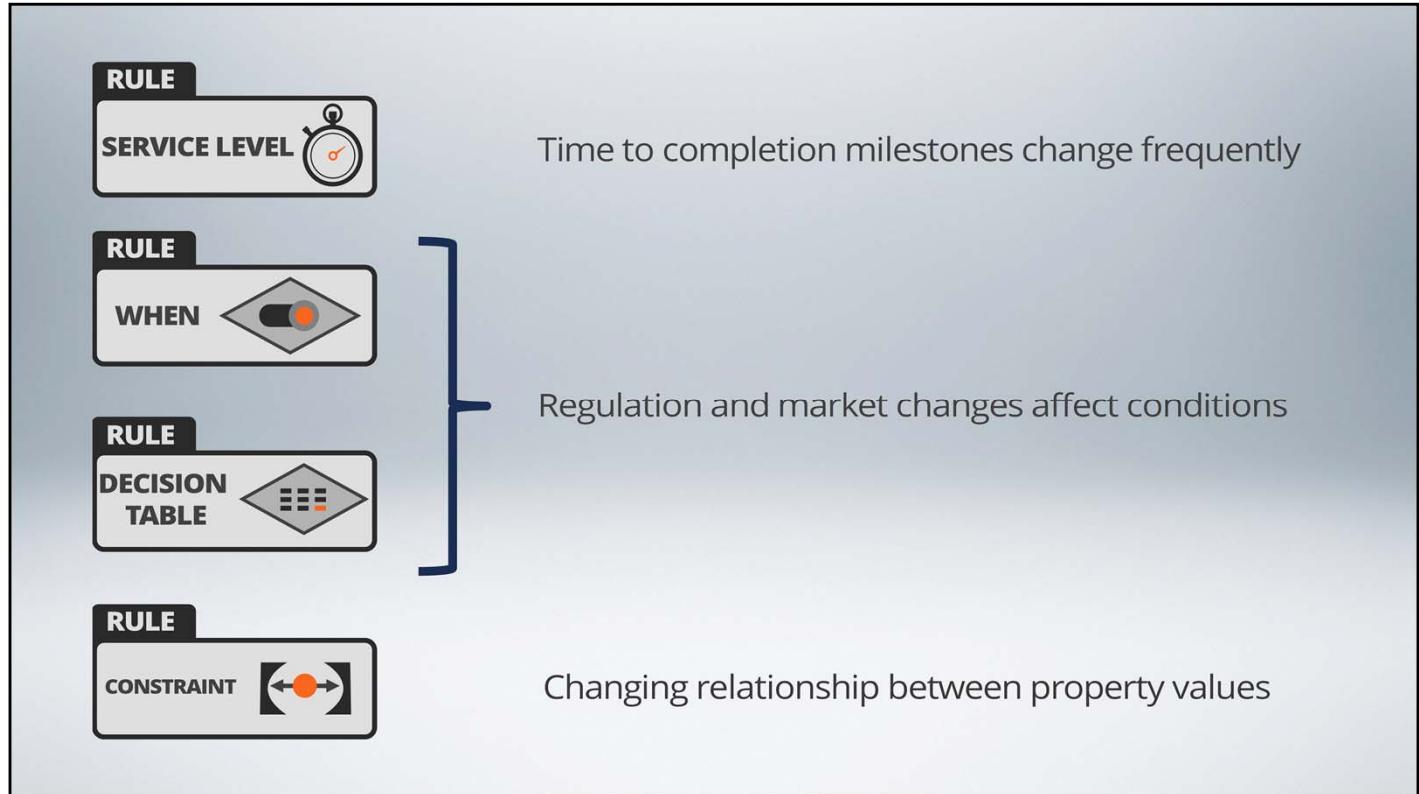
If borrower's income is < \$52,000  
and their credit score is < 655



Doing so can help promote an agile response to ever changing business conditions –  
what is true today  
can quickly change.



It can also help reduce the workload for architects of minor, low risk maintenance items and provide a degree of empowerment to those closest to the day-to-day operations. This shared responsibility goes a long way towards the success of our applications.



Although rules of any type can be delegated,

the rules most likely affected by constantly changing business policies are rules such as Service Levels.

With service levels, the time allocated to complete assignments may change based on seasonal adjustments or financial constraints.

Decision rules, such as “When” rules and “Decision Tables” are also subject to constant change.

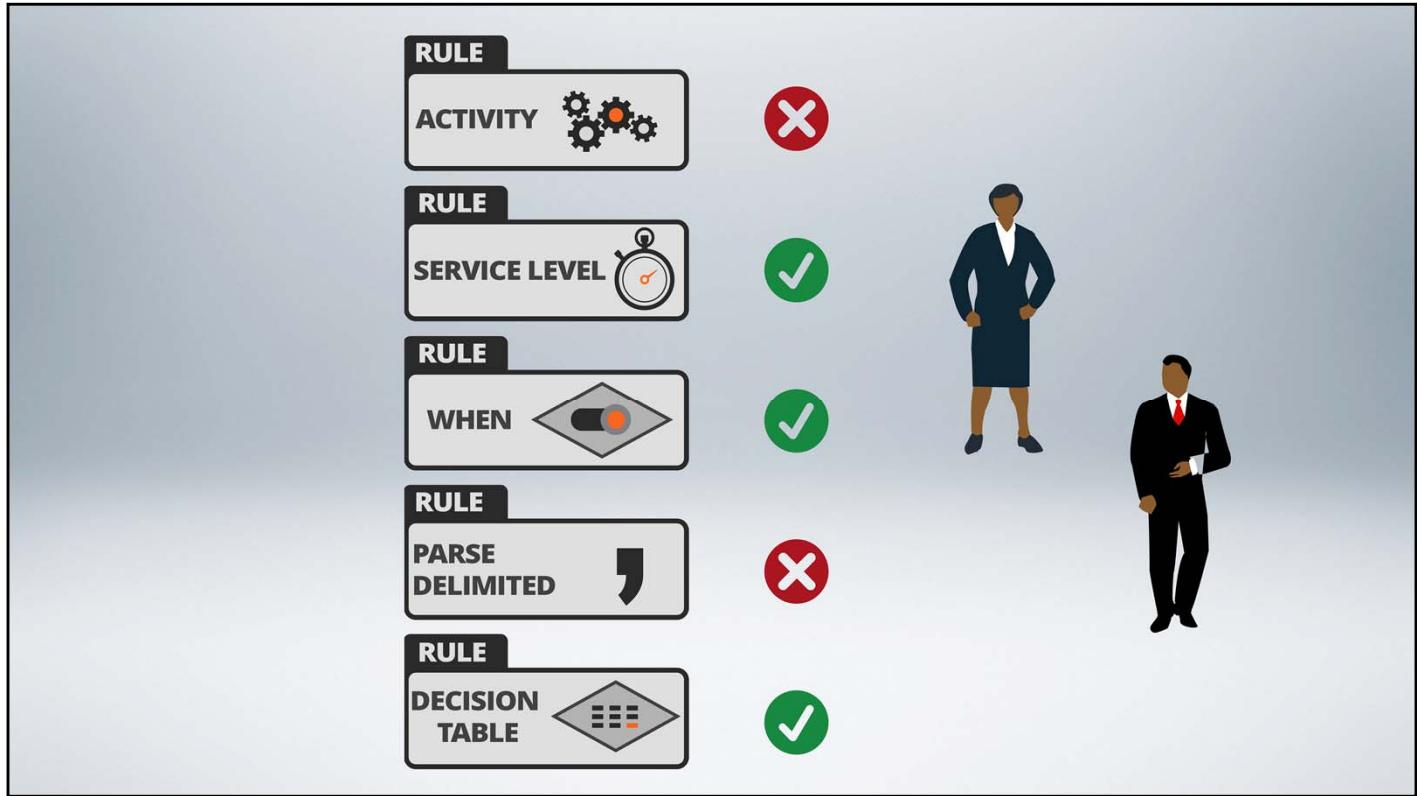
Market conditions and any number of regulations can instantly change a condition that was true yesterday, to false today.

Constraints might need regular updating as well.

For example, today an interest rate on a loan must be between six and nine percent. Tomorrow, it could change to between four and one half percent and eight and three quarters percent.



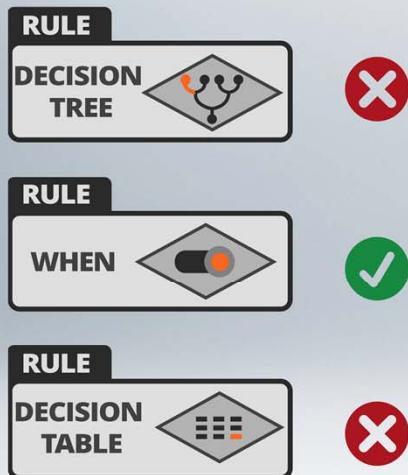
There are other rules, such as flows, map values, decision trees and declare expressions which could also be considered candidates for delegation.



Collaboration and understanding is key to successfully delegating business rules.

Work closely with the business users

and delegate those rules they are most likely to learn and understand.



*A cosigner is required if borrower's income is less than \$32,000 and their credit score is less than 635*

Enabling business users to maintain some of the rules in their application is a powerful feature. However, establishing guidelines and best practices is critical to successful rule delegation.

During design, identify which rules are useful to delegate to business users.

Remember, any rule can be delegated however, not every rule should be delegated. Work with the business users to determine what components of the business logic they want to maintain.

Once this is established, carefully consider which rule type best represents the logic.

## Choose short descriptions that are meaningful in the business context

RequiresCosigner



*Borrower's income is less than \$32,000 and their credit score is less than 635*

**RULE NAME + RULE TYPE + BUSINESS LOGIC**

Choose short descriptions that are meaningful in the business context.

A good test to determine if the rule name is in a business context is to build a sentence using the rule name, the rule type and the business logic.

## Provide appropriate Training and Documentation



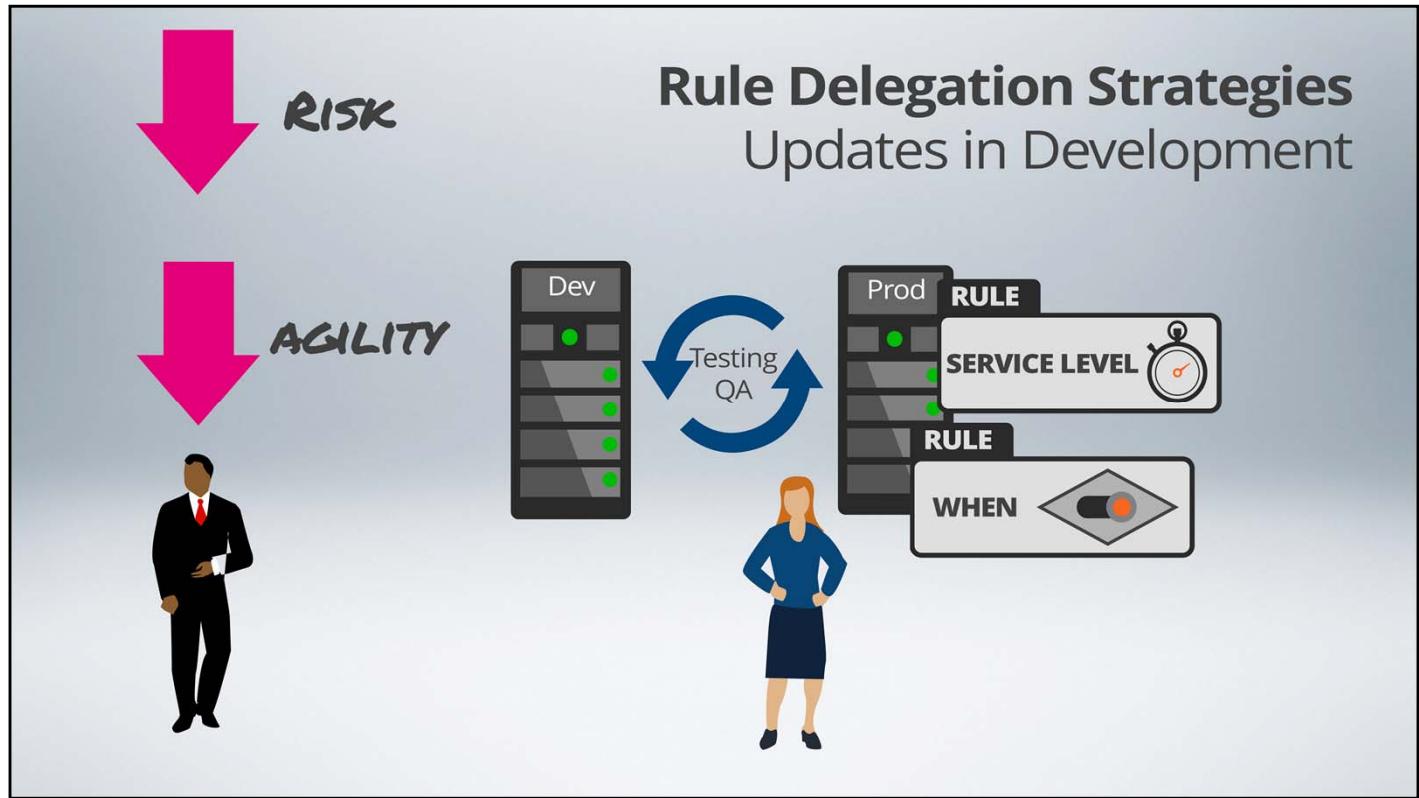
New Business Users



Business Users

Provide appropriate training and documentation. This may be the most important consideration when delegating rules to business users.

We need to recognize that new people may step into the role of updating delegated rules. Providing appropriate training and documentation can help make the rule delegation program self-sustaining.



Rule delegation can occur in any environment.

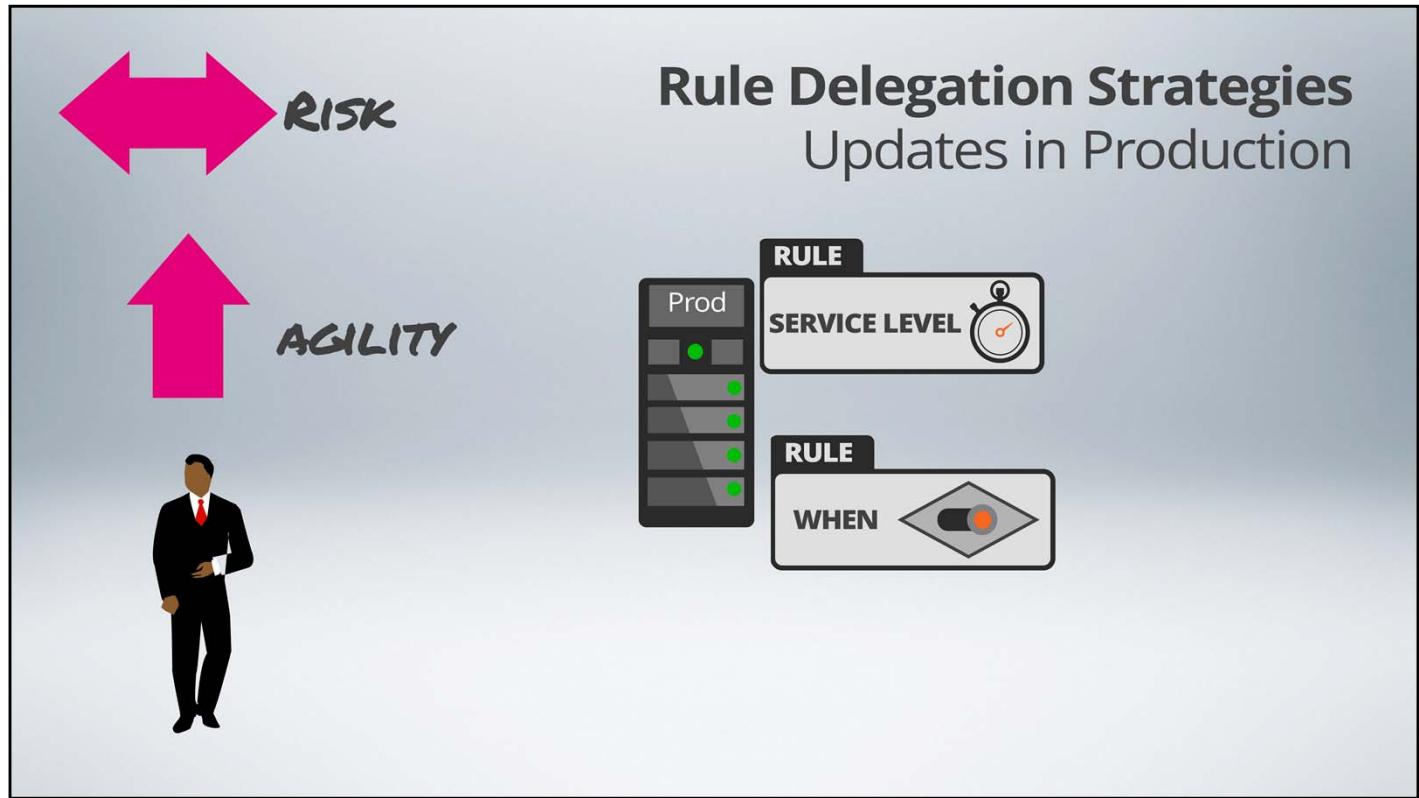
It can be done in the development environment

in which case the rules are changed and managed by the business

but the promotion of the rules would follow a normal product delivery lifecycle.

This approach has the least risk

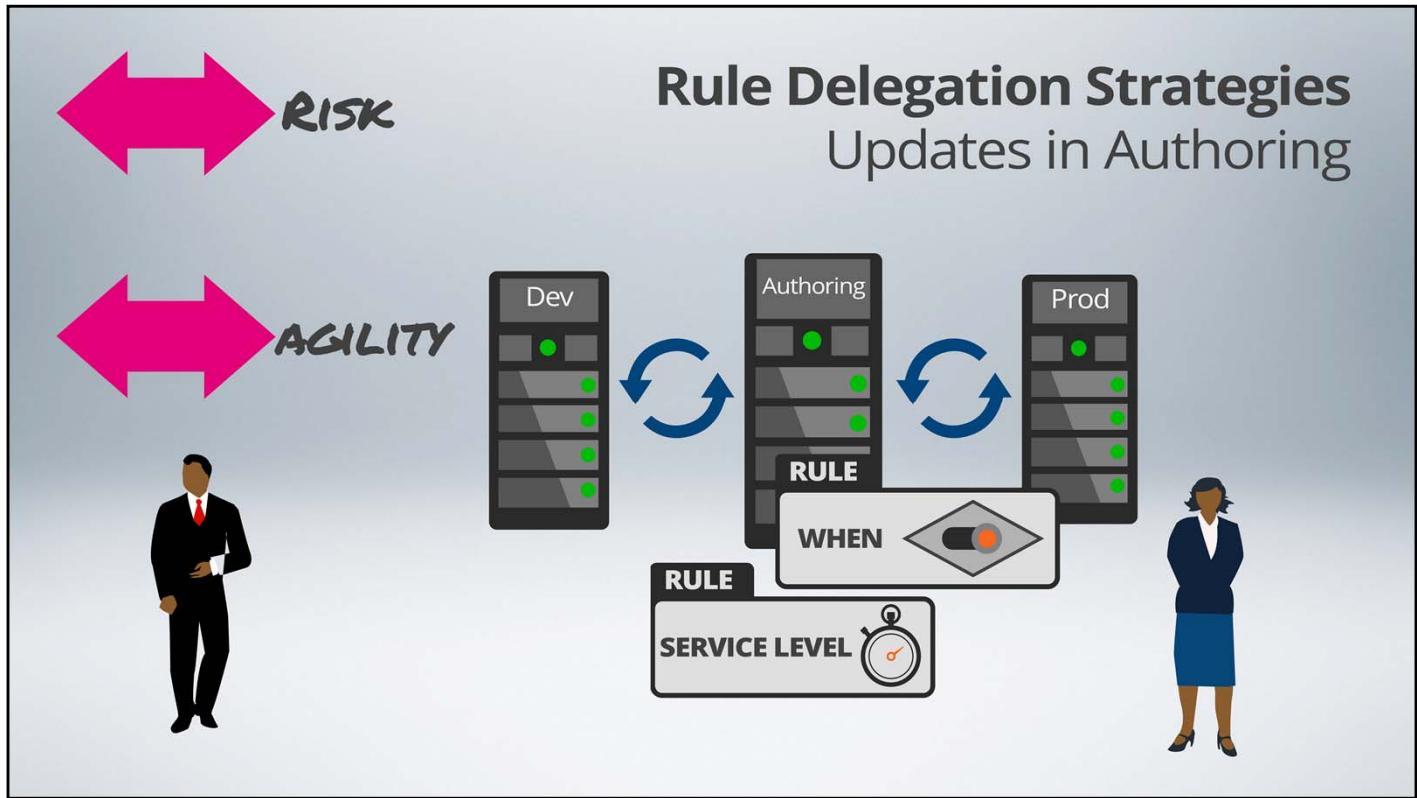
but provides the least agility.



On the other end of the spectrum is having the rules managed directly in production.

This is often useful if there is a small set of volatile rules that can be managed in production with minimal risk.

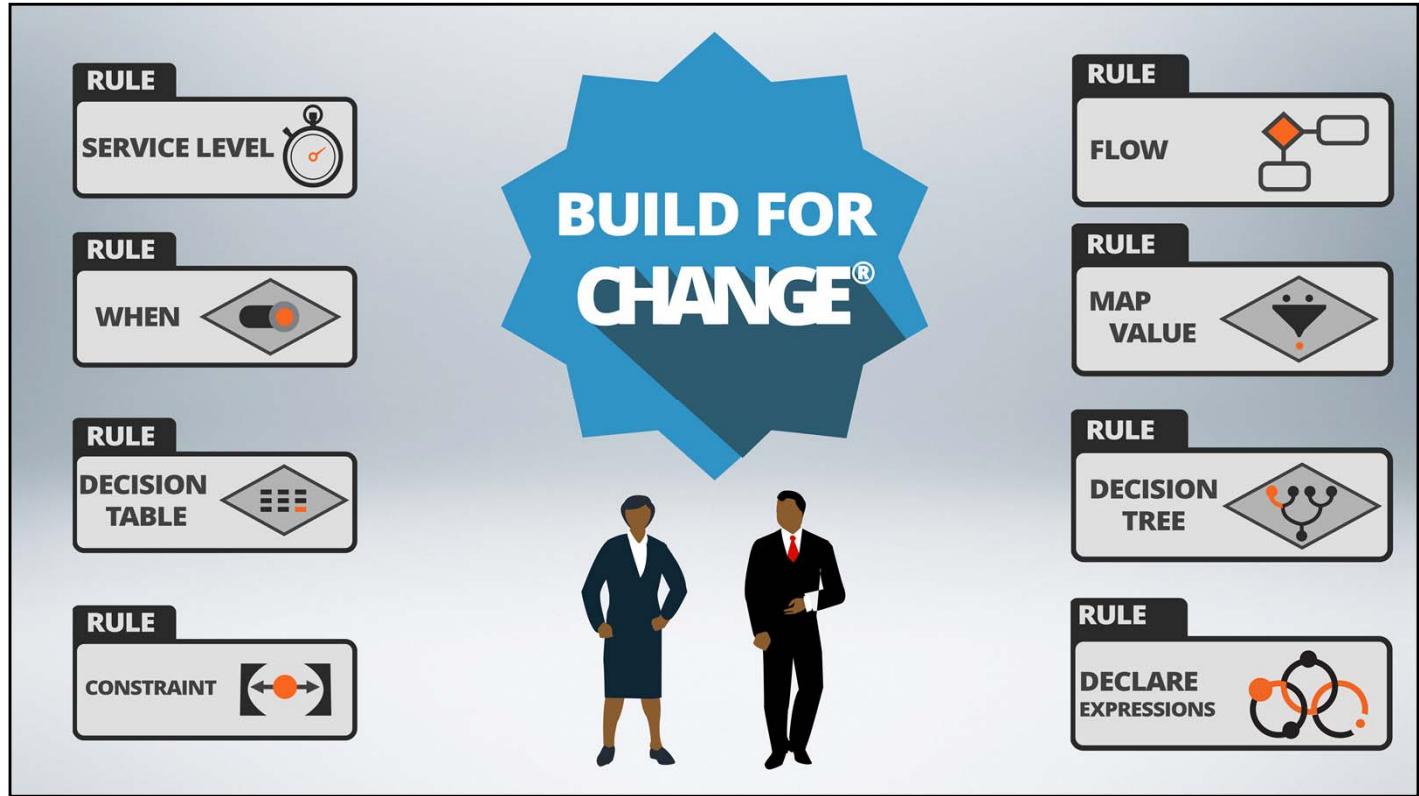
This option requires planning and risk mitigation but provides the business with a lot of agility.



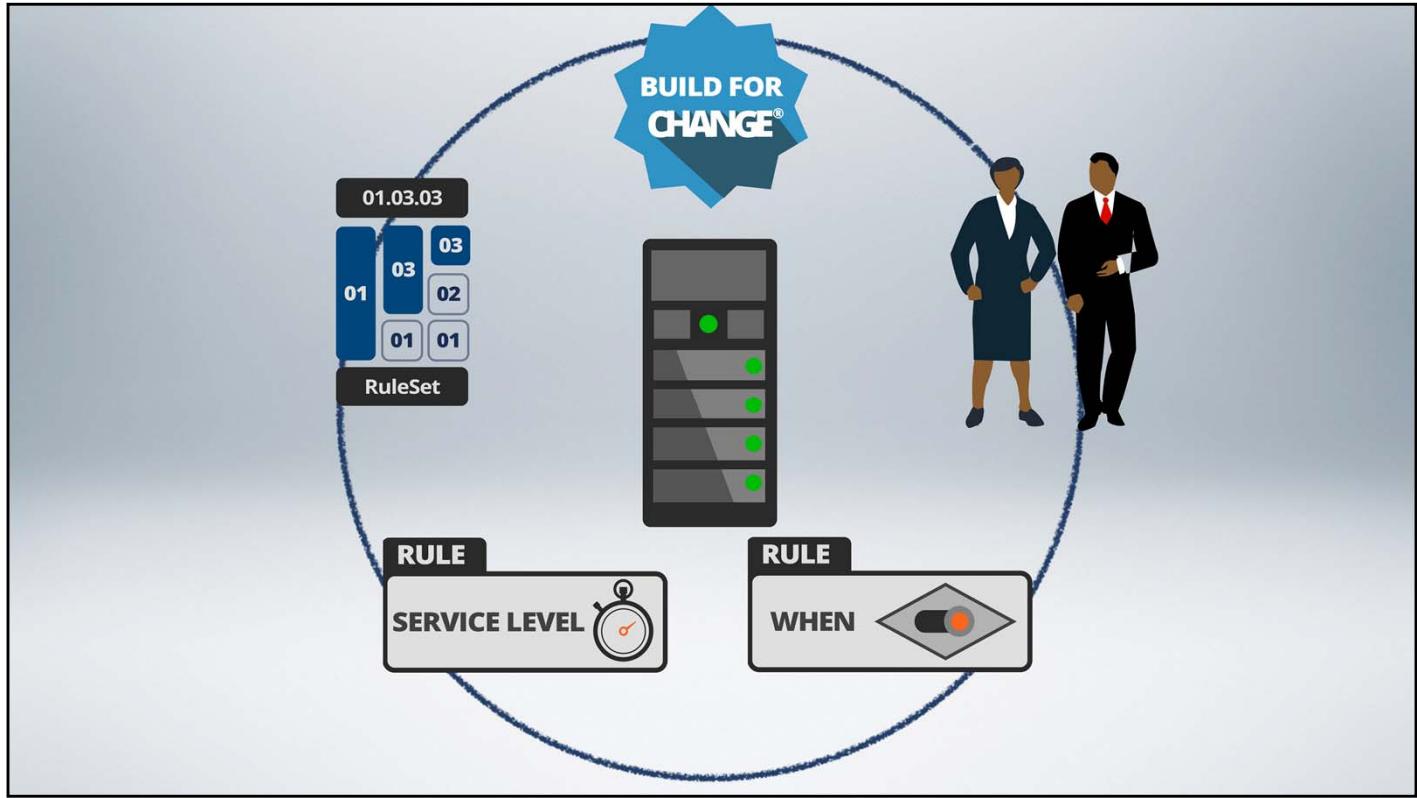
A compromise might be to consider using a separate authoring environment.

Once created in the development environment and moved to production, rules can be managed in an Authoring environment by the business users and tested without the risk of affecting production in any way.

Once tested, the migration wizard can be used to promote the rules into production on a separate cycle from the standard development environment.



Understanding how to “design” business rules for business users is critical to a successful “Build for Change®” strategy. However, only when we delegate rules do we truly realize the power of building for change.



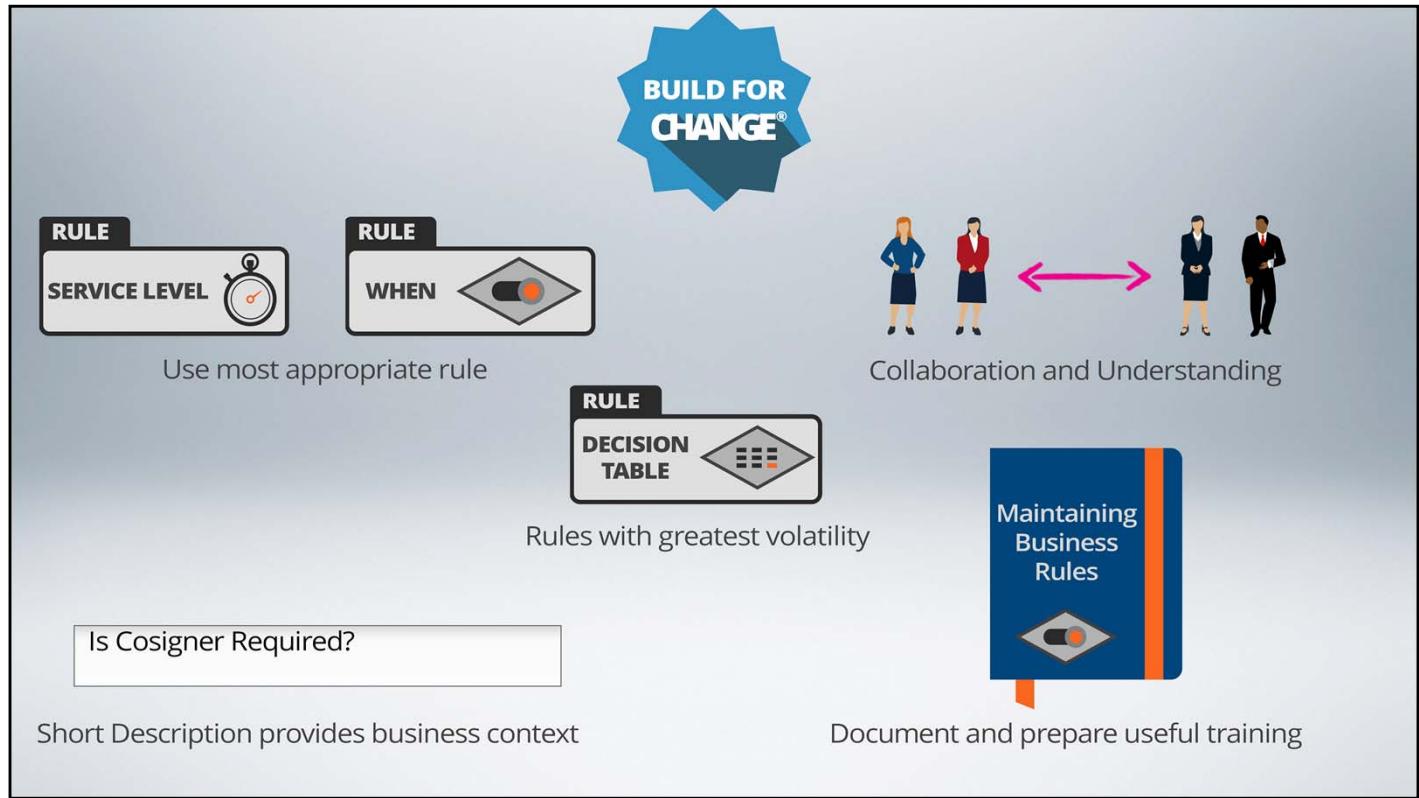
Before we summarize this lesson, let's be clear. To actually delegate business rules – regardless of the environment where the delegated rules would be managed –

we must address a few other items such as preparing a production RuleSet

and setting up the appropriate users with the appropriate privileges

so they can actually manage the delegated rules, but we'll save that for another time. Again, if we are prepared - and have built for change - the effort to actually delegate business rules is that much less labor-intensive.

Remember, to be most effective at building for change, we must be prepared. Following a few best practices at the very beginning goes a long way towards that effort.



Choose the most appropriate rule that best represents the business logic.

Work closely with the business users to identify those rules they are most likely to understand

– and have the greatest need to change more often than standard production release cycles might allow for.

When naming rules, make sure the short description provides a meaningful business context

Finally, thoroughly document the business rules and prepare useful training so the delegated rules program is self-sustaining.

# Enforcing Business Policies using Service Levels

In this lesson, we explore best practices and methods for ensuring cases are resolved in a timely manner.

At the end of this lesson, you should be able to:

- Identify common users for service levels
- Configure service level milestones, urgency adjustments and escalation actions
- Add a service level to an assignment
- Add a service level to a case

## Service Level Agreements SLAs



Urgency: 10



An important part of most business policies is not just about how and when work gets done, but the timeliness of the work.

An important part of most business policies is not just about how and when work gets done, but the timeliness of the work.

# Service Level Agreements

## SLAs



Service Levels - also known as Service Level Agreements or... SLAs - are used to ensure work is completed within the expected time intervals.

We define service levels using three milestones, which are used to indicate the expected turnaround times for the assignment or the overall case on which they are defined.

# Service Level Agreements

## SLAs



The first milestone is the goal.

The “goal” defines how long the assignment should take and is typically measured from when the assignment was started.



The next milestone is the “deadline.”

The deadline defines the longest amount of time the assignment may take before it is considered “late”.

The deadline is also usually measured from when the assignment was started.

# Service Level Agreements

## SLAs



Finally, there is the “Passed Deadline” milestone.

This is an additional milestone where we can take further action if the assignment is too far past the deadline.

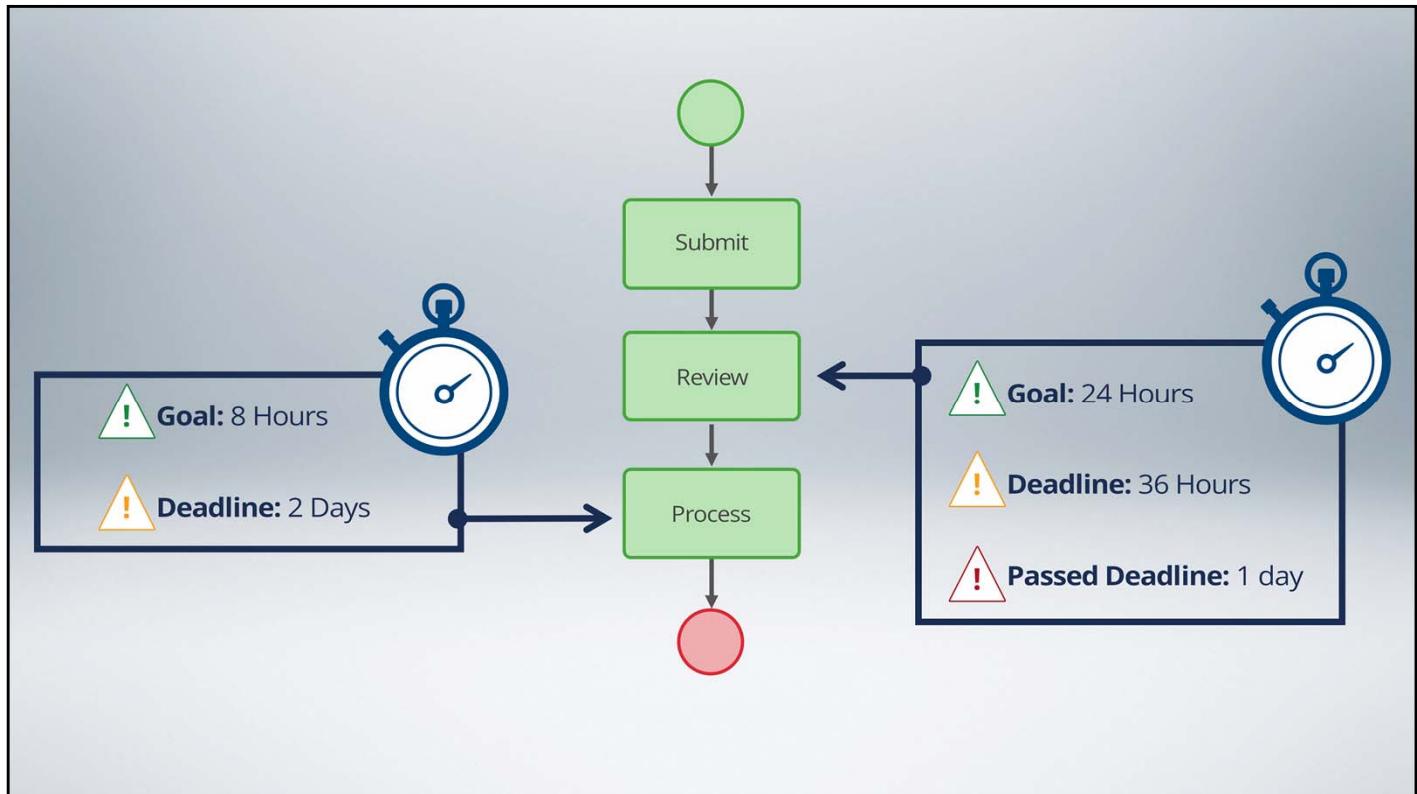
The “passed deadline” is always measured from the end of the deadline.

# Service Level Agreements

## SLAs



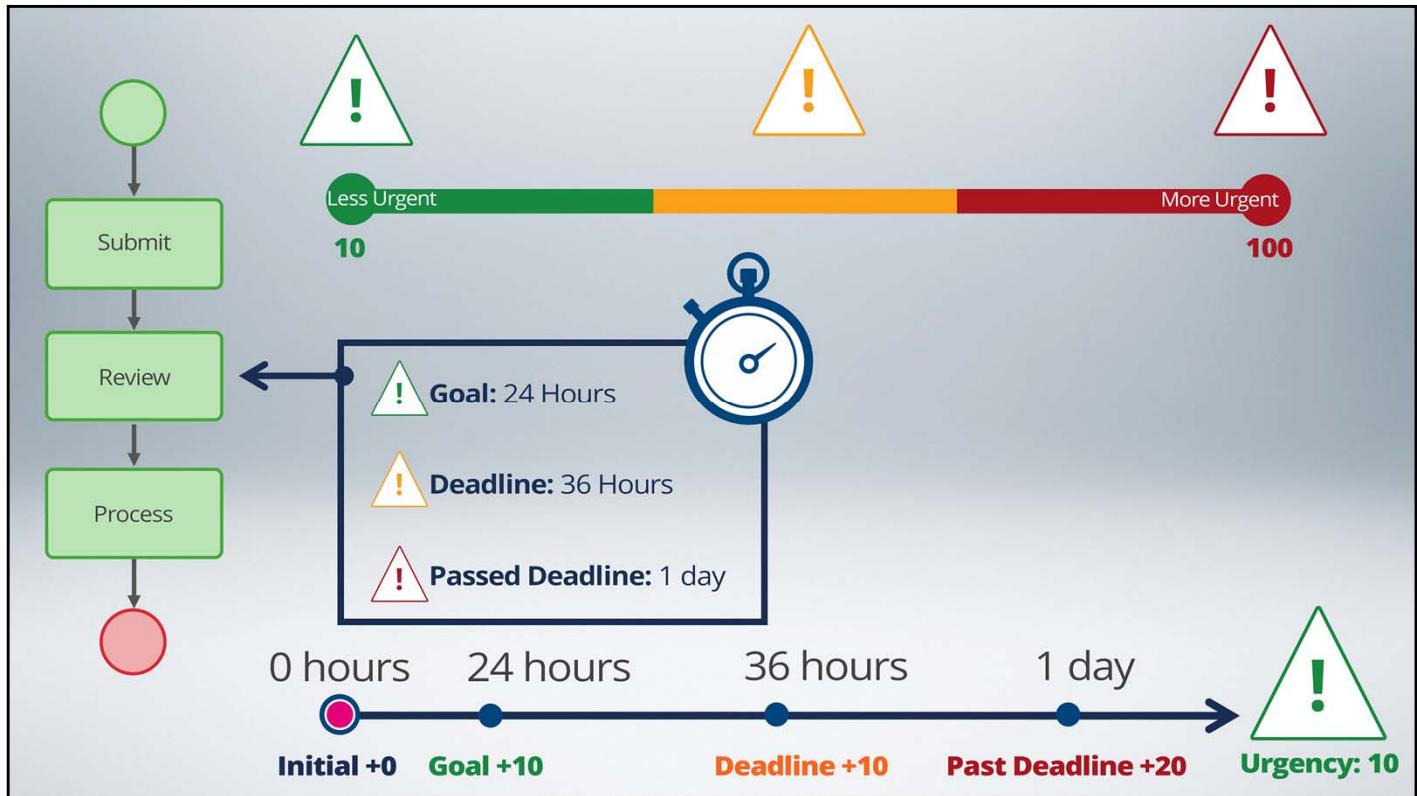
Also, the passed deadline milestone can be set to repeat its countdown cycle a specified number of times.



Let's look at an example focusing on an assignment; the behavior for a case is effectively the same. We'll use a simple process where someone submits a request, it gets reviewed, then processed.

We expect managers to be able to review a request within 24 hours of getting it, but allow them up to 36 hours before it is considered late, and one additional day before it is considered “passed the deadline”.

We may then expect it to take up to eight hours to process the request, but never more than two days.

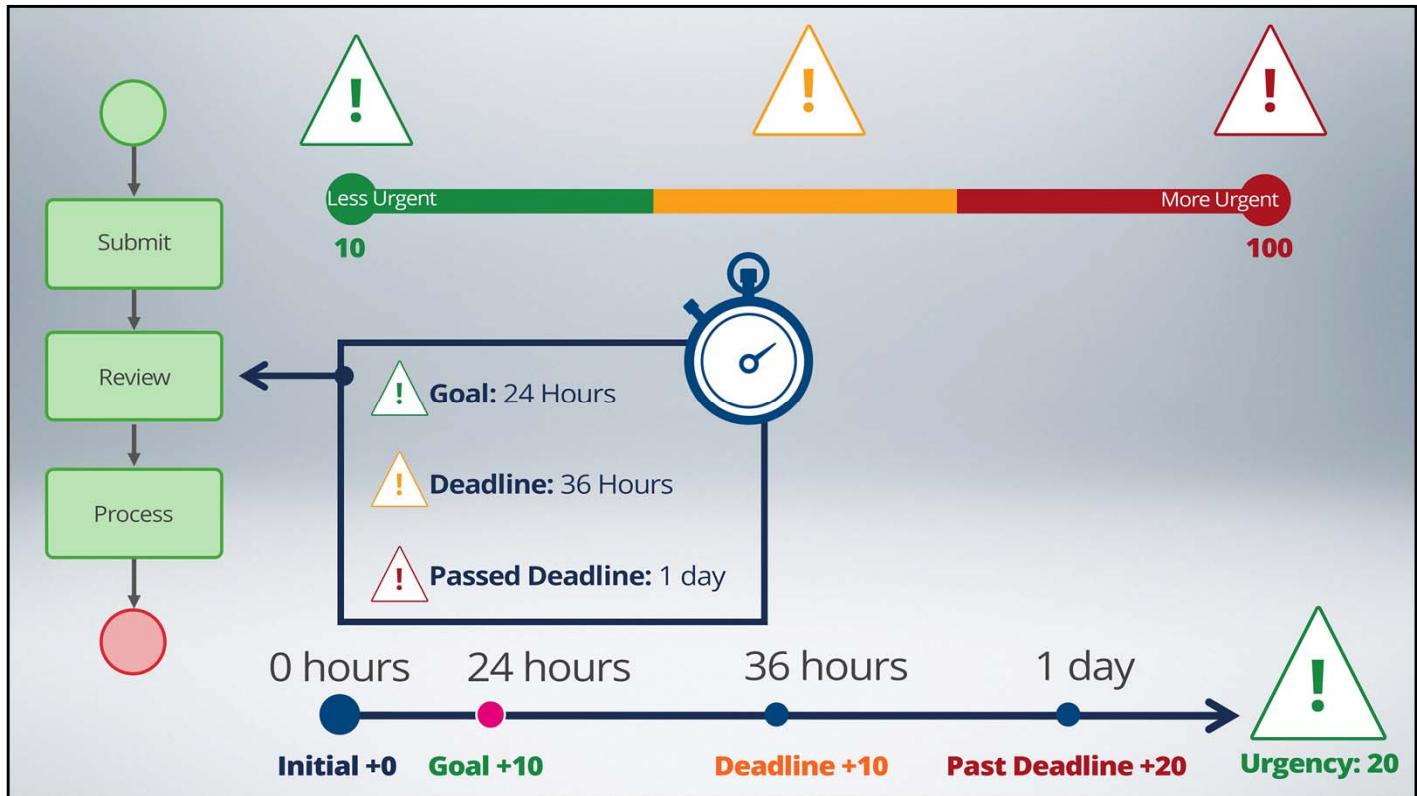


At each milestone, we can adjust an assignment's *urgency*.

The urgency is typically a value from ten to one hundred; the higher the value, the higher the assignment's urgency.

Let's take the Review assignment along a time line and see how the urgency can be adjusted based on the milestones we set.

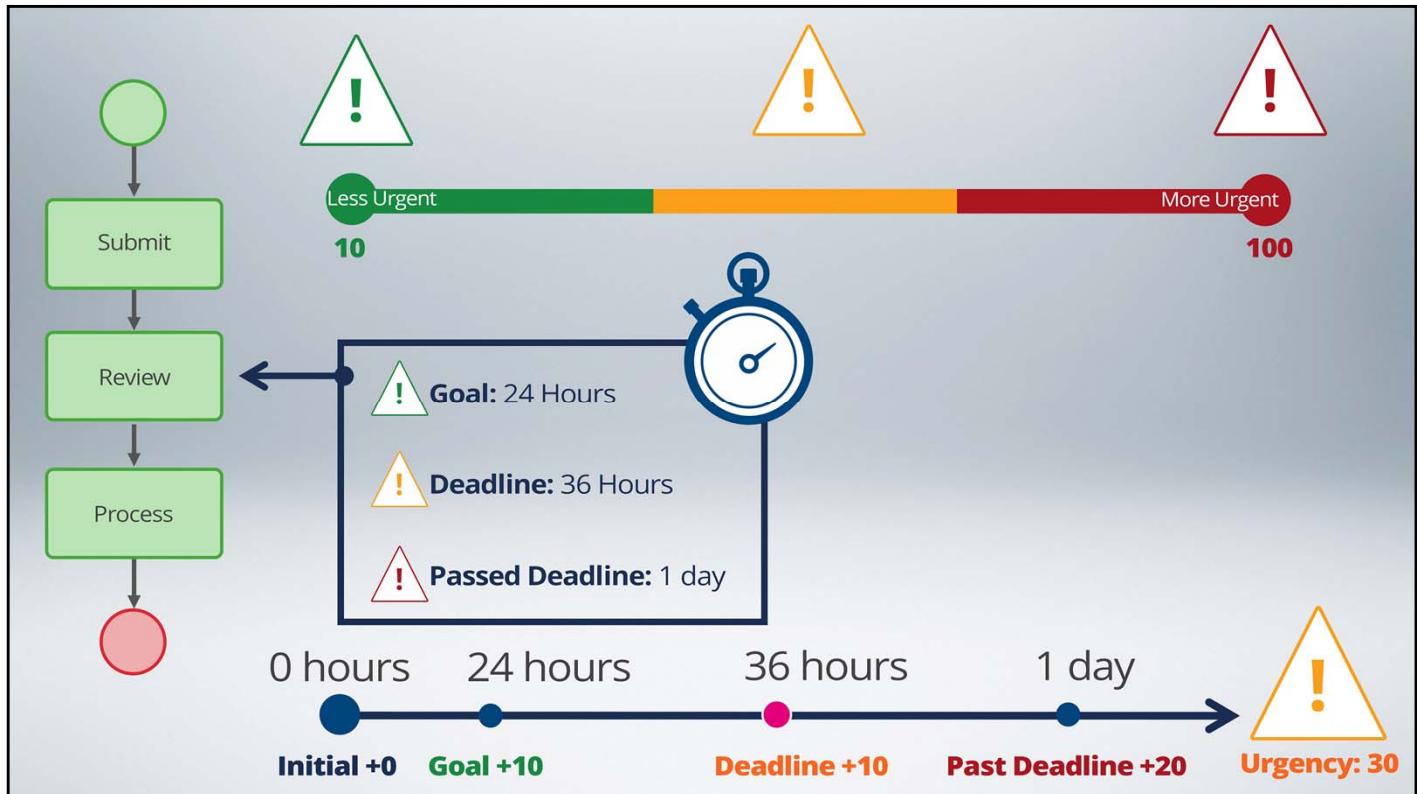
A new request was submitted and now must be reviewed. As soon as we get to the Review step, the assignment is considered started so the service level clock starts ticking.



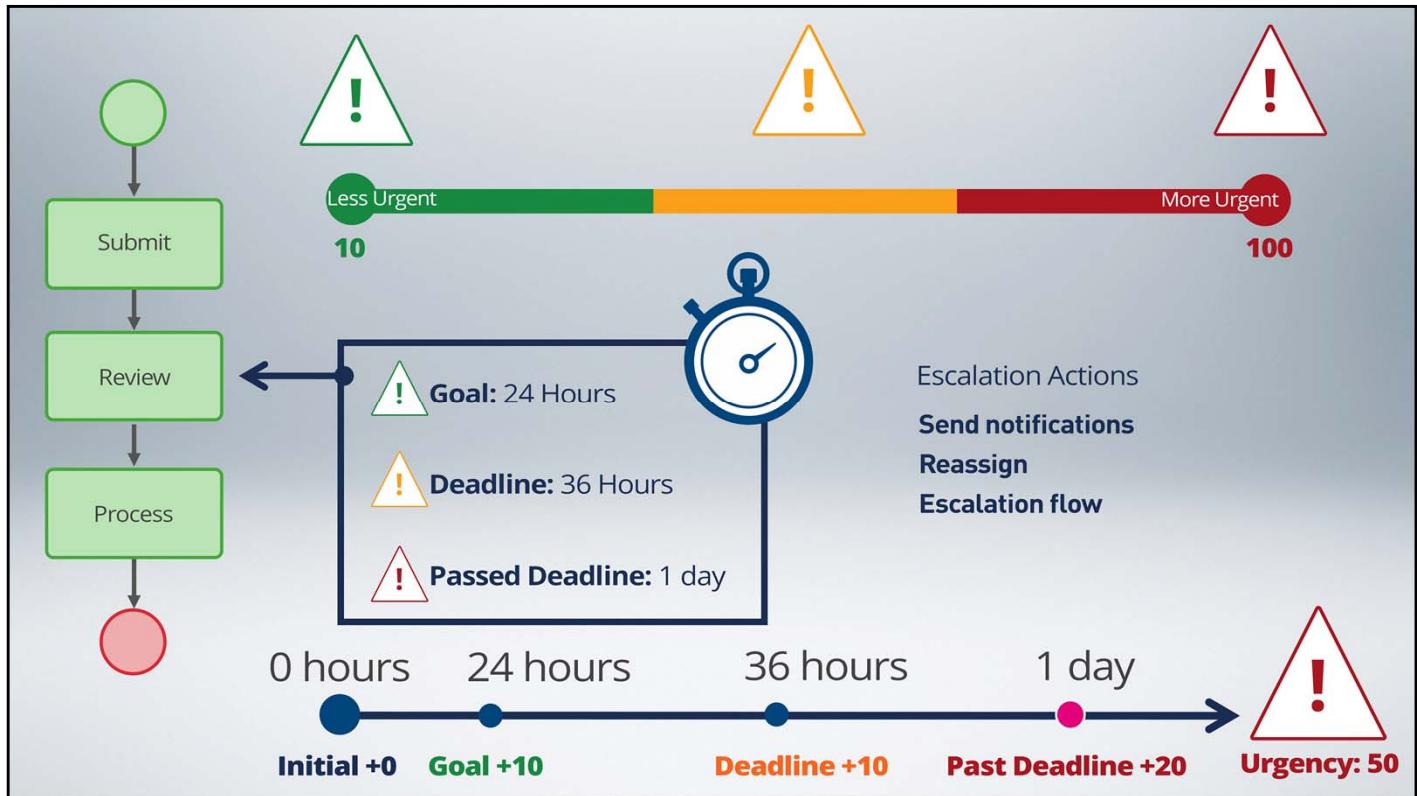
At the start of every assignment, the urgency is set to a default value, typically ten. If we wanted to, we could increase that urgency right when the service level clock starts, but in this example, we'll leave it as is. So, the urgency value for the review assignment is ten.

Then, time passes...

The *goal* for this assignment is twenty-four hours. If twenty-four hours elapse and the assignment is not yet completed, the urgency is increased by a defined value; in this scenario, let's increase the urgency by ten, so the assignment's urgency value for the review assignment is now twenty.



More time passes – let's say another twelve hours, which is the deadline – and the assignment is still not completed. We missed the deadline, so the urgency is incremented again by adding the value we defined; in this scenario, so now let's increase the urgency by ten. The urgency value of the assignment is now thirty.



Now, another full day passes from the time we missed the deadline – not the start of the assignment. According to our service level, we now add 20 to the urgency, so now it's 50.

In addition to changing the urgency value, we can also define an action to take to escalate the assignment if any of the milestones are missed. Common escalation actions might include:

Sending notifications to the assignment owner and/or their manager; we could transfer the assignment to another user; or maybe even kick off an “escalation flow”

## Demo



### Create A Service Level And Associate It To An Assignment

Let's go back to the Candidate case and create a new service level for conducting interviews.

Service level rules are in the process category.

Let's provide a short description – the name of the record - of "Candidate interview" and click Create.

The "Initial Urgency" field has a value of zero. If we want to increment the assignment's current urgency by some amount, we would set that value here.

We do not have any reason to increase the urgency of the assignment as it has just started, so we will leave the value at zero.

We can define when the service level clock starts ticking in the "Assignment Ready" field. Our options are immediately after the assignment is started, as determined by some value in a property or some defined interval from when the assignment was started.

We want our service level clock to start ticking when the assignment is created, so we will stay with the "Immediately" option.

Now..., let's define the criteria for the three milestones in the service level. We'll start with the goal.

We can set the days, hours, minutes and seconds that pass from the time the assignment was routed to the assignee.

Our goal is to have each interview completed within one business day, so we'll set the Days to one

and select the “Business Days” check box.

If the interview is not completed within the goal of one business day, let’s have the urgency of the assignment increase by a value of 10.

And, let’s send an email to the person assigned to the interview. So, we’ll add an Escalation Action and select “Notify Assignee” as the action to perform.

Notice the default notification to use is “Send Email To Assignee On Goal Time.” Since we know whom the task is assigned to, the notification is sent to the email address in their operator profile.

As to the deadline, we want all interviews to be completed no later than two business days after they are assigned. So, we’ll set the deadline to two days and select the “Business Days” check box here as well.

If the deadline is missed, we’ll increase the urgency of the assignment by a value of 20.

And, this time, not only will we send a notification to the interviewer,

we will also let their manager know they missed the deadline.

That leaves us with the “Passed Deadline” milestone. Let’s provide an additional eight business hours after the deadline to complete the interview.

If this milestone is missed, we increase the urgency by another 20 points and send two more emails.

Finally, our intent is to spam the interviewer and their manager until the interview is completed so we’ll set the “Repeating interval from Deadline” to 3. Now, every eight business hours they will receive yet another email letting them know the assignment is overdue. In our example this will happen three more times, but we could set the interval value to whatever we need it to be.

Okay, this service level is configured, so we’ll save it.

Now we need to associate the new service level record with the “Conduct Interview” assignment in the Candidate case. So let’s bring the Candidate case back into focus.

We’ll select “Configure process detail” to bring the assignments into focus.

We’ll select the “Conduct Interview” assignment.

With this screen setting, we’ll need to scroll down a bit to see the service level field.

Now...let’s select our “Candidate Interview” service level record.

And that’s it, we can save our flow.

## Demo



### Configure A Service Level For A Case

We can also set service levels at the case level. To do this, we open the case for which we want to set a service level. Let's use the Candidate case.

In the Case Designer, we'll select the "Details" tab.

Among the options we have are the "Goals and Deadlines."

As a default, there are no goals or deadlines set, so let's edit this option.

We can control when the goals and deadlines are calculated from the case we are currently editing, the parent case if the case we are adding the service level to is a child case, or a top-level case if the case we are adding the service level to is a grandchild case.

In our example, we are setting goals and deadlines for the main case, so we will stay with the "Me" option.

Now, let's add a goal and a deadline. We'll consider the goal to be five days and the deadline to be six.

Then, we click OK.

Now, let's save our changes so they take effect.

Behind the scenes, a new service level record was created – and we need to make a few more modifications for it to be useful.

Let's go to the App Explorer and open the newly created service level record.

We'll need to refresh the App Explorer – so let's do that.

Let's expand our Case, then the Process category, and finally the Service Level category.

The service level record for a case is named "p y Case Type Default: - let's open it.

Notice it is a standard service level record – so we can make whatever adjustments we need, such as

setting the initial urgency value, setting urgency increments for when the goal, deadline and passed deadline milestones are crossed, etc.

If you do make any additional changes, remember to save the record.

## Exercise: Configure a Service Level for a Case and an Assignment

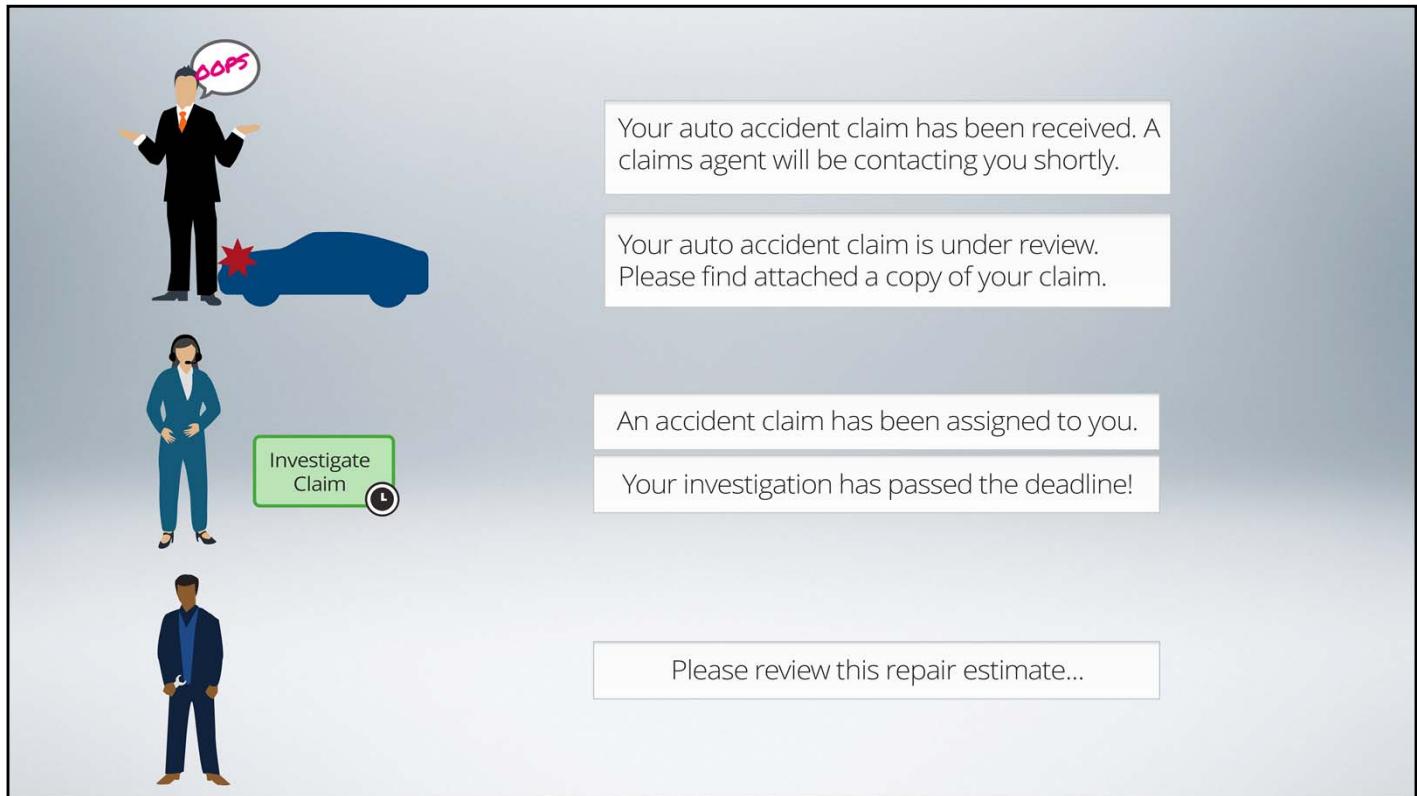


# Notifying Users From Within a Process

Notifying users of status and other changes are a very common requirement for most business applications. In this lesson, we will explore the most common reasons for communicating with end users, followed by a study of best practices for effecting these communications using Pega 7 out-of-the-box capabilities.

At the end of this lesson, you should be able to:

- State common reasons for communicating with users
- State the purpose of a Work Party for sending notifications
- Identify the four correspondence types available for



Most organizations depend on timely communication with customers, application users and others to establish a shared understanding of transactions or assignments.

For example, we may need to communicate with the originator of a case such as someone who filed an auto accident claim.

We might have a requirement to notify them that their claim was successfully filed and then periodically as the claim progresses towards resolution.

Another very common notification requirement is keeping case workers up-to-date.

There may be a requirement to notify a case worker that they have a new assignment or maybe keeping them up to date on their work progress.

And...finally, we may have a requirement to communicate with users who are indirectly involved in the case, such as an external agency.

**1****Who?**

Work Parties

**2****How?**

email - text message - fax - letter

**3****When?**

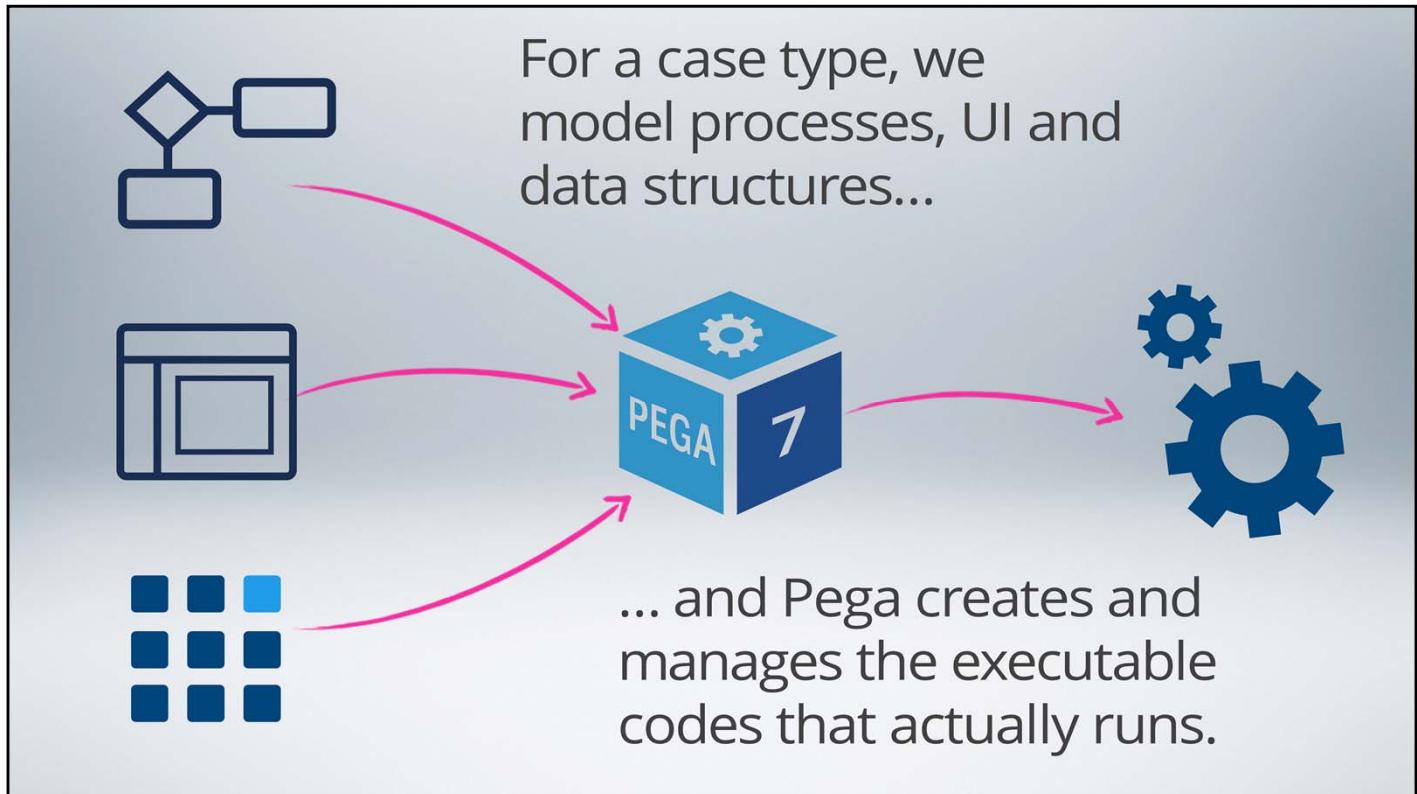
start of an assignment, missed deadline, change of status, etc.

To effectively communicate we need to answer three fundamental questions. First, we must know with whom do we need to communicate with.

Secondly, we need to know “what” do we need to tell them.

Finally, we need to determine “when” do we need to tell them.

Timely, clear communication keeps participants engaged in the resolution of a case and can help keep the process working efficiently and effectively.

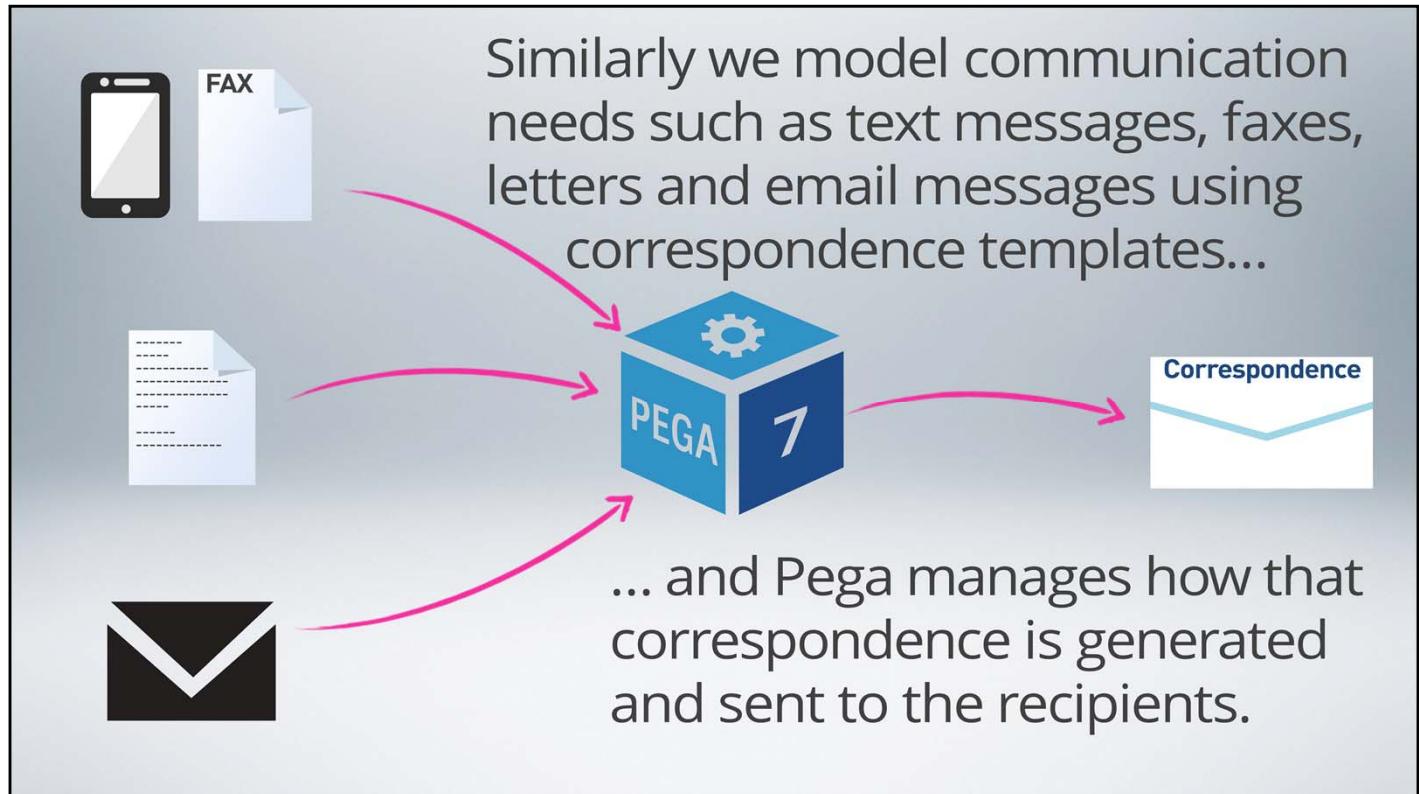


Our greatest need in crafting communications is to establish the recipient – with whom we need to communicate?

Remember, a case type is a model for managing a case.

We model the processes, user interfaces, and data structures, for example, and Pega creates and manages the executable code that actually runs.

Communication is no different.



We model our communication needs such as text messages or faxes, letters and even email messages using correspondence templates.

Then, Pega manages how that correspondence is generated and sent to the recipients.



We model the recipients of the communication using...

## Work Parties

- **Originator** – initially submits a case
- **Manager** – approvals and oversight
- **Customer** – on whose behalf the case was submitted
- **Interested** – needs to be aware of the case, but not involved

We'll even model the recipients of the communication.

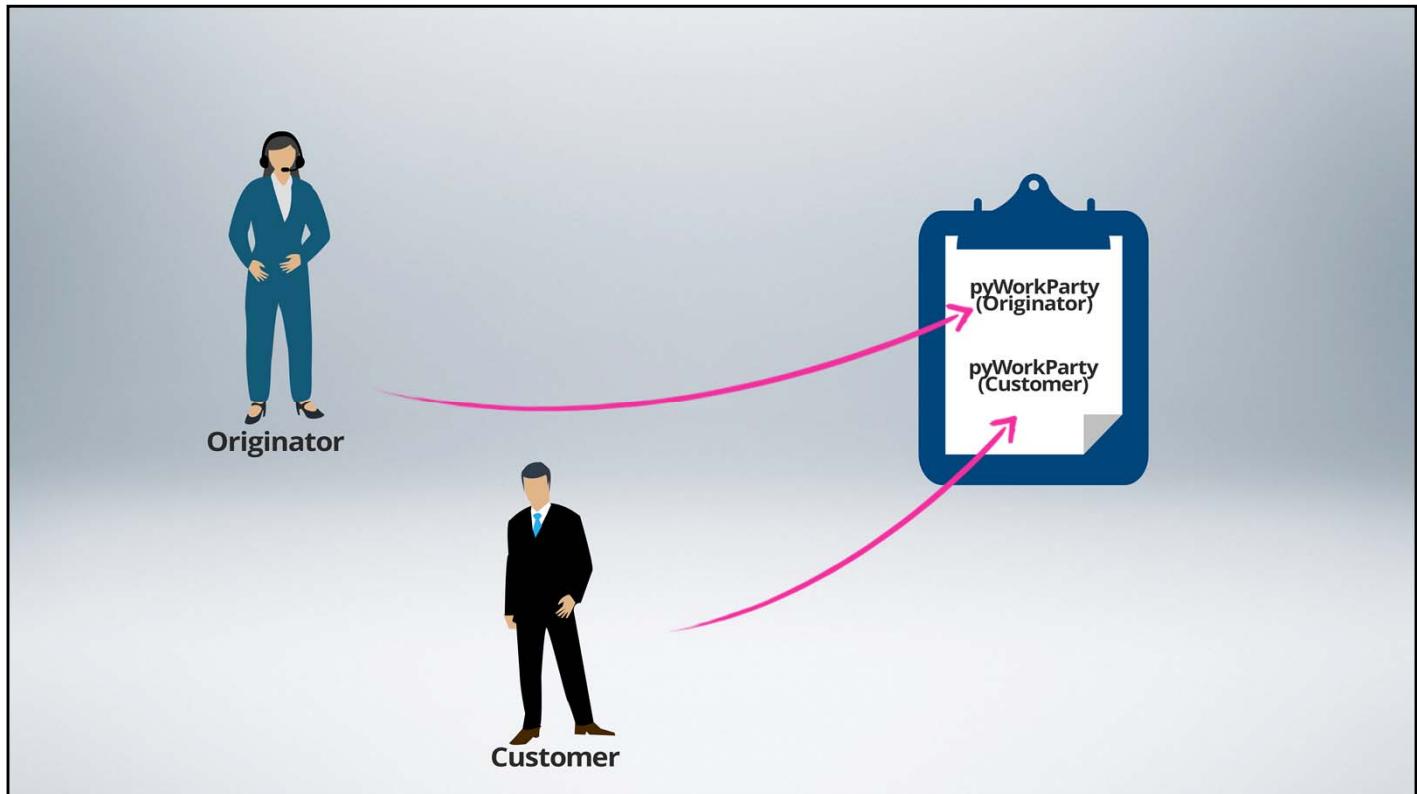
This allows us to refer to the recipients we mentioned earlier using common terms such as an "Originator" – the person who created the case.

We could use "Manager" to model the person who might be responsible for approvals and oversight.

"Customer" can refer to a person on whose behalf the case is transacted. This person may not be involved in the processing of the case, but may want – or need – to be informed of any changes. While we are here, a "customer" may sometimes be the "originator" of a case, but not always.

And, finally, we could use "Interested" to model someone who needs to be aware of the case progress, but is not personally involved in processing that case.

These people or entities can be associated with individual cases – and are called "Work Parties."

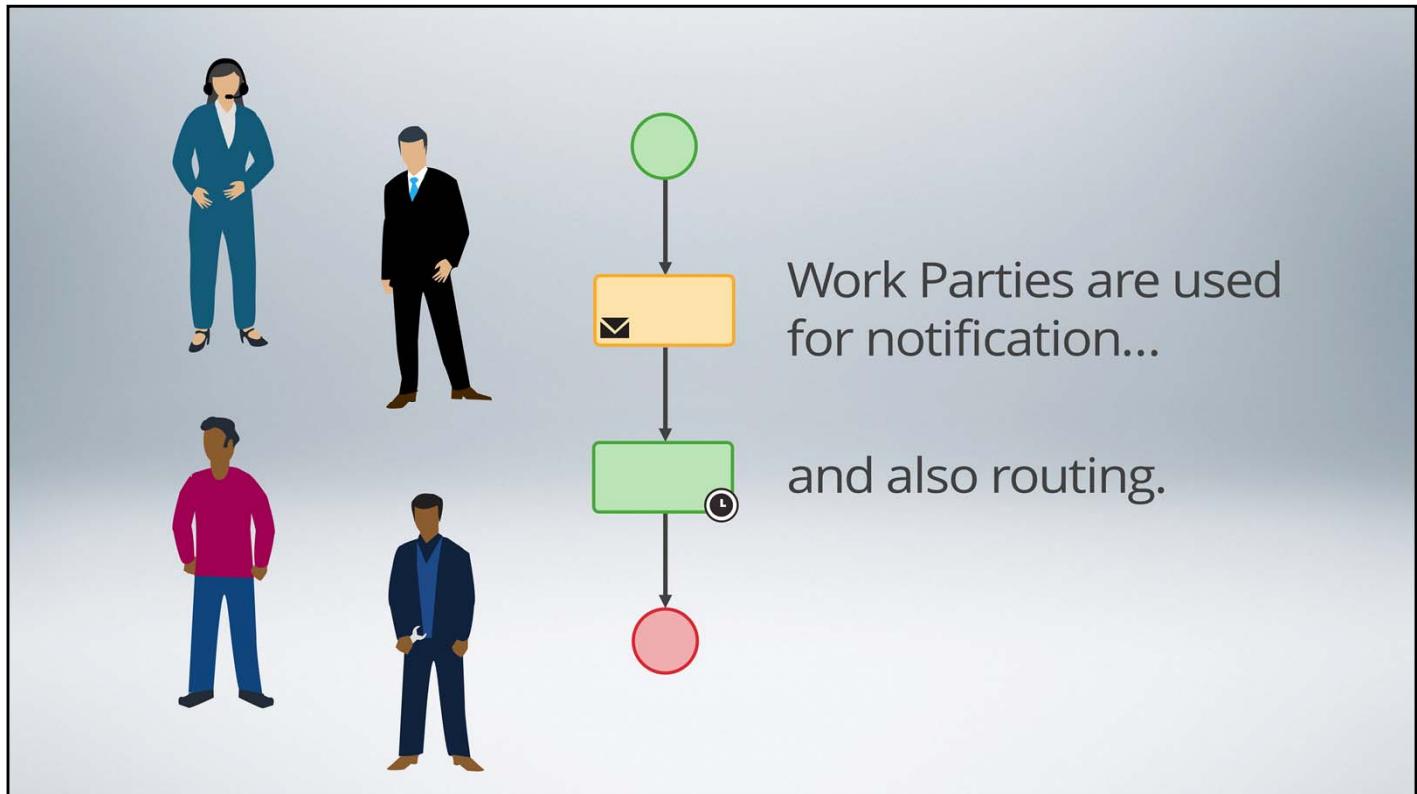


A “work party” allows us to create a placeholder – a page on the clipboard - with information collected from the case.

For example, we can use session data for the current operator to identify and populate the “Originator” work party role when the case is created.

And, if we collect customer information, we can use this information to identify and populate the Customer work party role.

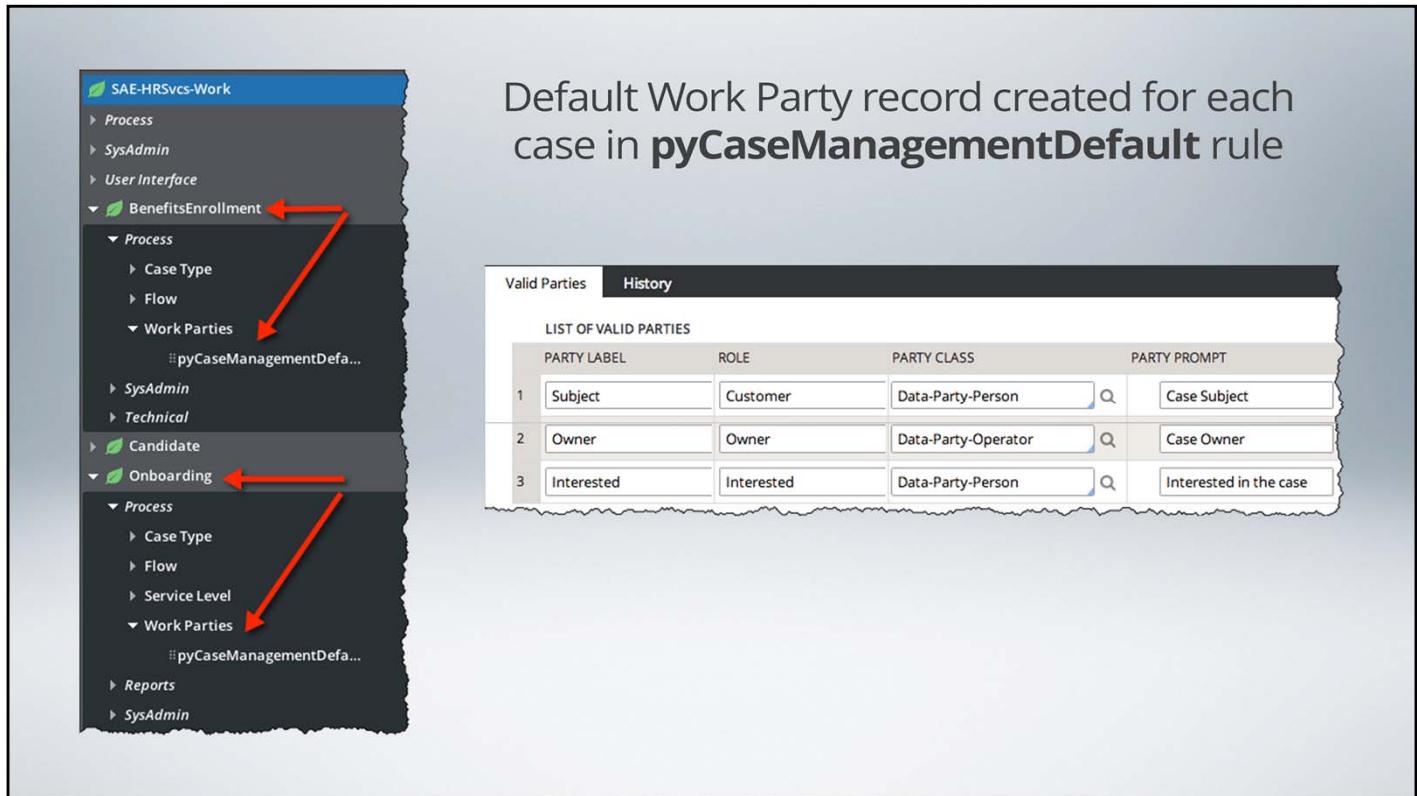
This avoids the need to hard-code user details, some of which we can't possibly know until after the case is created.



Now...before we move on. Work parties are used for more than just communicating with users.

They are also used for routing assignments to the right users.

Given that the focus in this lesson is on creating the actual correspondences, all we need to know about work parties is that they are used to identify the recipient of the communication; we'll leave the details of creating work party records for another time.



The screenshot shows the Pega Platform's navigation bar on the left and a list of case types on the right. Red arrows point from the text descriptions below to the 'Work Parties' sections for 'Benefits Enrollment' and 'Onboarding'.

**Default Work Party record created for each case in **pyCaseManagementDefault** rule**

Valid Parties History

LIST OF VALID PARTIES

PARTY LABEL	ROLE	PARTY CLASS	PARTY PROMPT
1 Subject	Customer	Data-Party-Person	Case Subject
2 Owner	Owner	Data-Party-Operator	Case Owner
3 Interested	Interested	Data-Party-Person	Interested in the case

However, for each case type we create, a default Work Party record is created – the **pyCaseManagementDefault** work party record.

This work party record contains three standard entries. One for the “Customer,” one for the current “Owner” of the case, and one for those “Interested” in the case.

The “Owner” work party data is automatically set when the case is created and then updated as the case moves from one case worker to another. The case type could be configured to add additional work party data as it becomes necessary.

**1****Who?**

Work Parties

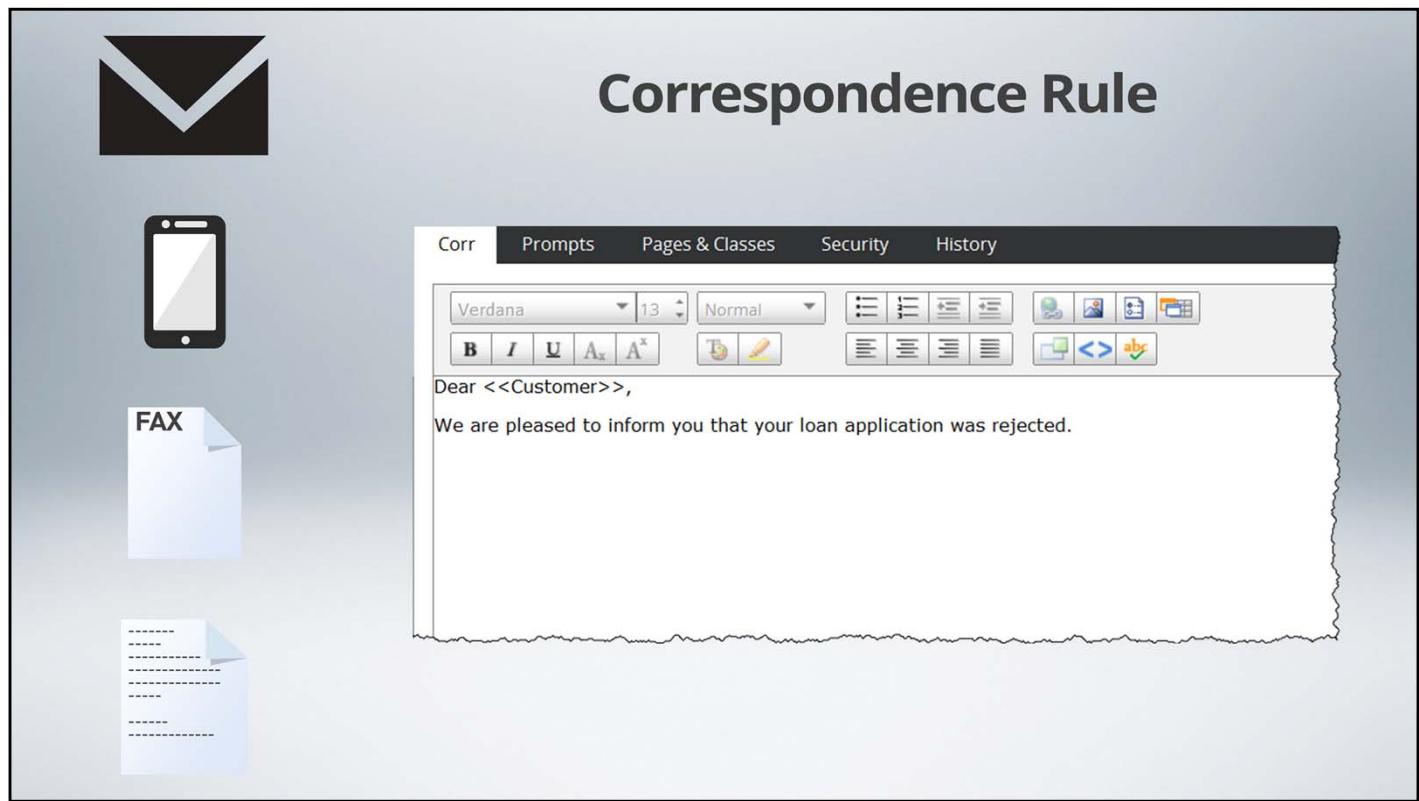
**2****How?**

email - text message - fax - letter

**3****When?**

start of an assignment, missed deadline, change of status, etc.

In order to generate correspondence, we not only need to define “who” to send the correspondence to, we need to know what to send – or “how” we want to communicate with the recipient.



Pega provides four correspondence types: email, text message, fax and regular mail.

Each correspondence type provides unique functionality behind the scenes, however they all share the same basic template.

A “Rich Text Editor” allows us to create formatted correspondence, including a variety of options for changing the appearance of the text, adding images and URLs and referencing properties.

And, then there is the text area for the body of the message.

Based on business requirements, we may create one or more of each of these for each type of correspondence we need to send.

## Demo



### Creating a Correspondence

Correspondence records are organized in the Process category. So, to create a new correspondence item, we'll right click on the Process category and select New, then Correspondence.

A new correspondence record opens. When considering how to identify the correspondence record, always provide a meaningful, short description which describes the intent of the correspondence record.

In this example, we want to send a confirmation letter when a new account is approved, so, we'll enter "Send Confirmation Letter." For the "Correspondence Type", we see the types available in the drop down. To invoke this list, place your cursor in the "Correspondence Type" field, and press the down arrow on your keyboard.

In our example, we want to send a letter, so we'll select Mail. Now, click Create. Using the Rich Text Editor, we'll enter the body of the message. So that this correspondence record can be used as a template to send a letter to any customer, we can include properties using the Add Property button. Let's add the customer's full name.

The correspondence rule provides a Rich Text Editor with a wide variety of formatting tools to change the appearance of our correspondence. The toolbar includes selectors for bold, italics, fonts, font size, color, spell checking, list formatting, alignment and including images and graphic elements. Complete the correspondence according to the business requirements, then click Save. The correspondence record is ready for use.

**1****Who?**

Work Parties

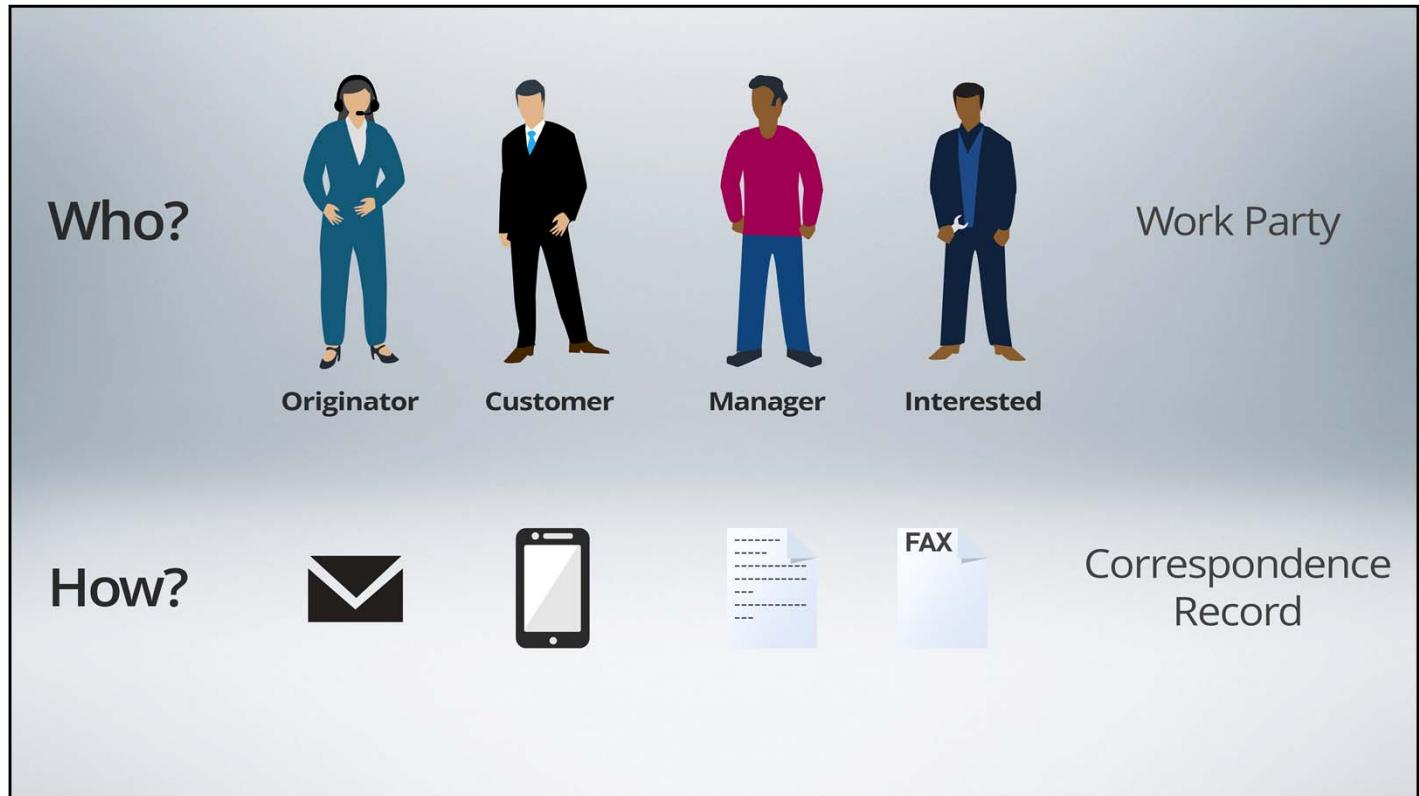
**2****How?**

email - text message - fax - letter

**3****When?**

start of an assignment, missed deadline, change of status, etc.

The last of the questions we need to answer is “when” do we communicate. But first, let’s do a quick review to make sure we keep everything straight.



To automate communications effectively, we must identify “who” we want to communicate with.

We model these entities using a “work party” record.

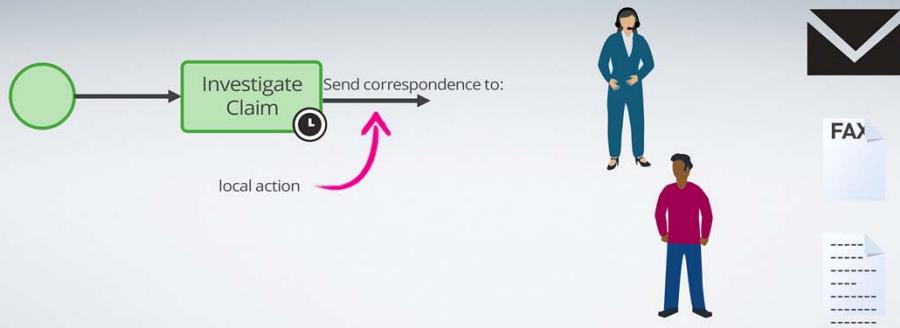
Next, we need to understand “how” we communicate with these parties.

We have four out-of-the-box options, and we model “how” we communicate using a correspondence record for each type of correspondence we need to send.

**Automated:** mandatory; sent automatically when shape is reached



**Manual:** optional; at the operator's discretion



The last of the questions we need to answer is “when” do we communicate with users. We have two options for managing when correspondence is sent.

Our first option is to send correspondence automatically, using various flow shapes. These types of correspondences will always be sent – as specified - when a shape which has correspondence defined on it is encountered at run time.

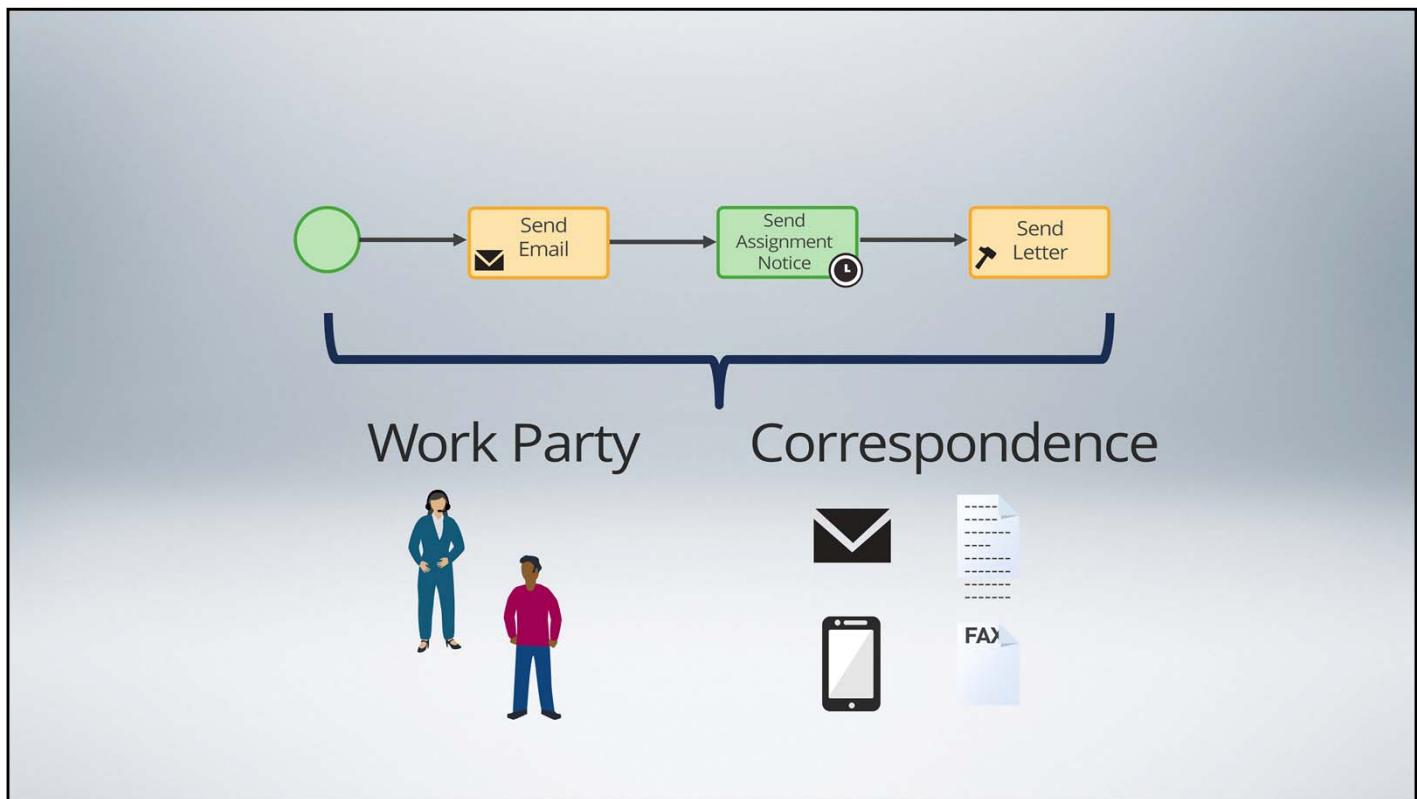
Use the Email smart shape to send...well...an email.

We can use the “Notification” tab on an assignment to send correspondence to one or more recipients. This is typically used to notify a user that a task is assigned to them, but we can also use it for any number of correspondence needs – such as notifying a customer that their case has reached a certain status.

Finally, there is the Utility shape. We use this shape to send other correspondence such as text messages, letters or faxes.

The other option is to let the user decide when working on an assignment, using a local action.

Using this option, we can configure the case so the operator can not only decide who to send the correspondence to, but how as well.



Regardless of when we send the correspondence, there are always two required pieces: we need to know who we are sending the correspondence to – the Work Party - and what we need to send them – the correspondence record.

Now you can see why we identified those two pieces first.

## Demo



### Sending Email Correspondence

Let's start with the easiest correspondence – and one we should already know how to use - the email smart shape. We'll use this as a short review opportunity. Let's confirm we have the two components always needed to effect communicating with a recipient. We need to have the work party identified, and we also need to have our correspondence record ready.

Check...and check. Now, let's turn our attention to the email smart shape. Let's double-click to open the properties panel. We want to use our work parties so let's select the "Party" option. In the "Parties" field, we'll select our "Customer" – which is defined in the py Case Management Default work party record. Next, we'll provide a subject line and, then, let's use the nifty correspondence rule we took so much time to create.

Click OK and we're done. All we have left to do is save the flow record to effect our changes

## Demo



### Sending Notifications From An Assignment

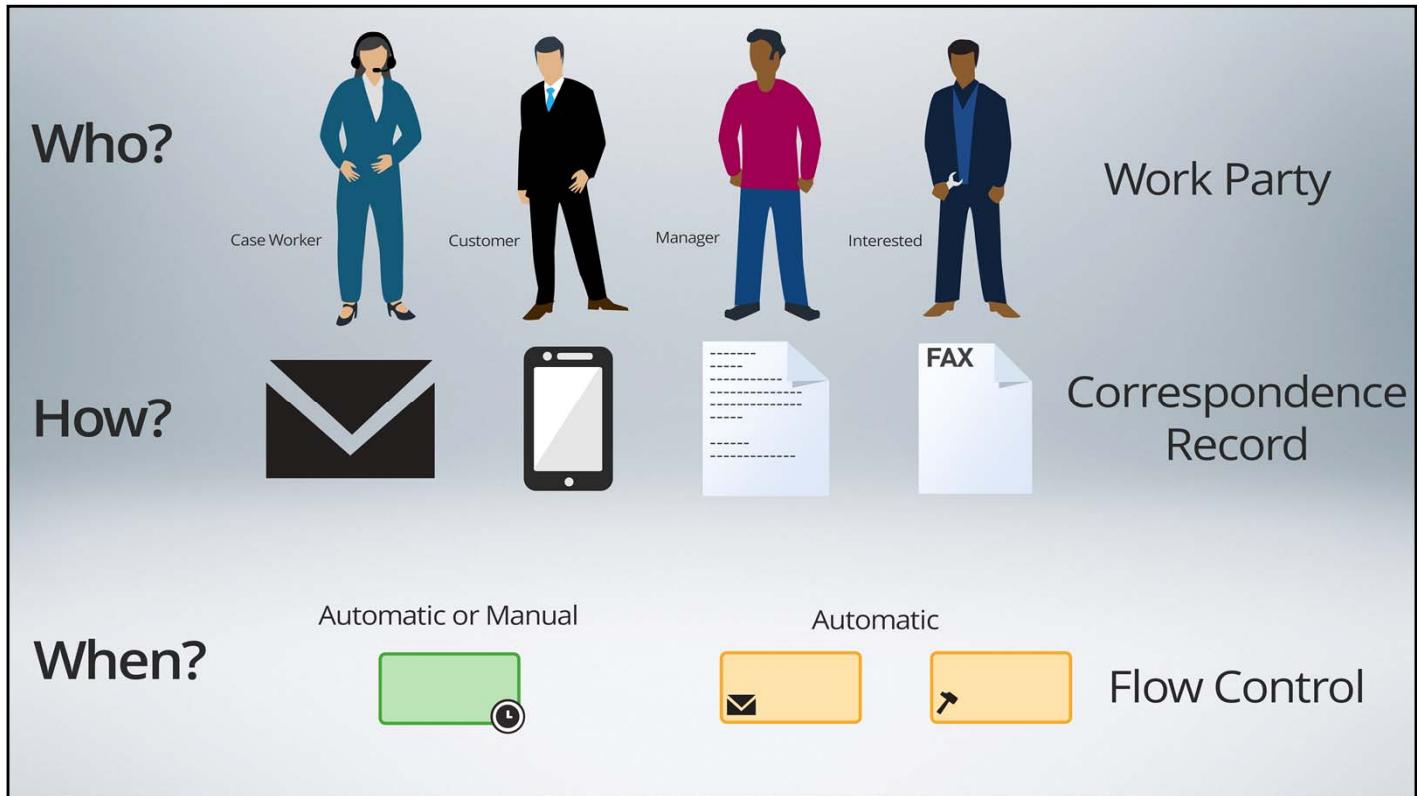
Correspondence can also be sent using the Notification tab of an Assignment's properties panel. But, first, let's make sure we are ready. We need to have identified to whom we want to correspond. Remember, a default work party record is created for each case type, and this record contains entries for a "Customer," an "Owner" and an "Interested" party.

We also need to have prepared how we are going to correspond. We've created a correspondence rule in advance.

Now, it's time to set up the notification. We'll double click on an assignment to open the properties panel. In this example, we will configure a notification on the "Review Applicant Information" assignment. Next, we'll click the "Notification" tab. We'll place our cursor in the "Notify" field, and use the down arrow on our keyboard to display the smart prompt choices. We have a number of options including notifying everyone identified in the work party record, specific parties, or just the individual assigned to the task.

In our example, we want to simply notify the user that the assignment is ready. Now, we need to provide at least a subject line and the correspondence template we want to use. It's important to note that "notifications" are meant to be accomplished via email, so the correspondence record should be an email type.

That's it – we'll click OK to effect the changes to the assignment, and, click Save to effect the changes to the flow record



As we mentioned when we started, timely and clear communication keeps participants engaged in the resolution of a case and can help keep the process working efficiently and effectively.

Effective communication can be accomplished by getting answers to three simple questions. First, “who” do we need to communicate with?

Identifying the recipient of any correspondence is fundamental. These recipients can be people directly involved in the resolution of a case,

Or external personnel or agencies with an interest in the case, but no direct involvement.

We model the recipients of our correspondence using a Work Party.

We also need to know “how” to communicate with the recipients. We can craft messages that range from a basic notification – a text-only message that reads “an assignment is ready for you” – to advanced, interactive messages that can even create new work items.

We model the types of correspondence we will need to send using a correspondence record.

Finally, we need to know “when” to send a correspondence.

Using an assignment shape, we can send correspondence automatically when the assignment is reached or manually at the user’s discretion.

Then, using the Email smart shape of the Utility shape, we can send various correspondences automatically.

Remember, regardless of the shape we are using, the correspondence is only sent when that shape is reached in the flow, so the business process requirements drive when we send any given correspondence.

## Exercise: Notifying Users from within a Process



# Enforcing Business Policies Using Decision Rules

In this lesson we discuss how to control the flow of process execution, decision making, and exception management across multiple organizational roles.

At the end of this lesson, you should be able to:

- State the business value of automating business policy decision-making
- Identify three key components for automating decision making
- Identify the steps for delegating business rules



Traditionally, business users are called on to make many decisions based on policies and procedures.

Should this request be approved?

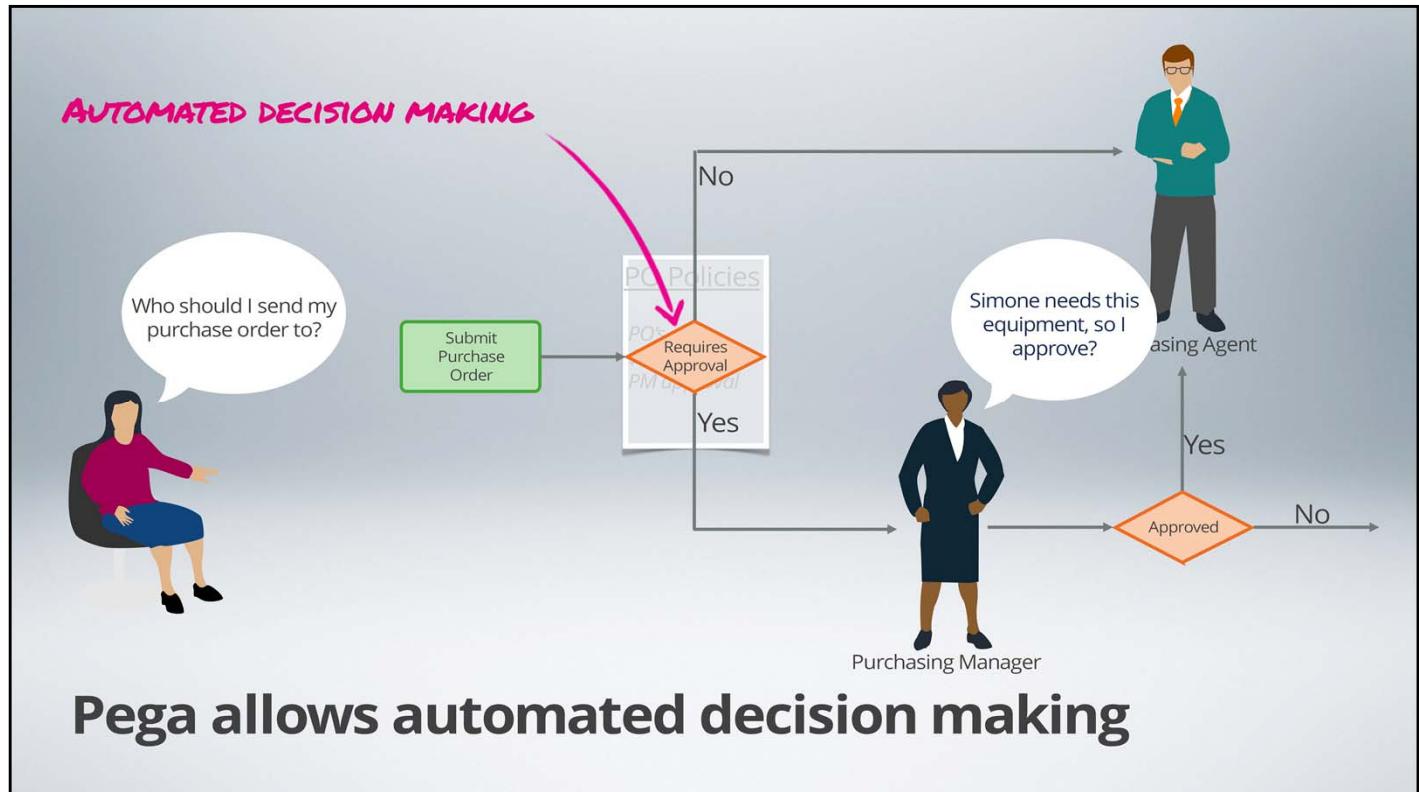
What service should we offer that customer?

What team should handle this issue?



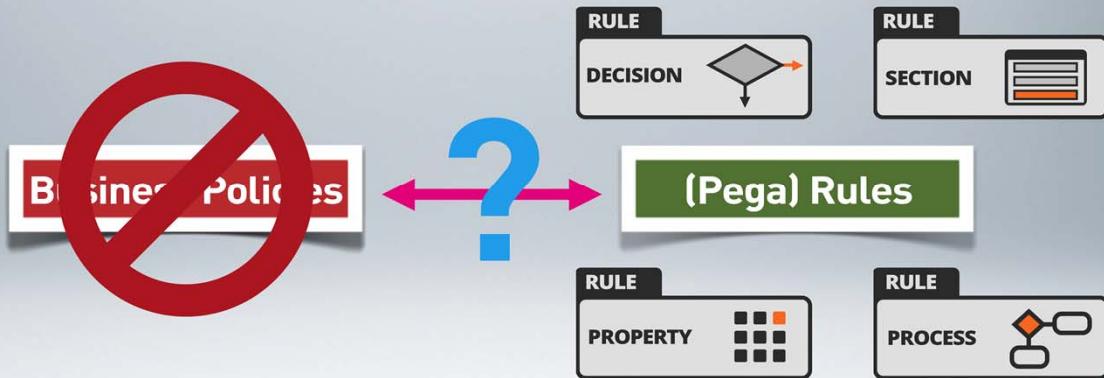
A major feature of any case management solution is automating decision making.

Every company has policies in place  
that govern how decisions are made and who makes them.



Pega allows us to record these policies  
and automate the decision-making that is often done by humans,  
so business users can focus on more nuanced decisions that require human expertise.

## What are "Business Rules"?



**Business policies, procedures and logic are often referred to as "Business Rules"**

Before we continue we should briefly mention terminology. Business analysts often refer to the policies, procedures and business logic that govern how work should be done as "business rules".

This phrase can be confusing in the context of PRPC applications, because Pega uses the term "rules" to refer to the building blocks of the application;

Decisions, sections, properties, processes... these are all types of "rules".

## "Business Rules" refers to (Pega) Rules



**Use "Business Policies" instead to refer to rules that govern decision making.**

To avoid confusion with the term business rules,

Let's consider using the term "business policies" to refer to the rules that govern our decision making. This is the term we use throughout this lesson – and the course – and recommend you adopt in your organization.



These “business policies” we speak of are translated as “decision rules” that answer questions based on available information.

One of the simplest decisions is to answer a basic yes-no question by evaluating one, or more, pieces of information, such as “Is this subscriber’s membership up to date?”

or “Is this item under warranty?”

or “Is this loan amount over eight thousand dollars and for a car more than 5 years old?”

In PRPC, these questions can be answered using a “When” rule.

Use a “When” rule when you know the condition, and simply need to check if it exists.

### SAME FACTORS WITH DIFFERENT VALUES DETERMINE DIFFERENT RESULT

If Order Total	and Department	and In Stock	Approval Required
> \$2000	Major Accounts		Director
> \$1000	Sales	Yes	Director
> \$1000	Sales	No	Manager
\$500-\$1000	Engineering	Yes	None
> \$2000	Engineering	No	Manager

Some decisions are more complex, with many factors to be considered, or many possible outcomes.

When the factors to be evaluated are consistent – such as the example we have here – a “Decision Table” best meets our needs.

We can have any number of results.

The important take away here is that we are evaluating the same set of factors – properties – over and again, but with different values, which determine different outcomes each time.

## FACTORS WITH DIFFERENT VALUES DETERMINE OTHER FACTORS TO EVALUATE



But...how do we handle situations where the value of any given factor defines what other factors are evaluated?

In this case, “Decision Trees” are the most appropriate choice. Decision Trees are similar to decision tables, but can support more complex logic such as nested “If” conditions.

**RULE****WHEN**

- known condition exists or it does not

**RULE****DECISION TABLE**

- same properties, different results

**RULE****DECISION TREE**

- different property values determine other properties to evaluate

Alright, let's summarize.

Use a "When" rule when you know the condition and are just checking to see if it exists.

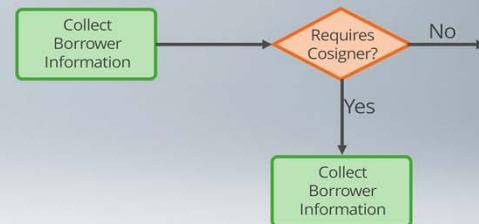
Use a "Decision Table" when the properties to be evaluated are the same, but the values determine the result.

Finally, use a "Decision Tree" when the value of any given property determines what other properties are evaluated.

## Cosigner required when:

### TODAY

If borrower's income is < \$45,000  
and their credit score is < 649



### TOMORROW

If borrower's income is < \$52,000  
and their credit score is < 655

Remember... business policies will change,

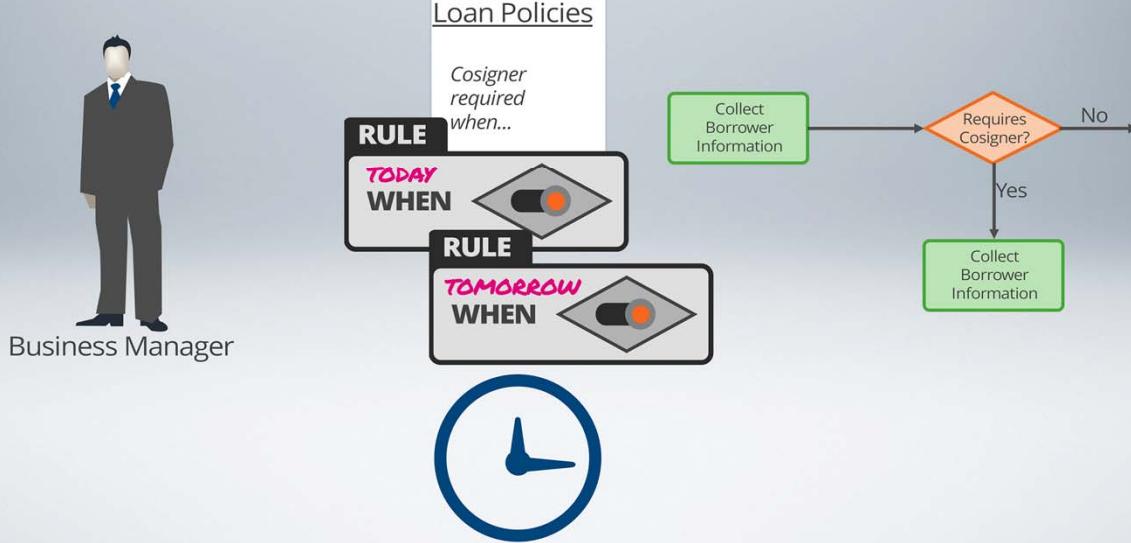
sometimes suddenly, in response to external or internal factors.

We've seen how recording these policies in rules rather than in code allows us to design and implement applications that respond to change with agility and efficiency.

But...making changes to an application takes time – regardless of how agile we are.

Remember, we can delegate responsibility for updating some parts of the application, such as rules that reflect business policy, to business managers.

# Delegate Business Rules to Business Users



## 1 Architect sets up the structure of the rule

Edit When: Requires Cosigner (Available)  
TGB-CardServices-Work-NewAccount • RequiresCosigner | CardServices:01-01-01

Conditions Advanced Parameters Pages & Classes History

When...

Is Student = true  
AND Total Income < 42000.00  
AND Credit Score < 640

## 2 Architect marks the rule as a "favorite" and sets appropriate access criteria

Save Delete Actions Close

Add to favorites

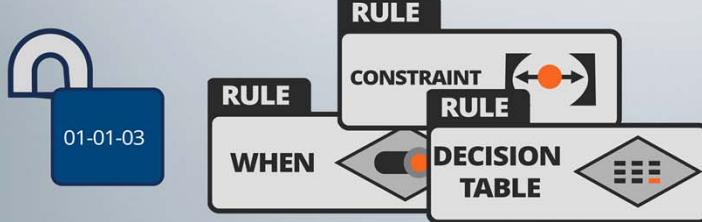
Run Refresh View references View siblings

Favorites for: System Architect  
Label\*: Requires Cosigner  
Add to: My Access Group  
 Open the highest version  
 Always open this version  
OK Cancel

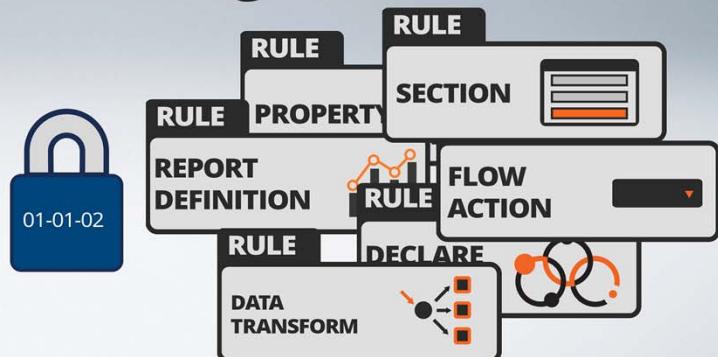
The architect sets up the general structure of the rule to be delegated.

Then, the architect adds the rule as a favorite – and sets the appropriate access criteria.

## Delegated Rules



## Non-Delegated Rules



As a best practice, segregate delegated rules – those expected to change from time to time - from the larger collection of undelegated rules in the application.

The delegated rules are placed in an unlocked RuleSet.

Non-delegated rules should always belong to a locked RuleSet version.

## Exercise: Enforcing Business Policies with Decision Rules



# Enforcing Data Relationships with Declarative Rules

In this lesson we discuss how declarative, decision, and validation rules help automate non-procedural calculations, decision making, and data validation

At the end of this lesson, you should be able to:

- State the business need for declarative processing
- Name and describe the declarative rule types
- Configure a declare expressions

## Declarative Processing

Identify and maintain computational relationships between properties

`.NetWorth = .AssetTotal - .LiabilityTotal`

When `.PostalCode` changes, check if `a flood zone exists`

Do not allow `loans longer than three years` for `vehicles more than five years old`

Declarative processing helps to simplify our application by describing a computational relationship among property values that is expected to be valid “always” or “often” or “as needed.”

For example, we could “declare” the value of a given property to be the sum of other property values.

Or, maybe “declare” that a specified action be performed when a property value changes.

We can even restrict the allowed value of a property based on the value of another property.

In each case, we have one or more input values.

These values are used to determine the value of another property.

## Declarative Processing

**Identify** and maintain computational relationships between properties

**Update** values automatically as inputs change, without the need for explicit programming

<u>Assets:</u>		<u>Liabilities:</u>	
Checking		Credit Card	
Savings		Credit Card	
Investments		Unsecured Loan	
Vehicle		Auto Loan	
<b>Total Assets:</b>	\$0.00	<b>Total Liabilities:</b>	\$0.00
<b>Net Worth</b>	\$0.00		

The primary benefit of declarative processing is that the computations are made automatically, rather than relying upon explicit instructions built into our application.

Let's say we're totaling an individual's assets and liabilities to determine their net worth.

## Declarative Processing

**Identify** and maintain computational relationships between properties

**Update** values automatically as inputs change, without the need for explicit programming

<u>Assets:</u>		<u>Liabilities:</u>	
Checking	\$9,852.32	Credit Card	\$1,235.00
Savings		Credit Card	
Investments		Unsecured Loan	
Vehicle		Auto Loan	
<b>Total Assets:</b>	<b>\$9,852.32</b>	<b>Total Liabilities:</b>	<b>\$1,235.00</b>
<b>Net Worth</b>	<b>\$8,617.32</b>		

Each time we add – or remove – an asset or a liability, or update its value, we want the respective total value and the overall net worth to adjust accordingly.

## Declarative Processing

**Identify** and maintain computational relationships between properties

**Update** values automatically as inputs change, without the need for explicit programming

<u>Assets:</u>		<u>Liabilities:</u>	
Checking	\$9,852.32	Credit Card	\$1,235.00
Savings	<b>\$3,250.00</b>	Credit Card	
Investments	<b>\$52,365.00</b>	Unsecured Loan	
Vehicle	<b>\$45,000.00</b>	Auto Loan	<b>\$39,564.21</b>
<b>Total Assets:</b>	<b>\$110,467.32</b>	<b>Total Liabilities:</b>	<b>\$40,799.21</b>
<b>Net Worth</b>	<b>\$151,266.53</b>		

If we determine these values declaratively, PRPC automatically computes these totals every time we add, remove, or update one of the input values.

## Six ways to manage declarative processing

Type of Declarative Processing	Processing Action

There are six ways we can manage declarative processing. In no particular order,

Do not allow loans longer than three years  
for vehicles more than five years old

Type of Declarative Processing	Processing Action
Constraint	Records an expected relationship between property values

there is a "Constraint." A "Constraint" allows us to define and enforce comparison relationships among property values.

Constraints are tested every time the property's value is "touched", rather than the explicit validation provided by the property definition or validation rules.

Net worth = total value of assets = total value of liabilities

Type of Declarative Processing	Processing Action
Constraint	Records an expected relationship between property values
Declare Expression	Records a computational relationship among property values

“Declare Expression” allows us to define automatic computations of property values based on computational expressions.

This expression may consist of property references, decision rule results, or... even other declare expressions.

## Copy the product list to a clipboard page

Type of Declarative Processing	Processing Action
Constraint	Records an expected relationship between property values
Declare Expression	Records a computational relationship among property values
Data Page	Records what the contents of a clipboard page should be

A “Data Page” creates a clipboard page.

Data pages can improve performance and reduce memory requirements when all or many requestors in an application need to access static information or slowly changing information.

## Increment a counter when a work item changes in value

Type of Declarative Processing	Processing Action
Constraint	Records an expected relationship between property values
Declare Expression	Records a computational relationship among property values
Data Page	Records what the contents of a clipboard page should be
Declare OnChange	Records a computation that must occur when the value of "watched" properties change

Next up is “Declare OnChange.” “Declare OnChange” causes a computation to occur when the value of certain “watched” properties change.

When triggered, a “Declare OnChange” starts an activity that can perform more complex or comprehensive computations than can be defined by expressions.

## Increment a counter when a work item changes in value

Type of Declarative Processing	Processing Action
Constraint	Records an expected relationship between property values
Declare Expression	Records a computational relationship among property values
Data Page	Records what the contents of a clipboard page should be
Declare OnChange	Records a computation that must occur when the value of watched properties change

This may cause more frequent updating than a “Declare Expression,” so we must consider its use carefully.

[Send an email](#) when **saving a customer record** with an updated address

Type of Declarative Processing	Processing Action
Constraint	Records an expected relationship between property values
Declare Expression	Records a computational relationship among property values
Data Page	Records what the contents of a clipboard page should be
Declare OnChange	Records a computation that must occur when the value of “watched” properties change
Declare Trigger	Records processing that must occur automatically when an instance of a specific class is saved or deleted in a database

There is also a “Declare Trigger.” A “Declare Trigger” will cause specified processing to occur automatically when an instance of a specific class is saved or deleted.

For example, we can use a declare trigger to send an email whenever a customer record is updated, perhaps with a changed address or preference.

Copy a **property value** to **another database table** for indexing

Type of Declarative Processing	Processing Action
Constraint	Records an expected relationship between property values
Declare Expression	Records a computational relationship among property values
Data Page	Records what the contents of a clipboard page should be
Declare OnChange	Records a computation that must occur when the value of "watched" properties change
Declare Trigger	Records processing that must occur automatically when an instance of a specific class is saved or deleted in a database
Declare Index	Records criteria for automatically maintaining index instances for faster access

Finally, there is “Declare Index.” “Declare Index” is used to define criteria under which PRPC automatically maintains index instances for faster access.

An index can improve search and reporting access for properties that cannot be exposed as database columns because they are embedded within an aggregate property.

## Declarative Processing

**Identify** and maintain computational relationships between properties

**Update** values automatically as inputs change, without the need for explicit programming

**Declarative rules** are never referenced

Do not allow loans longer than three years for vehicles more than five years old

Net worth = total value of assets = total value of liabilities

Copy the product list to a clipboard page

Increment a counter when a work item changes in value

Send an email when saving a customer record with an updated address

Copy a property value to another database table for indexing

As we close this topic, it is important to note that, in general, no other rules explicitly reference declarative rules.

Your use of a property that's referenced in an expression or constraint rule causes the expression or constraint to execute.

## Demo



### Configure A Declare Expression

Let's have a look at how the line item subtotal and purchase request total is calculated in the purchase application.

The subtotal is calculated using a declare expression, which is in the line item class. The value is calculated whenever the inputs change, which in this case is the quantity and unit price. The expression is executed in the context of purchase requests and purchase orders.

The purchase request total is calculated in a similar way. It is also calculated whenever the input changes. The total is set to zero if the purchase request gets resolved as rejected or withdrawn. Otherwise the total is set to the sum of the line items subtotal.

## Exercise: Automatically Calculating Property Values



# Guardrails for Enforcing Business Policies

This lesson discusses guardrail warnings you may encounter when enforcing business policies, in addition to best practices for enforcing business policies.

At the end of this lesson, you should be able to:

- Identify common guardrail warnings encountered when enforcing business policies
- Identify motivations for establishing best practices as additional guardrails for enforcing business policies
- Provide a summary of best practices for enforcing business policies

One has to put extra effort into violating guardrails when creating components for enforcing business policies.



But...let's look at a few of the guardrails that could be violated.

One has to put extra effort into violating guardrails when creating components for enforcing business policies. But...let's look at a few of the guardrails that could be violated.

**Avoid Inline Styles**

**AVOID JUSTIFYING WARNING**

**Caution - Best Practice**

**BestPractice: Caution**

Styles or Scripts should not be included as part of the correspondence text. They should be moved to a separate, included fragment.

**Add Justification**

**INCONSISTENT LOOK AND FEEL**

**CHANGES ARE TIME CONSUMING**

**USE RICH TEXT EDITOR**

**ADVANCED FORMATTING OPTIONS AVAILABLE**

The slide features a yellow triangle with an exclamation mark inside, labeled "Caution - Best Practice". A pink handwritten note "AVOID JUSTIFYING WARNING" is written across the top right. Below the note, a callout box contains the text "BestPractice: Caution" and "Styles or Scripts should not be included as part of the correspondence text. They should be moved to a separate, included fragment." with a link "Add Justification". To the right of the callout box, four pink handwritten notes are listed: "INCONSISTENT LOOK AND FEEL", "CHANGES ARE TIME CONSUMING", "USE RICH TEXT EDITOR", and "ADVANCED FORMATTING OPTIONS AVAILABLE". At the bottom, there is a screenshot of a rich text editor interface showing some sample code.

When creating correspondence records, using inline styles or scripts will prompt a best practice caution warning.

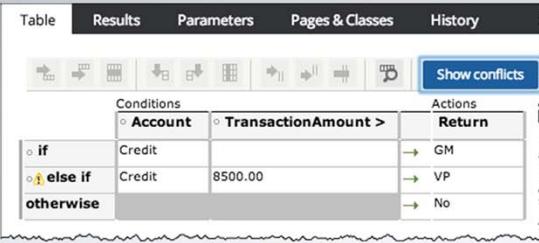
Including HTML styles inline makes it difficult to maintain consistency over time.

And maintenance becomes much more difficult and time consuming.

Consider using the rich text editor to format correspondence.

However, you can also store the HTML styles in a “text file” record and then include that text file record in the correspondence text,

so there should never be a reason to justify this warning.



**Moderate - Best Practice**

# Logic Conflict

**NEVER JUSTIFY WARNING**

Logic: Moderate  
This rule contains 1 logic conflict. Click Show Conflicts to see details.  
[Add Justification](#)

**DECISION MAY NOT WORK AS INTENDED**

**USE "SHOW CONFLICTS" TO HELP RESOLVE**

When building decision tables and trees, it's possible to create a logic conflict where a subsequent row is unreachable.

This warning is an alert that the decision record may not work as intended.

To bring the record into conformance, identify the conflicting rows using the "Show Conflicts" button.

Given that the conflicts may produce unintended or, worse, inaccurate, results, never justify this warning.

**RULE**

**SERVICE LEVEL**

**RULE**

**WHEN**

Use most appropriate rule

**RULE**

**DECISION TABLE**

Rules with greatest volatility

Is Cosigner Required

Short Description provides business context

Collaboration and Understanding

DOCUMENTATION

Description

This table defines our policies for when approval is required for a given purchase order

Usage

Used in the Purchase Order case type to automate decision-making for purchase order approvals.

Table Results Parameters Pages & Classes History

Document all records

Remember, to be most effective at building for change, we must be prepared. As always, establishing best practices when building rules for enforcing business policies goes a long way towards that effort.

Choose the most appropriate rule that best represents the business logic.

Work closely with the business users to identify those rules they are most likely to understand – and have the greatest need to change more often than standard production release cycles might allow for.

When naming rules, make sure the short description provides a meaningful business context.

Finally, thoroughly document the business rules using the “History” tab.

## Module 08: Process Visibility through Business Reporting

This lesson group includes the following lessons:

- Preparing Your Data for Reporting
- Building Business Reports
- Guardrails for Business Reporting

# Preparing Your Data for Reporting

In this lesson you will discuss the need for optimizing your properties to improve performance of your reports.

At the end of this lesson, you should be able to:

- Discuss the difference between business data and process data used in reports.
- Identify why properties need to be optimized for reports.
- Optimize a property needed for a report

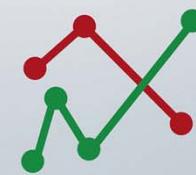


The real purpose of reports is insight.

You need ways of understanding how complex processes are functioning — where the bottlenecks are, where there are opportunities to improve response time, and what emerging trends need attention.

A report that asks the correct questions, and therefore provides us with relevant information rather than an unsorted heap of data, can show us what's going on now, what has been going on over a period of time, or how what is going on matches or differs from what was planned.

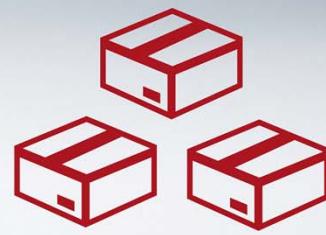
Anyone involved in your business processes, from the developers through the managers to the customer service representatives, will find occasions when the information in a PRPC report will be very valuable.



Business Metrics = Company Data



Orders Processed



Orders Cancelled

There are two types of metrics associated with report data, business metrics and process metrics.

For example, the number of orders processed or how many orders get canceled. Put more simply, business metrics represent the data that YOU defined when you created the properties.

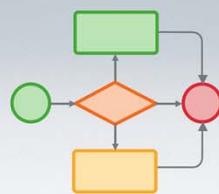


Process metrics (or statistics) are tracked by Pega and includes:

Process Data = Statistics



Time to complete an assignment



How often a path is followed



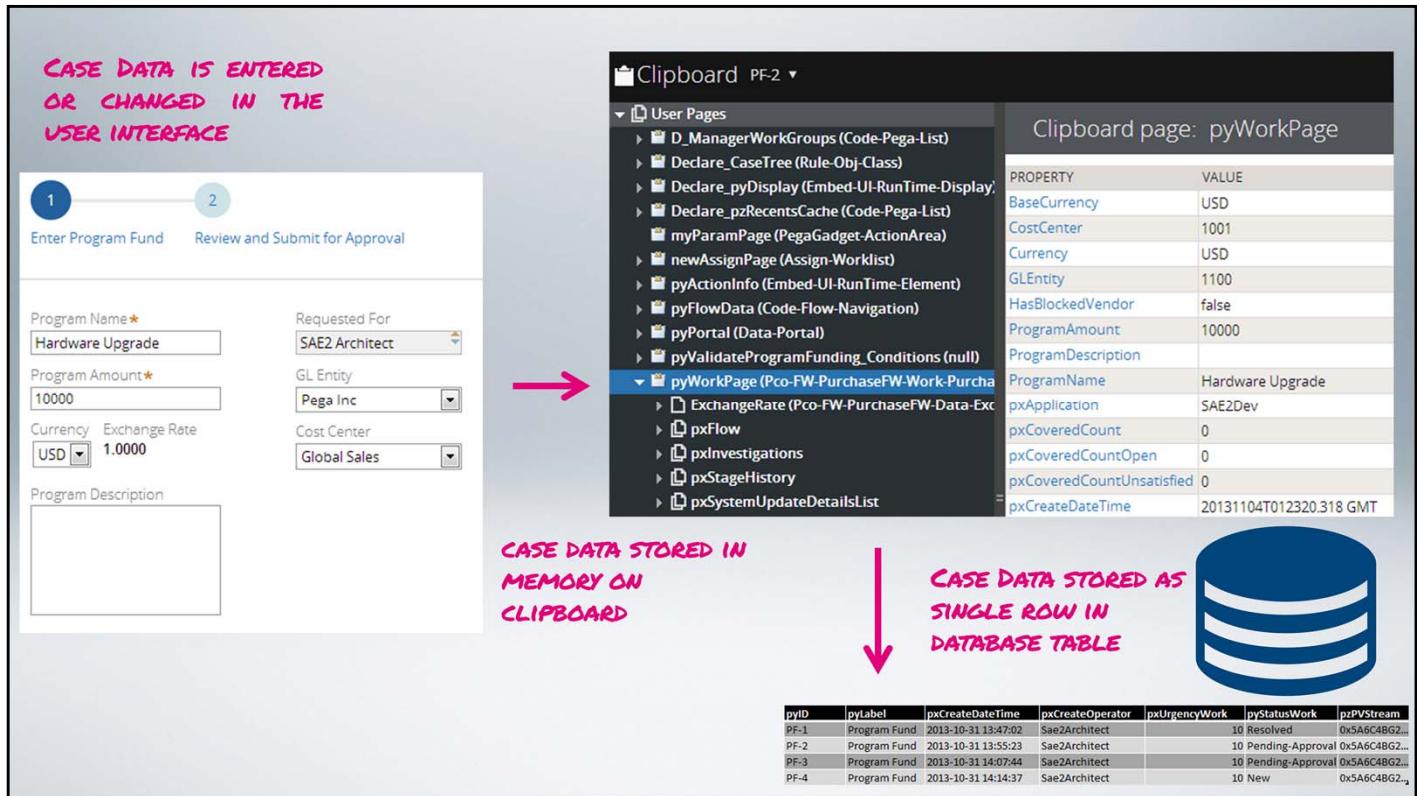
Urgency: 80

SLA violations

#### Process metrics - or statistics

- are tracked by Pega and include how long it takes to complete an assignment, how often a path is followed in a flow or how often Service Level Agreements (or SLAs) are violated.

Understanding the difference between business and process metrics is crucial to building effective reports.



The data entered or displayed in applications

are initially stored in memory on what's known as the clipboard (which is not visible to the end user).

When an end user clicks OK or Submit to save changes, the data is then written to the database. A case is written as a single row within a table in the PRPC database.

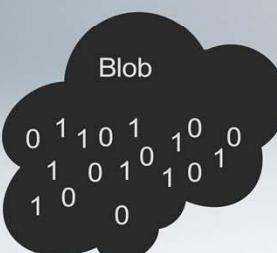
However, the data model for a case may be quite large and complex, including multiple levels of hierarchically embedded sets of fields (PRPC refers to these as pages, page lists, and page groups).

Most of the database operations that PRPC performs (opening a case from a worklist or workbasket, saving changes to a case, reading a business rule, etc.) are optimized through PRPC's architecture which uses a single binary large object (BLOB) field within the record to store all of the data for the case.

pyID	pyLabel	pxCreateDateTime	pxCreateOperator	pxUrgencyWork	pyStatusWork	pzPVStream
PF-1	Program Fund	2013-10-31 13:47:02	Sae2Architect	10	Resolved	0x5A6C4BG2...
PF-2	Program Fund	2013-10-31 13:55:23	Sae2Architect	10	Pending-Approval	0x5A6C4BG2...
PF-3	Program Fund	2013-10-31 14:07:44	Sae2Architect	10	Pending-Approval	0x5A6C4BG2...
PF-4	Program Fund	2013-10-31 14:14:37	Sae2Architect	10	New	0x5A6C4BG2...

**BLOB FIELD**

BLOB field has no size constraints



BLOB data model is flexible

Optimized for transaction performance

BLOB not optimal for reporting

The use of a BLOB field to store (in a proprietary compressed form) all of the data for a case offers the following powerful advantages: There are no physical size constraints on BLOB fields, so they can hold any amount of information.

BLOBS deliver high performance. It is faster to retrieve a case and place all of its information in memory on the clipboard for use by an application, and also faster to save changes to that information. Instead of having to retrieve data from multiple tables using joins, and then perform complex object-relational mapping of the retrieved information to place it into memory on the clipboard, PRPC simply reads a single record out of the database, decompresses the BLOB field value and loads the data, which is already structured in the format required by the rest of the application, into memory and onto the clipboard.

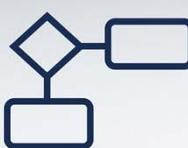
The use of the BLOB to store case data is highly optimized for performance and flexibility in operations on individual cases. However, reports (and other list-centric activities such as displaying work lists and work queues, finding open items by Customer or Policy ID, etc.) retrieve not the entire contents of the BLOB, but only specific properties from multiple cases through database queries using the SQL language.

PRPC reports generate the appropriate SQL queries automatically for the user, based on their description of the desired report. Retrieving specific properties directly from the BLOB for many instances can be inefficient. Because of this, PRPC offers a hybrid data storage model in which data can be stored as relational database columns as well as in the BLOB, offering the best of both approaches.

Properties used for reporting can be added as database columns to the class table. These additional columns are also known as **exposed columns**.

**OPTIMIZED PROPERTY**

pyID	pyLabel	pxCreateDateTime	pxCreateOperator	pxUrgencyWork	pyStatusWork	pzPVStream	ProgramAmount
PF-1	Program Fund	2013-10-31 13:47:02	Sae2Architect		10 Resolved	0x5A6C4BG2..	500
PF-2	Program Fund	2013-10-31 13:55:23	Sae2Architect		10 Pending-Approval	0x5A6C4BG2..	10000
PF-3	Program Fund	2013-10-31 14:07:44	Sae2Architect		10 Pending-Approval	0x5A6C4BG2..	6283
PF-4	Program Fund	2013-10-31 14:14:37	Sae2Architect		10 New	0x5A6C4BG2..	1200



Process Metrics

Properties used for reporting can be added as database columns to the class table; these additional columns are also known as exposed columns, as they expose the properties found in the BLOB into an ordinary column in the table, so that normal SQL instructions can be performed on them. These exposed columns are updated in addition to the BLOB each time an instance update is committed to the database. Notice these optimized properties are stored in their own column. This can become problematic when more and more properties get optimized so it's possible to see separate tables - called declare indexes - created for this reason.

Additionally notice in this table there are several process metrics that are already being tracked, values for who created the case, the urgency for the case, and the current status are all automatically tracked.

It is very important that when optimizing properties that you consult with the entire development team and have a plan as to which properties should be optimized.

## Demo



# Optimizing Properties

Let's take a look at how we optimize a property for reporting. In the Application Explorer let's take a look at the PurchaseOrder class.

Now let's optimize the POTotal property. Remember identifying the properties that need optimizing is an organizational effort that includes lots of stakeholders. For this demo those conversations have already occurred and it was decided to optimize the POTotal property.

Right click the POTotal and choose Optimize for reporting.

We now go through the Property Optimization wizard to optimize the POTotal property. First, we specify which POTotal property we want to optimize.

Next we get a list of eligible classes, tables and databases for the property we selected in the previous screen.

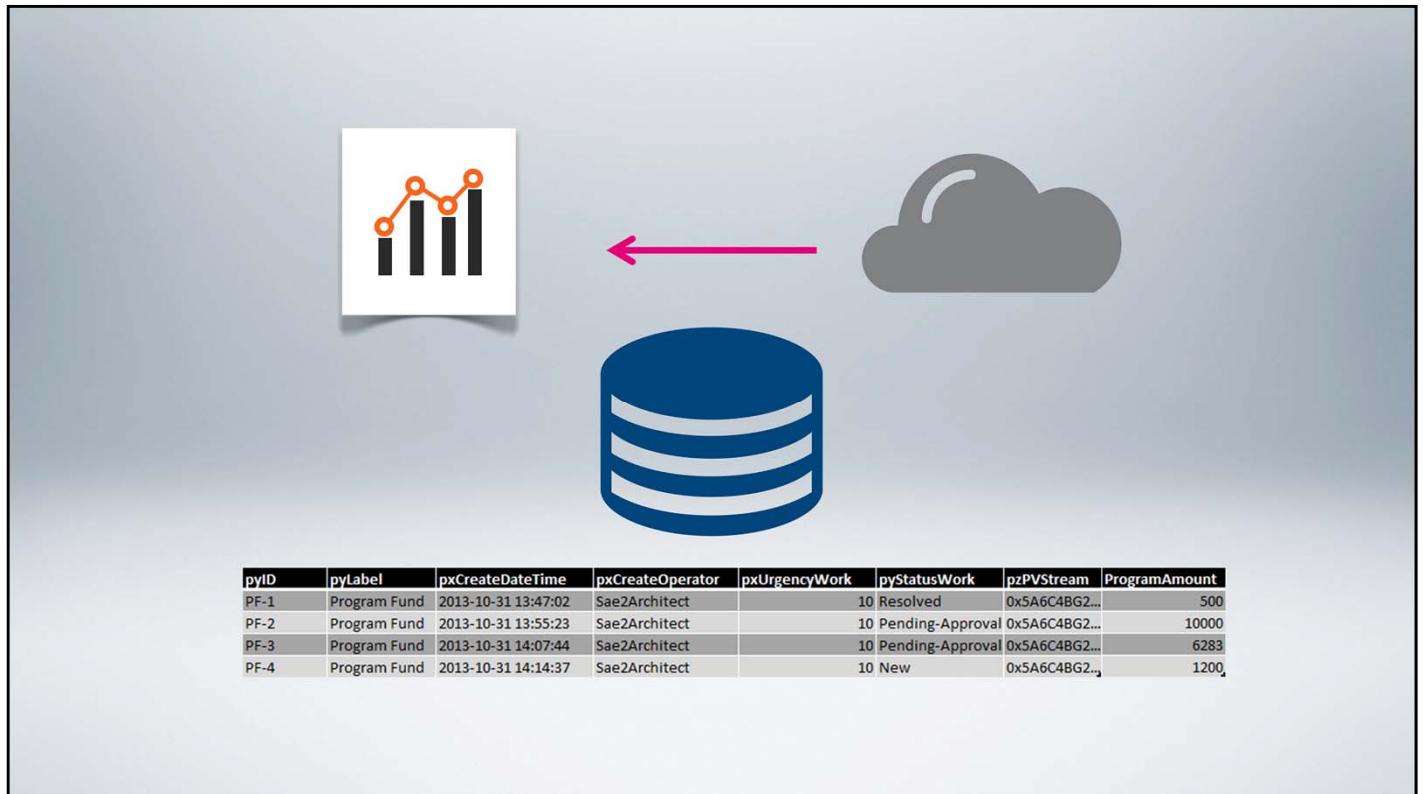
Lastly the wizard shows us the number of existing instances that will have to have the POTotal property populated. In this scenario, we do not have any existing instances but imagine if there were 10 existing purchase orders. After the POTotal column is created, the value for that column would have to be populated for each of the existing 10 rows. Click Finish to complete the process.

Let's confirm that in the POTotal property we can see that it is flagged for optimization.

This can be determined by looking in the Advanced tab.

We may need to refresh the property to see that the property is marked for optimization.

After refreshing there is a new value for the classes Optimized for property.



Reports offer us insight into the data contained within our application. There are actually two types of metrics, business metrics that refer to data specific to the company's case performance and process metrics that show us statistics about our application.

This data is stored as a row of data in an internal Pega database. Inside the row our data is stored as a BLOB. This allows for better transaction performance and also allows us to have a flexible data model as the database structure doesn't need to change.

This however is not ideal for reporting. In order to improve performance of reports we can select properties to optimize. By optimizing properties we add columns to the table and Pega makes sure that the values contained within the BLOB match the value in the column.

## Exercise: Optimizing Properties for Reporting



# Building Business Reports

In this lesson we will discuss how to display the data collected by your application can be shown and categorized in a meaningful way.

At the end of this lesson, you should be able to:

- Name the different report types
- Explain the differences between the different report types



We have previously discussed report definitions in the context of defining a set of data that is used to populate data pages. Report definitions are also used to create reports on both business and process metrics. There are two types of reports that are available list reports and summary reports. Report Definitions allow for both summary and list views within the same rule.



## List Reports = Spreadsheet style reports

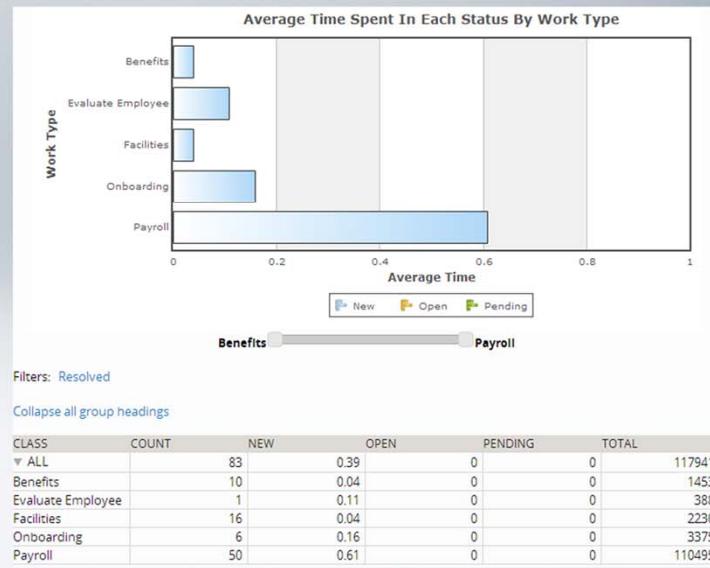
can combine business and process metrics

ID	WORK STATUS	VENDOR ID	VENDOR NAME	PAYMENTTOTAL
PO-4	Pending-Purchasing	Amazon		
PO-2	Pending-Purchasing	Amazon		
PO-3	Resolved-Completed	Amazon		30.95
PO-1	Pending-Purchasing	Amazon		

List reports display requested information in spreadsheet style. On each row of the report detailed information about an individual case, status, product, or other type of data. For example, a user may ask for a list purchase orders by vendor. List reports are what we used earlier to source our data pages. We configure what properties we want to see in the report and they show in the columns of a table. Notice that we can combine both business and process metrics in a report. In the purchase order by vendor example we have business metrics such as Vendor ID, Vendor Name and Payment Total along with process metrics such as ID and Work Status.



## Summary Reports = Synopsis



Summary reports show on each row of the report; counts, totals, averages, or other data summarized from multiple cases or work items. For example, a user may ask for a summary of cases by the operator who created them, based on whether the case was resolved in a timely manner.

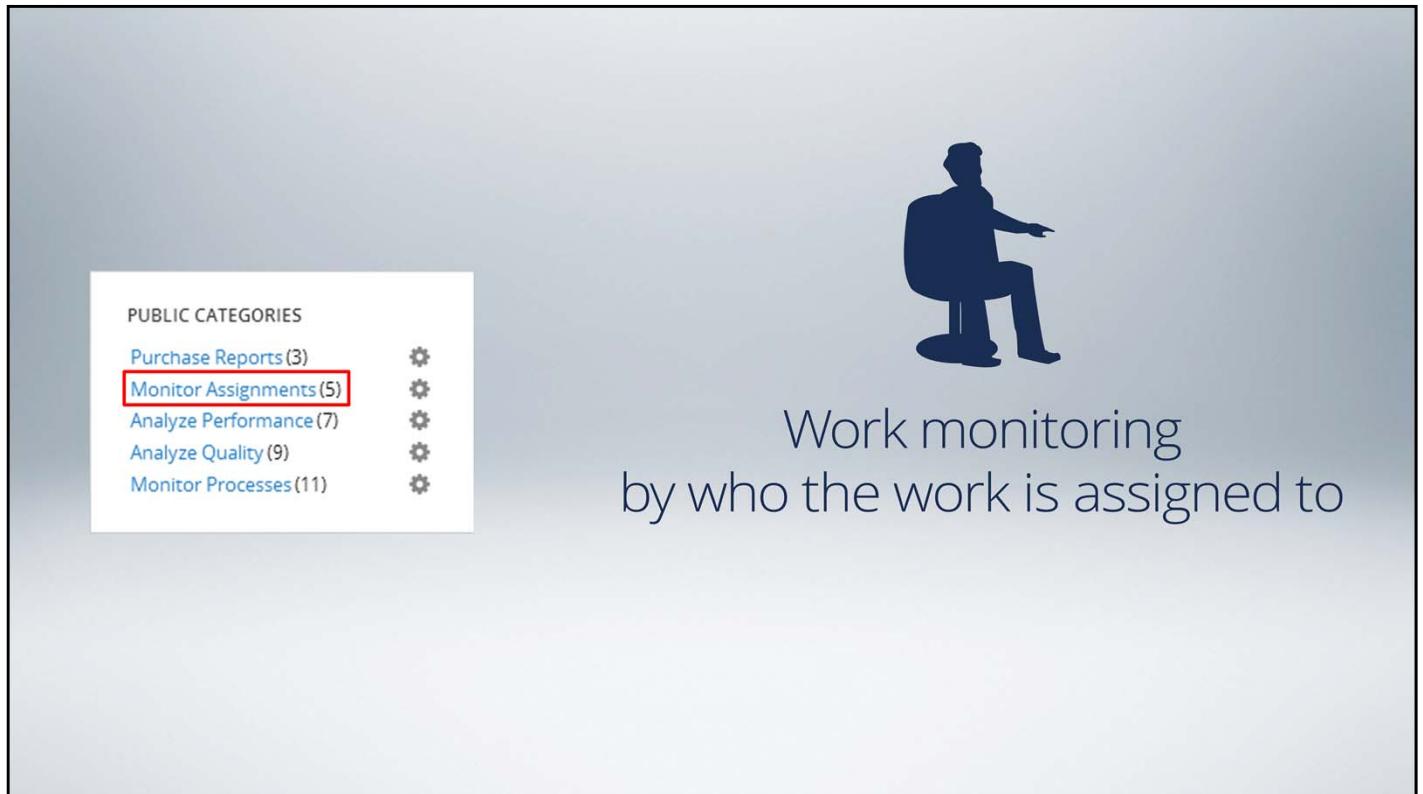
Summary reports are often used to show trends over time. Trend reports show summarized data over time (hours, days, weeks, months, etc.).

As these examples show, summary reports may also use a chart or graph to display the summarized information. Available chart types include bar charts, column charts, line charts, pie charts, bubble charts, and gauges. Line charts are usually used for trend reports.

# Report Browser

The screenshot shows the Report Browser interface. At the top, there's a navigation bar with tabs for 'Recent reports' and 'All reports'. A search bar labeled 'Search reports' is also present. Below the navigation bar, there are sections for 'TITLE' and 'EMAIL NOTIFICATION'. The 'TITLE' section lists several report titles with icons: 'Average elapsed time per status' (bar chart), 'List of processes entered by operator' (list icon), 'Average elapsed time per status' (bar chart), 'Average Performance Time By Task' (bar chart), 'Timeliness by work type' (bar chart), and 'Average elapsed time per status' (bar chart). To the right of these titles is a column of six gear icons. On the far right, there are two sections: 'PRIVATE CATEGORIES' which says 'No items', and 'PUBLIC CATEGORIES' which lists 'Monitor Assignments (5)', 'Analyze Performance (7)', 'Analyze Quality (8)', and 'Monitor Processes (11)'.

The Report Browser allows users to browse, search, organize, and run existing reports. It allows users to access and run any kind of reports. It lets users easily create their own reports, and share them with other users. Reports can be categorized and searched to help people find the report they are looking for. You can also filter on the different report types.



The image shows a software interface with a sidebar on the left and a main content area on the right. The sidebar is titled "PUBLIC CATEGORIES" and lists the following items:

- Purchase Reports (3)
- Monitor Assignments (5)** (This item is highlighted with a red border.)
- Analyze Performance (7)
- Analyze Quality (9)
- Monitor Processes (11)

Next to each category name is a small gear icon. The main content area features a large blue silhouette of a person sitting at a desk, facing right. To the right of the silhouette, the text "Work monitoring by who the work is assigned to" is displayed.

Pega provides hundreds of out-of-the-box reports and puts them in categories. Many of these are designed for use by developers and show information about the design of the application. Many of them are designed for use by business analysts and managers, to help them monitor and analyze their business processes.

There are four categories of reports. The first is Monitor Assignments, these reports display information about assignments for open (unresolved) work items in an application. They support ongoing monitoring of work from the perspective of who the work is assigned to. For example, a report that measures timeliness by task.

PUBLIC CATEGORIES

- [Purchase Reports \(3\)](#)
- [Monitor Assignments \(5\)](#)
- [Analyze Performance \(7\)](#)
- [Analyze Quality \(9\)](#)
- [Monitor Processes \(11\)](#)



Granular analysis of completed work

The next category is Analyze Performance, these reports display information about resolved work items in an application at the level of individual steps, or flow actions, within business processes. They support analyses of completed work to determine whether business processes were efficiently and effectively performed, as a basis for improving business processes. For example, a report that measures the average processing time in hours by task and flow action.

The screenshot shows a software interface with a sidebar titled "PUBLIC CATEGORIES". The categories listed are: Purchase Reports (3), Monitor Assignments (5), Analyze Performance (7), **Analyze Quality (9)**, and Monitor Processes (11). The "Analyze Quality" category is highlighted with a red border. To the right of the sidebar, there is a large magnifying glass icon with a green checkmark inside it. Below the magnifying glass, the text "Application analysis of completed work" is displayed.

The next category is Analyze Quality, these reports display information about resolved work items in an application. They support analyses of completed work to determine whether business processes were efficiently and effectively performed, as a basis for improving business processes. For example, a report measures the average elapsed time per status.

PUBLIC CATEGORIES

- [Purchase Reports \(3\)](#)
- [Monitor Assignments \(5\)](#)
- [Analyze Performance \(7\)](#)
- [Analyze Quality \(9\)](#)
- [Monitor Processes \(11\)](#)



Monitoring of work by the work being performed

The last category is Monitor Processes, these reports display information about assignments for open (unresolved) work items in an application. They support ongoing monitoring of work from the perspective of the work being performed. For example, a report that measures throughput in past week by work type.

## Demo



## Creating a Summary Report

In this demonstration we need to build a report that displays a chart and shows the total cost for each cost center manager. When creating a report definition it is important to create the report in the same class that contains the properties you want to use. In our case that is in the PurchaseRequest class.

There are a couple of different ways to create a report, the simplest is to expand Reports and right click Report Definition. We need to give the report definition a short description which will also be the default title of the report. Then we create the report. Next we add the column names that we want to see in the report. There are three columns of interest to us, the CostCenterManager, the TotalPRCost and the pyStatusWork, which gives us the status of the purchase request. Then we'll save our report.

When saved you will notice a warning occurs. This is because the CostCenterManager property has not been optimized for reporting. It is still OK to use an unoptimized property just realize that it could impact performance depending on how often this report is used.

To continue editing we need to check out the rule.

In order to have a chart, we need to have at least one of our fields aggregate some data. For our report that is the TotalPRCost column, under summarize we can choose one operation, which for our report is SUM.

We also have the ability to customize the sort order for our report. For our report we want to have CostCenterManager first, pyStatusWork second and TotalPRCost third.

It's time to create a chart, to do so we click the chart tab.

We can choose from various chart types that are shipped with the product. For our application let's use a

pie chart.

After selecting a chart we can choose which fields we want to use to determine what the chart shows. We can drag the available columns to the X and Y axis of the chart.

Let's look at some of the additional options we have for any report on the Report Viewer tab. On the Report Viewer tab there are a bunch of options for reporting, we can alter the title and we can control how the report is viewed. We can also select whether or not the report should appear in the report browser. When we select our report to display in the report browser we are then required to choose which category we want to put our report in. Our report is complete, let's save our changes and test it.

There are a couple of ways to test our report, the first is to run the report from the Actions tab on the rule form.

We can see our pie chart and below we see a summary of all the records that made up the report. Now that our report looks how we want it, we can check in the result.

We can also view our report from the Report Browser. Remember we put our report in the Purchase Reports category. We can click on our report to view it.

Once again our report shows up with the same information.

We've created a summary report that shows which managers have the highest amount of purchases.

## Exercise: Create Reports



# Guardrails for Business Reporting

This lesson reviews guardrail warnings and best practices that you may encounter when creating reports.

At the end of this lesson, you should be able to:

- State two report definition guardrail warnings
- State three questions to consider when creating reports

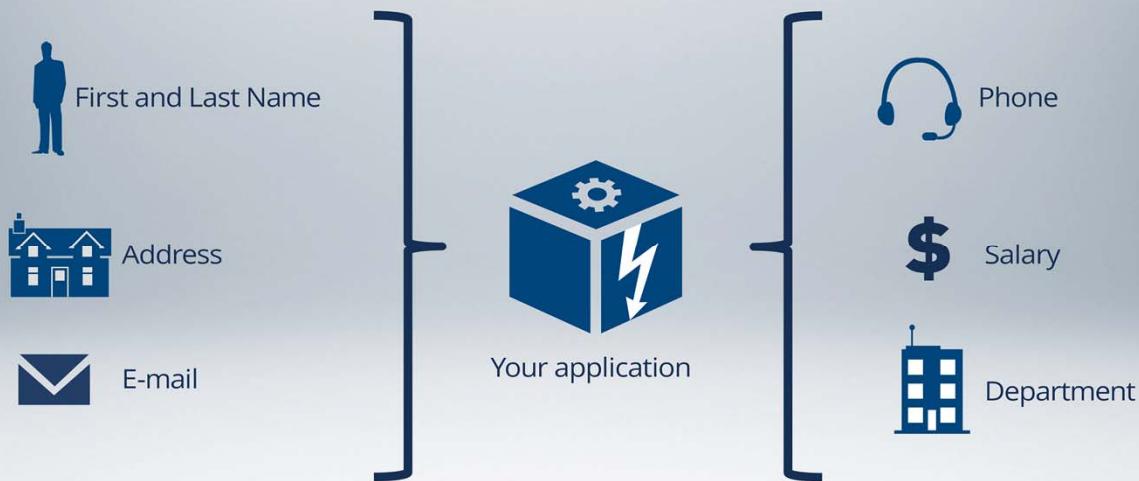
## Unexposed Property



pyID	pyLabel	pxCreateDateTime	pxCreateOperator	pxUrgencyWork	pyStatusWork	pzPVStream
PF-1	Program Fund	2013-10-31 13:47:02	Sae2Architect		10 Resolved	0x5A6C4BG2...
PF-2	Program Fund	2013-10-31 13:55:23	Sae2Architect		10 Pending-Approval	0x5A6C4BG2...
PF-3	Program Fund	2013-10-31 14:07:44	Sae2Architect		10 Pending-Approval	0x5A6C4BG2...
PF-4	Program Fund	2013-10-31 14:14:37	Sae2Architect		10 New	0x5A6C4BG2...

The most common guardrail warning you will encounter when working with report definitions is that you have used an unexposed property. This warning occurs when you use a property in your report definition that has used properties that have not been optimized. While many out of the box properties are already exposed (optimized).

## Determine appropriate properties to optimize



Based on the reporting needs of the business, the lead architect along with the database administrator should determine which properties are optimized. In addition to optimizing properties, the database administrator will also want to add/modify any database indexes to ensure optimal performance. Exposing all properties adds more overhead and causes issues with your application's performance. Exposing no properties wouldn't necessarily affect your application's performance but does affect the performance of your reports.

## Format on DateTime property



Display as configured  
May 5, 2:37

calculations  
group by  
sorting  
charting



Unformatted  
value

Another warning you could see is the Format on DateTime property. This warning appears when a column in the Columns To Include area on the report definition's Design tab uses a property with a DateTime type and a control is specified in that column's Format Values field. The warning is an alert that even though the system displays the column in the specified format any calculations, grouping, sorting, and charting use the unformatted values. The warning text provides the name of the property (or properties) and the specified formats.



When creating reports it's important to follow a Think, Plan and Iterate process. When thinking about a report ask yourself the following questions. What data will interest report consumers? Who will use the report? Are there others who may want to use a similar report? How are report consumers organized? Are there any security considerations? As a best practice, reporting requirements often affect the design of the data model so we should mention that this should be taken into account when setting up your data model.



Think



Plan



Iterate

- What data types should be covered?
- What information should appear?
- What sort of chart would be useful?
- Is there an existing report you can use as a starting point?

When planning your report ask yourself the following questions:

What data types should be covered?

What information should appear?

What sort of chart would be useful?

Is there an existing report you can use as a starting point?



Think



Plan



Iterate

Build - run - review - improve - build again

Lastly, it's important that you continually iterate over the report to make sure it is presenting the information needed. This should involve the business users that will use the report when development is done. The goal is to build the report, run it, review it, find areas for improvement, then build it again. This helps to create successful reports that provide end users with the information they want to see.

## Exercise: Process Visibility with Reports Verification Exercise



## Module 09: Best Practices for Preparing an Application for Testing Deployment

This lesson group includes the following lessons:

- Using Guardrail Reports to Ensure the Best Performance
- Guidelines for Maintaining Requirements and Specifications

# Using Guardrail Reports to Ensure the Best Performance

In this lesson we look at guardrail warnings and best practices that can affect the performance of your application.

At the end of this lesson, you should be able to:

- Identify two guardrail warnings that can impact an application's performance.
- Identify application development best practices



We have discussed many times that guardrails are best practices and guidance about situations that contain risky conditions or that might result in an undesirable outcome.

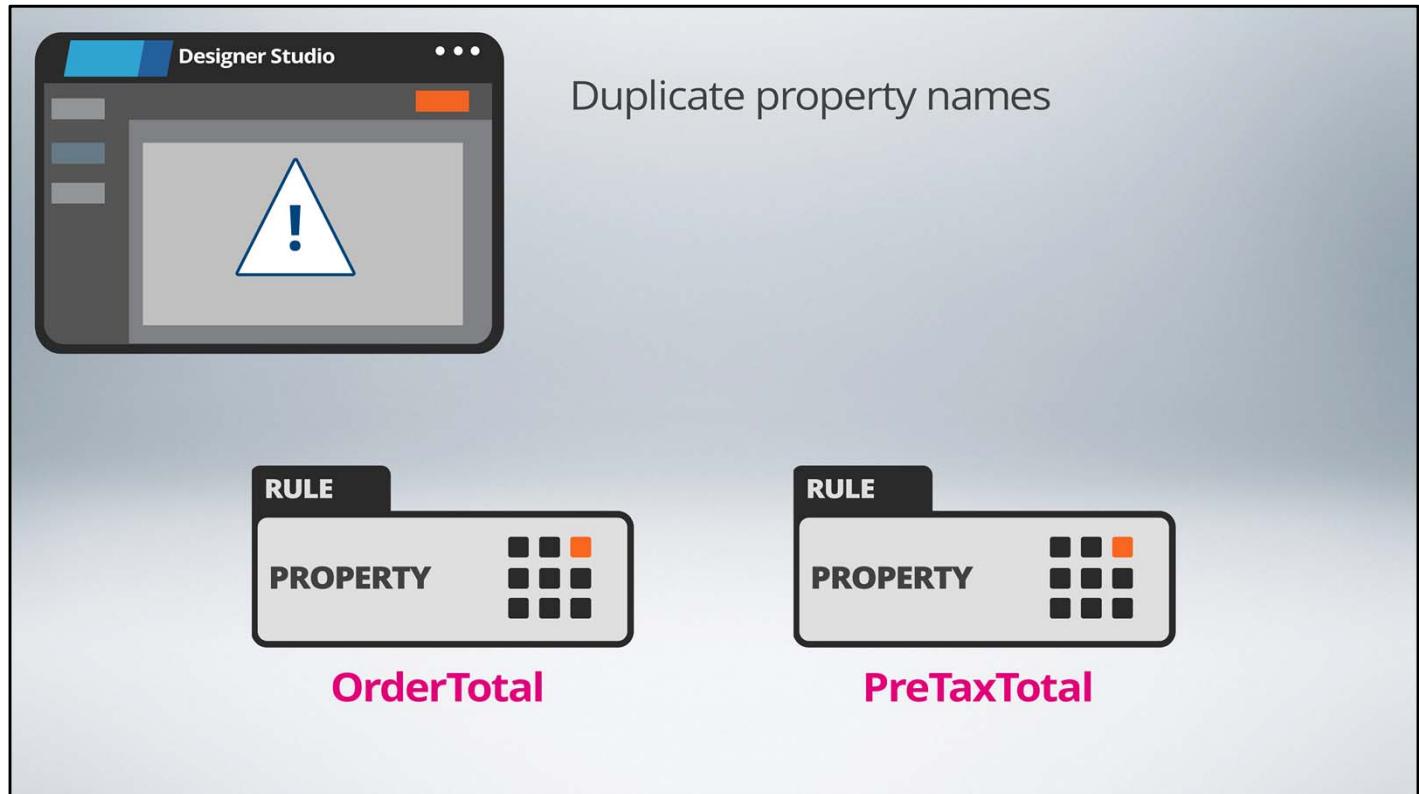
Every guardrail warning consists of a severity and a warning type.

Pega professionals identify guardrails from developing, testing, and reviewing Pega 7 applications in a variety of industries and organizational settings. Guardrails ensure you and your team use Pega 7 according to their guidance, and help you avoid troublesome situations.

While implementing an application you should continuously check the guardrail landing page for warnings in your application. Some guardrails need to be resolved before you can deploy your application. Some examples of these include:



The duplicate property warning appears when a property of the same name already exists in the class inheritance path. Properties that are duplicates of other property instances have an effect on caching, which can have a negative effect on system performance. As a best practice for good runtime performance, choose property names that are distinct and unique throughout your application.





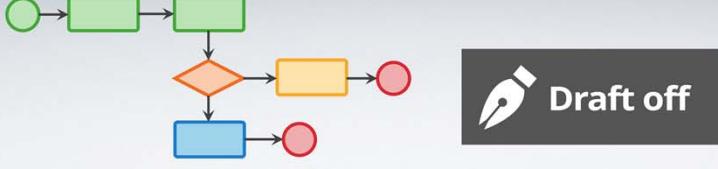
Duplicate property names  
Optimized properties for reporting



We also discussed having non-optimized properties used in your reports. This can adversely effect performance because when the report is run the entire BLOB has to be unpacked to access the property that you use. The amount this could effect performance depends on a number of factors such as how often that report will actually run and how many records are in the underlying tables.



Duplicate property names  
Optimized properties for reporting  
Turn off Draft Mode for flows



```
graph LR; Start(( )) --> A[ ]; A --> B[ ]; B --> C[ ]; C --> Decision{ }; Decision --> Path1[ ]; Decision --> Path2[ ]; Path1 --> End1(( )); Path2 --> Path3[ ]; Path3 --> End2(( ));
```

A flowchart diagram illustrating draft mode behavior. It starts with a green oval, followed by three green rectangular boxes connected by arrows. An orange diamond decision box follows. From the diamond, two paths emerge: one leading to a red circle and another leading to a blue rectangular box, which then leads to a red circle. To the right of the flowchart is a dark grey button with a white pen icon and the text "Draft off".

Another thing to make sure you do is to turn off draft mode for all flows you've created. Remember that when running in draft mode you are allowed to access rules that might not exist yet, this is OK in development but once you go to production flows that are in draft mode are ignored.



You should use the guardrail report to guide you in what needs to be done before you deploy your application. Before you deploy all guardrails should be either resolved or justified as to why they are not resolved.

Don't duplicate logic

**RULE**

**DATA TRANSFORM**

map a  
map b  
copy  
map c

**RULE**

**DATA TRANSFORM**

map a  
map b  
copy  
map d

In addition to resolving guardrail warnings, while implementing your application there are some general best practices that you should follow.

Don't duplicate logic.

If the same set of steps is used in two places,

capture the steps in a single data transform and call that record from both places where the steps are needed.



Don't duplicate logic

**RULE**

DATA  
TRANSFORM



map c

**RULE**

DATA  
TRANSFORM



map a

map b

copy

**RULE**

DATA  
TRANSFORM



map d



## Don't duplicate logic

First name	Last name
<input type="text"/>	<input type="text"/>
Phone	City
<input type="text"/>	<input type="text"/>

Submit

First name	Last name
<input type="text"/>	<input type="text"/>
Email	Manager
<input type="text"/>	<input type="text"/>

Submit

Similarly, if the same UI elements appear in two places, capture them in a single section, and reference the section in both places where the UI elements are required.



Don't duplicate logic

Section

First name	Last name
<input type="text"/>	<input type="text"/>



Section

First name	Last name
<input type="text"/>	<input type="text"/>
Phone	City
<input type="text"/>	<input type="text"/>

Submit

Section

First name	Last name
<input type="text"/>	<input type="text"/>
Email	Manager
<input type="text"/>	<input type="text"/>

Submit

**@baseclass** Don't duplicate logic

**Work-** Use Inheritance

**Work-Cover-**

```
graph TD; Class[Class] --> ClassRule1[Class RULE]; Class --> ClassRule2[Class RULE]; Class --> ClassRule3[Class RULE]
```

Use inheritance.

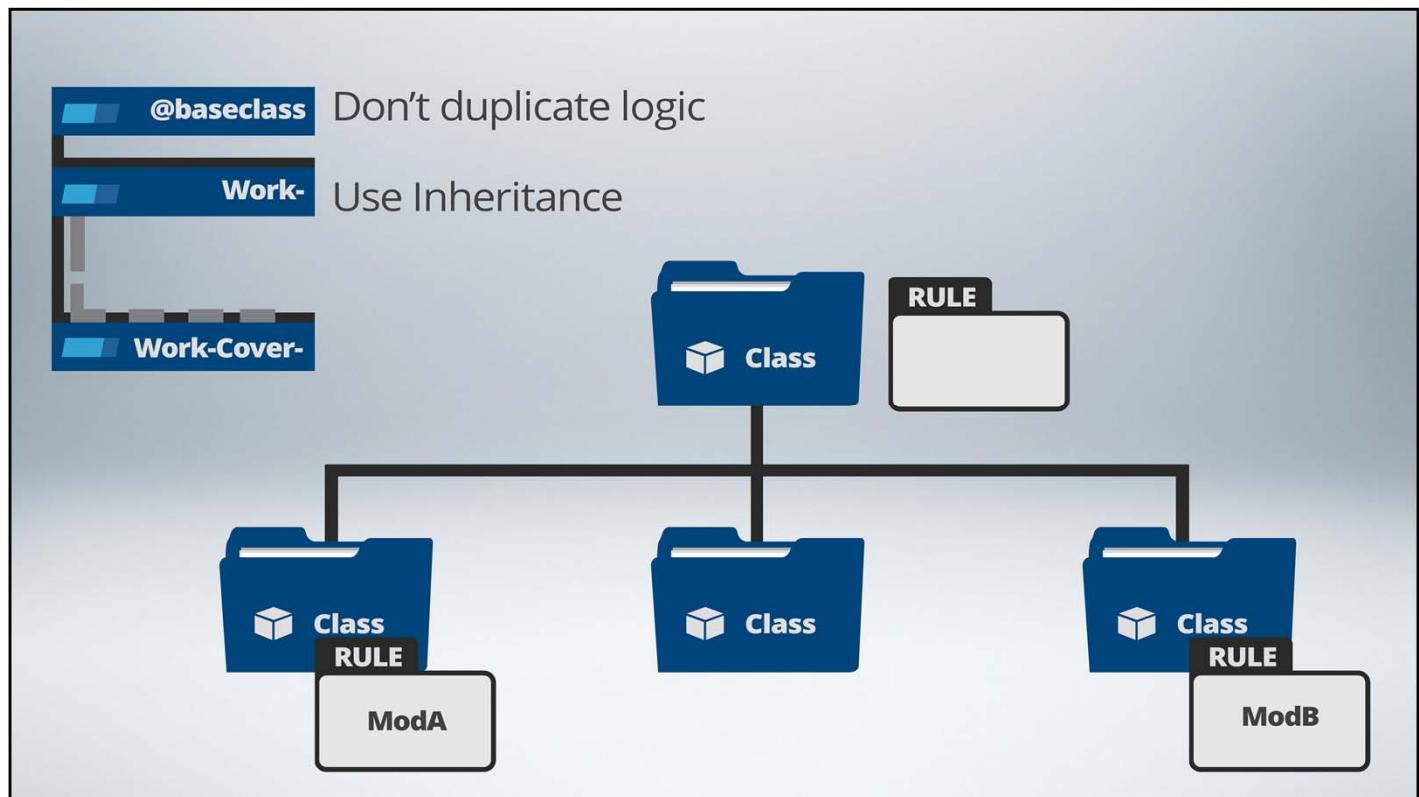
If the same rule applies to two different classes, move it to a shared parent class so both child classes can inherit from it.

**@baseclass** Don't duplicate logic

**Work-** Use Inheritance

**Work-Cover-**

```
graph TD; Class1[Class] --- Class2[Class]; Class1 --- Class3[Class]; Class1 --- RULE[RULE]
```



If the same record applies to two classes, with tiny differences, move the shared logic to a record in a shared parent. Overwrite the extension points as needed in the subclass and include the subtle differences there.

The diagram illustrates the concept of reusing APIs. On the left, a blue circular icon with three arrows forming a loop represents 'Don't reinvent the wheel'. To its right, three text items provide advice: 'Don't duplicate logic', 'Use Inheritance', and 'Don't reinvent the wheel'. Below this is a screenshot of the 'Designer Studio' interface, showing a window titled 'API' with several tabs. To the right of the screenshot, a black plug icon represents 'Integration API', followed by a list of integration methods: Rulesets, Ruleset Versions, Branching, Updating Records, HTTP Requests, XML and JSON, and SOAP.

API Reuse	Integration API
Don't duplicate logic	HTTP Requests
Use Inheritance	XML and JSON
Don't reinvent the wheel	SOAP

Don't reinvent the wheel.

Leverage the work of others whenever you can. If an API already exists for your task, and has been tested and proven reliable, use it. Time spent researching what's already available to you when you are in unfamiliar territory is almost never time wasted. If you don't know, ask around and find someone who knows the territory to advise you.

The Designer Studio has APIs for creating RuleSets, creating RuleSet Versions, creating branches, updating records, and thousands upon thousands of other actions.

Integration has APIs, records, and utilities for creating HTTP requests, parsing XML and JSON, making SOAP requests, authenticating requests, parsing and validating URLs, and thousands of other tasks.



We have discussed what guardrails are and why they are important for you to monitor when building applications. By following guardrails you are implementing best practices in your application.

There will always be exceptions to guardrails but if you can minimize these exceptions and follow these best practices you will build better applications that are Built For Change.

## Exercise: Resolve Guardrail Warnings



# Guidelines for Maintaining Requirements and Specifications

This lesson discusses the importance of keeping specifications and requirements up to date during a development effort.

At the end of this lesson, you should be able to:

- State the system architect's role in the development process
- State the difference between a specification and a requirement
- List at least two reasons why it is important to maintain up-to-date requirements and specifications



To build an application, we have to know what it is we actually want to build.

A set of stakeholders, including subject matter experts, process participants, process analysts and process designers,

build the specifications and requirements for the application in Designer Studio. Creating the specifications and requirements is an iterative process that should include all stakeholders.



The development team, consisting of system architects, uses Designer Studio to access the specifications, what they need to build, and the requirements, conditions for the specifications.

This makes Designer Studio the system of record for the specifications and requirements of an application.

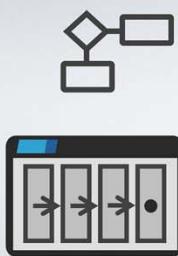


Applications are defined by a set of...



### specifications

what the application does

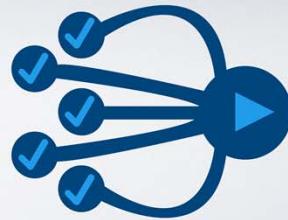


and



### requirements

conditions that must happen



Applications are defined by a set of specifications and requirements.

In Pega, a specification represents a small processing unit performed by one or more actors for a given case type within an application.

A single specification may correspond to an entire process, a single flow action, a screen flow, or the New harness of a flow. A specification uses one or more requirements to define success criteria that the specification is complete.

In Pega, requirements are an inventory of events, conditions, or functions that need to be satisfied and tracked by an application.



Applications are defined by a set of...



### specifications

what the application does

Collect employee information

Employees can select benefits

Hiring manager can assign assets to an employee

and



### requirements

conditions that must happen

System needs to have 2-3 seconds screen to screen interaction

Employees have the option to waive benefits

Employees cannot be assigned more than one of the same asset type

Requirements are described in various forms. Some are non-functional while others articulate specific business rules that must be satisfied.

Some examples of specifications include:

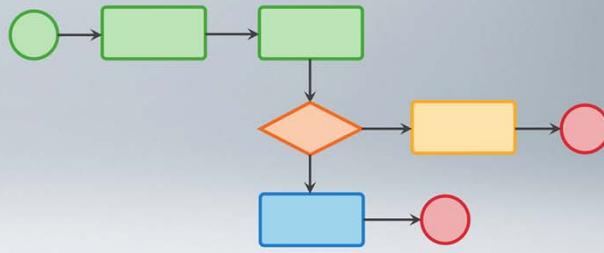
- Collect employee information
- Employees can select benefits
- Hiring manager can assign assets to an employee

Some examples of corresponding requirements include:

- System needs to have 2-3 seconds screen to screen interaction
- Employees have the option to waive benefits
- Employees cannot be assigned more than one of the same asset type.



Keep Specifications & Requirements up to date in Designer Studio

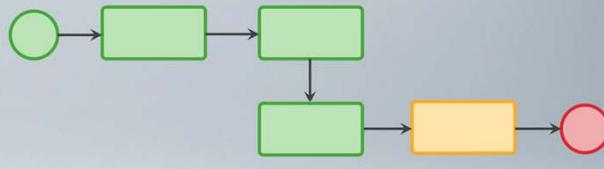


- Manage expectations
- Limit any surprises

As part of the implementation of an application it is very important to continually keep the specifications and requirements up to date in Designer Studio. First by keeping everything up to date we help manage expectations of all the stakeholders involved in the project. This helps to limit any surprises from any of the stakeholders over what will be completed as part of the application.



Keep Specifications & Requirements up to date in Designer Studio



- Manage expectations
- Limit any surprises
- Eliminate unnecessary work

Keeping everything up to date also helps to eliminate unnecessary work as the goals of a project may change and parts of a project that are removed can be communicated and not built.

The slide features a blue cube icon with 'PEGA' and '7' on it, accompanied by a circular arrow icon. Below is a screenshot of the 'Designer Studio' application interface, showing a search icon and a network graph icon. To the right, a process flow diagram is shown, consisting of green rectangles, orange diamonds, and red circles. A red box highlights a section of the flow.

UP TO DATE SPECIFICATIONS AND REQUIREMENTS REDUCES SCOPE/FEATURE CREEP!

```
graph TD; Start(( )) --> A1[ ]; A1 --> A2[ ]; A2 --> A3[ ]; A3 --> A4[ ]; A4 --> Decision1{ }; Decision1 --> A5[ ]; Decision1 --> A6[ ]; A5 --> Decision2{ }; Decision2 --> A7[ ]; Decision2 --> A8[ ]; A7 --> A9[ ]; A9 --> A10[ ]; A10 --> End1(( )); A8 --> A11{ }; A11 --> A12[ ]; A11 --> A13[ ]; A12 --> A14[ ]; A13 --> A15[ ]; A14 --> A16{ }; A15 --> A16; A16 --> A17[ ]; A17 --> A18[ ]; A17 --> A19[ ]; A18 --> End2(( )); A19 --> End3(( ));
```

Lastly, keeping specifications and requirements up to date can help reduce scope and feature creep by providing visibility into what is being built.