Table of Contents

University Accounts Payable System Executive Summary

Over the past two months, our group has been working towards implementing a working Accounts Payable System for a major university. The purpose of this project is to design a system to replace the University's current Retail Food Service Vendor Account Payable System (VAPS), which suffers from limited computerized support and inefficient processes that are completed manually. Specifically, the current VAPS has caused the university to lose money through late payment fees and loss of vendor discounts. As a result of the lack of automation, report gathering is extremely time consuming for the employees and resulted in high turnover rate of employees.

The objective of our project is to create a system that satisfies the university's capability requests. The capabilities are: allowing clerks to enter invoices, weekly generation of request issuances, elimination of duplicate data entries, tracking of request issuances to the clerk's office, monthly report generation, support for unexpected management report requests, and finally, savings that outweigh the system costs.

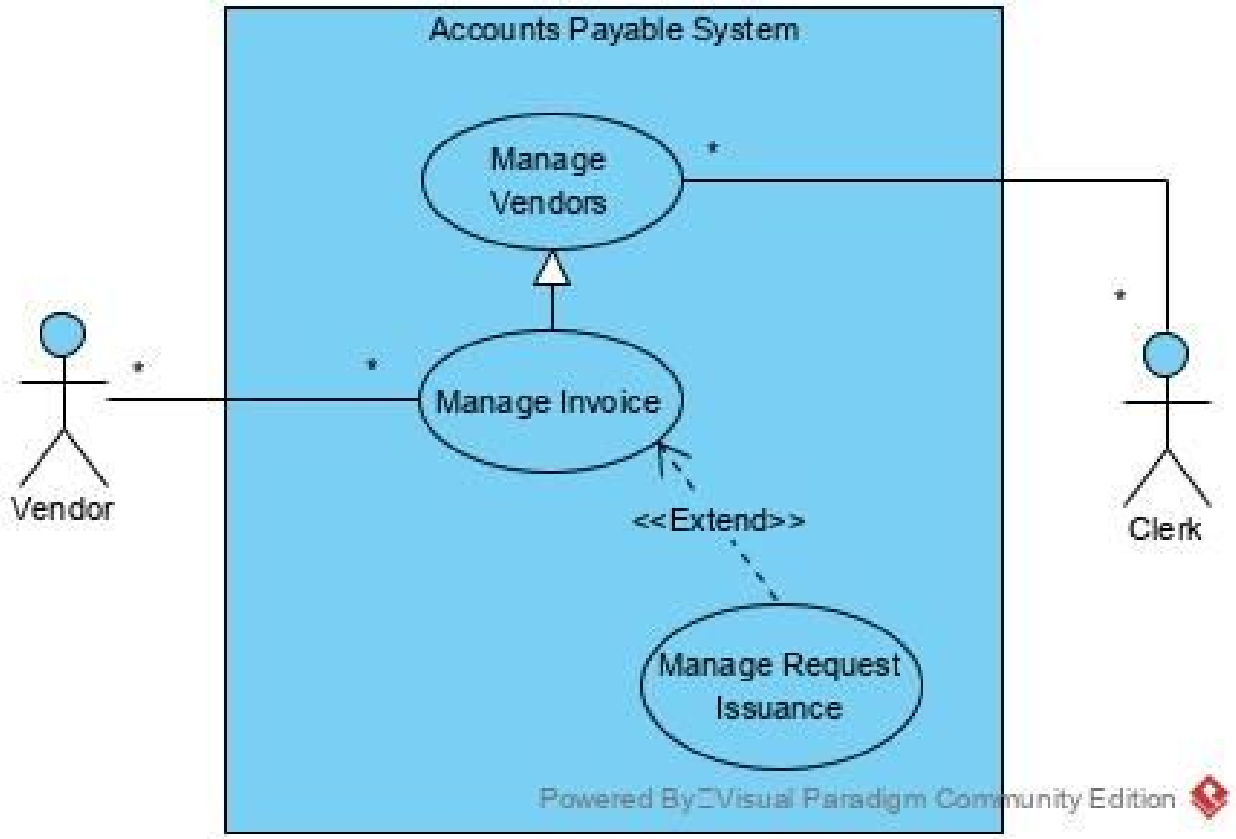
The new system we have been working on aimed to fix the drawbacks of the university's current system by automating many of the manual processes as well as provide a single, easy to use, platform to perform all necessary operations within the system. As a result, we were able to meet the capabilities that were requested by the major university.

Requirements Definition

- ★ Non-Functional Requirements
  - Operational Requirements:
    - The system should run in Windows environment.
    - The system should be capable of showing daily invoices.
    - The system should be able to store information on vendors, franchises, and products.
    - The system should be able to generate a payment report.
    - The system should be able to generate unique accounting code for each product for tracking purposes.
    - The system should be able to create request issuance form.
    - The system should create only one data entry on vendors, franchisers, and product types.
  - Performance Requirements:
    - The system should be capable of handling around 25 invoices per month.
    - The system should generate request issuances weekly.
  - Security Requirements:
    - The system should be capable of sending issuance requests and bank reports to the university check issuance office (to prevent fraud).
  - Cultural and Political Requirements:
    - None
- ★ Functional Requirements
  - Manage Vendors
    - Create Vendor
    - Edit Vendor
    - Delete Vendor
  - Manage Invoice
    - Select Vendor
    - Create Invoice
    - Complete Invoice
    - Cancel Invoice
  - Manage Request Issuance
    - Create Request Issuance

- Get Invoice Information
- Complete Request Issuance
- Cancel Request Issuance

Use Case Diagram



User Stories

- ★ Non-Functional User Stories:
    - As a clerk, I need to be able to manage vendors, enter daily invoices, create an issuance request and generate a payment report.
    - As an Accounts payable accounts, I need to be able to manage payments between our retail vendors effectively with the use of a new computerized system.
- ★ Functional User Stories
    - As a ***Clerk***, I want to be able to stow necessary documents and information making the university's payment process more efficient.
    - As a ***Clerk***, I want to be able to manage vendors so that the University has the right info and status of their relationship.
    - As a ***Vendor***, I want to be able to manage invoices so all the information is correct and approved to receive payment for services.
    - As a ***Clerk***, I want to be able to manage request issuance so that the rest of the request issuance info is added to the request issuance and submitted.
    - As a ***Clerk***, I want to be able to manage reports so that everything is organized.
    - As an ***Accounts Payable Supervisor (Clerk)***, I want to be able to maintain positive relationships between my organization, other university units, and our retail vendors.
    - As the ***University Financial Comptroller (Clerk)***, reduce costs and identify savings opportunities optimized through a new system.
    - As the ***Retail Vendor Accountant (Clerk)***, I want to be able to quickly and efficiently handle account transactions, which primarily supports the on-campus food franchises.

Use Case Descriptions

Use Case Name: Manage Vendors	ID: 1	Importance Level: High
Primary Actor: Clerk	Use Case Type: Detail, Essential	
Stakeholders and Interests: Accounts Payable Staff – wants to manage vendors in the system by creating new vendors, editing existing vendors, or removing vendors Vendors – want to ensure University has correct info and status of their relationship		
Brief Description:This use case describes how vendors will be managed by the staff within the system		
Trigger: Vendor contacts University notifying a change in info or status Type: External		
Relationships: Association: clerk Include: Extend: Generalization: Manage Invoice		
Normal Flow of Events: 1. University contacts Accounts Payable office regarding a vendor. If the University wants to add a new vendor, the S-1: new vendor subflow is performed If the University wants to update information on an existing vendor, the S-2: edit vendor subflow is performed If the University wants to remove a vendor, the S-1: delete vendor subflow is performed 2. The Accounts Payable clerk provides concluding status of action performed.		
SubFlows: **S-1:** New Vendor 1. The clerk asks the University for the vendor information. 2. The clerk fills in the information in order to add a new vendor. **S-2:** Edit Vendor 1. The clerk asks the University for the vendor information. 2. The clerk updates info on the existing vendor. **S-3:** Delete Vendor 1. The clerk asks the University for the vendor information 2. The clerk removes the vendor from the list of vendors.		
Alternate/Exceptional Flows:		

Use Case Name: Manage Invoice		ID: 2	Importance Level: High
Primary Actor: Clerk, Vendor	Use Case Type: Detail, Essential		
Stakeholders and Interests: Accounts Payable Staff – wants to manage Invoices received from vendors Vendors – want to submits Invoices to receive payment for services			
Brief Description: This use case describes how invoices will be managed by the staff within the system.			
Trigger: Invoice is created Type: External			
Relationships: Association: Include: Extend: Manage Request Issuance Generalization:			
Normal Flow of Events: 1. University sends Accounts Payable office an Invoice. 2. If the Invoice includes a new vendor, or vendor info has changed Execute the Manage Vendors use case. 3. The clerk selects a vendor to fill out vendor info in the Invoice If the clerk wants to fill out the rest of the invoice info, the S-1: complete Invoice subflow is performed If the clerk wants to cancel this Invoice, the S-2: cancel Invoice subflow is performed 4. The Accounts Payable clerk provides concluding status of action performed.			
SubFlows: **S-1:** Complete Invoice 1. The clerk enters products, quantities, prices, etc. 2. The clerk creates a new filled out Invoice. **S-2:** Cancel Invoice 1. The clerk wants to cancel the Invoice.			
Alternate/Exceptional Flows:			

Use Case Name: Manage Request Issuance	ID: 3	Importance Level: High
Primary Actor: Clerk	Use Case Type: Detail, Essential	

Stakeholders and Interests:
Accounts Payable Staff – wants to manage Request Issuances in the system by creating new Request Issuances
Vendors – want to ensure University has a system in place for properly paying for vendor services

Brief Description:This use case describes how a Request Issuance will be managed by the staff within the system

Trigger: Accounts Payable Office compiles necessary invoices to write a Request Issuance to send to the check issuance office.
Type: Temporal

Relationships:
Association: Clerk
Include:
Extend:
Generalization:

Normal Flow of Events:

1. Accounts Payable office clerk compiles necessary invoices for the Request Issuance
2. The Clerk creates a new Request Issuance form
3. The Clerk imports Invoice info related to this particular Request Issuance
   If the clerk wants to fill out the rest of the Request Issuance info,
   the S-1: complete Request Issuance subflow is performed
   If the clerk wants to cancel this Request Issuance,
   the S-2: cancel Request Issuance subflow is performed
4. The Accounts Payable clerk provides concluding status of action performed.

SubFlows:

S-1: Complete Request Issuance

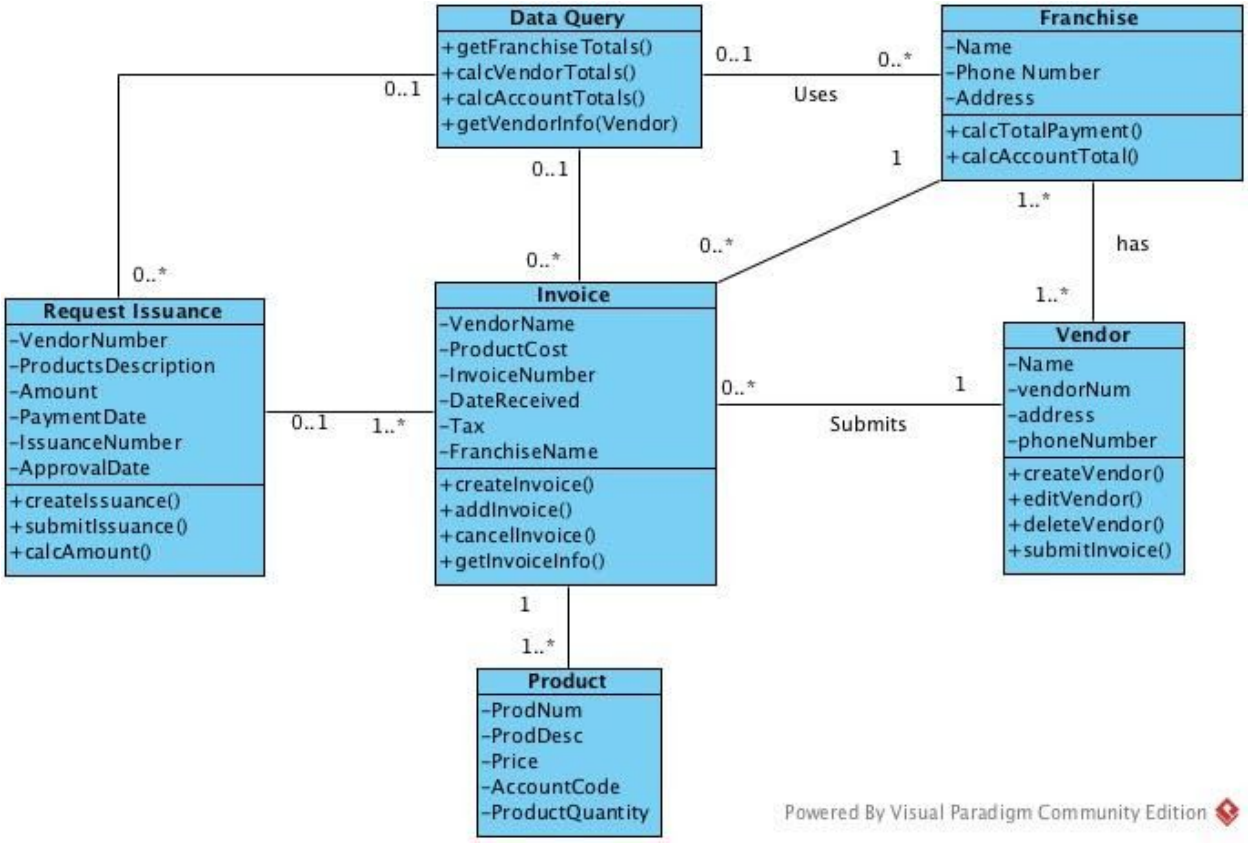
1. The clerk enters dates, description, check number, etc.
2. The clerk creates a new filled out Request Issuance form.

S-2: Cancel Request Issuance

1. The clerk wants to cancel to Request Issuance.

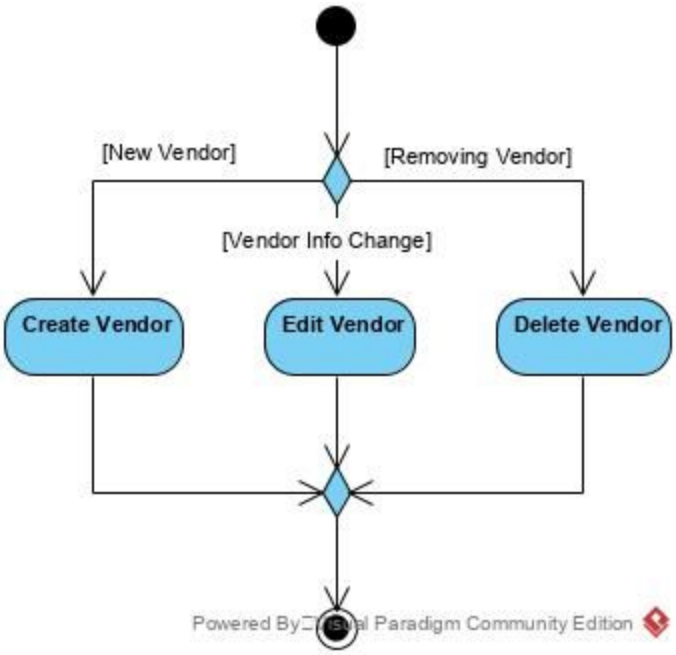
Alternate/Exceptional Flows:

Class Diagram

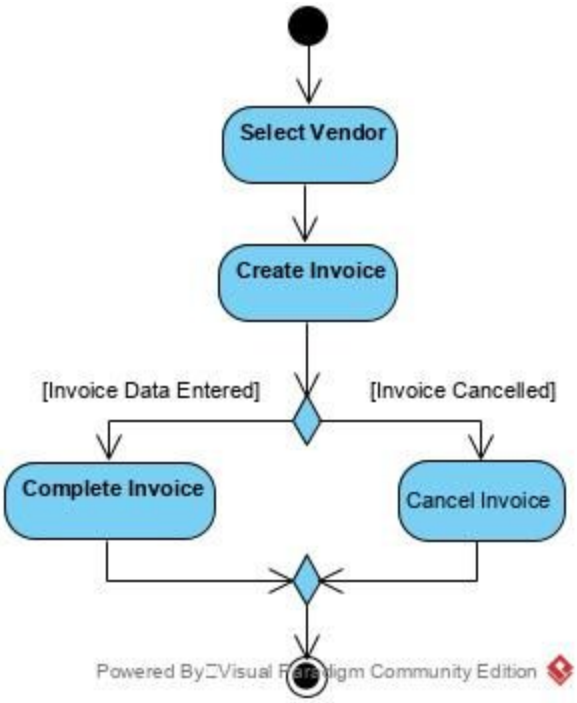


Activity Diagrams

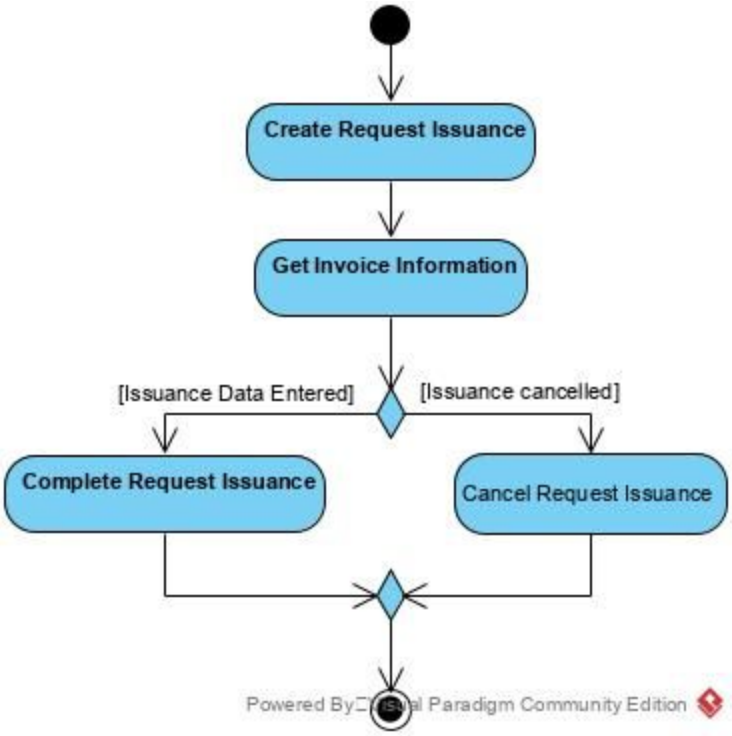
★ Manage Vendors Activity Diagram



★ Manage Invoice Activity Diagram

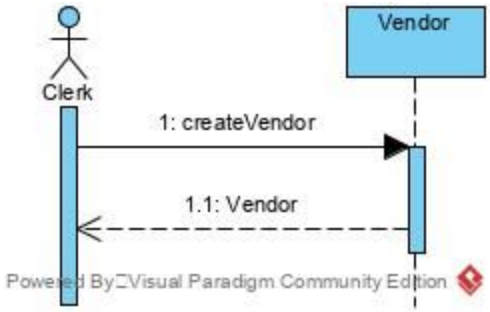


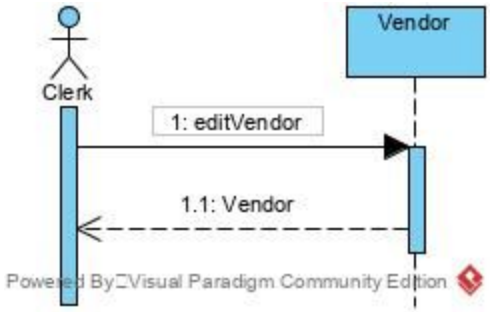
★ Request Issuance Activity Diagram

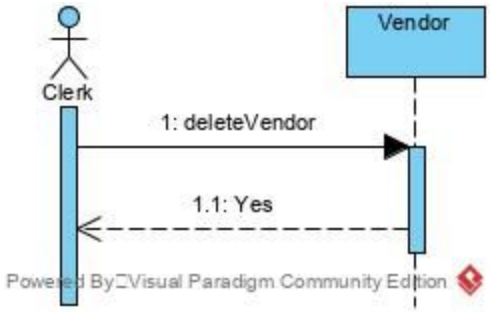


Sequence Diagrams

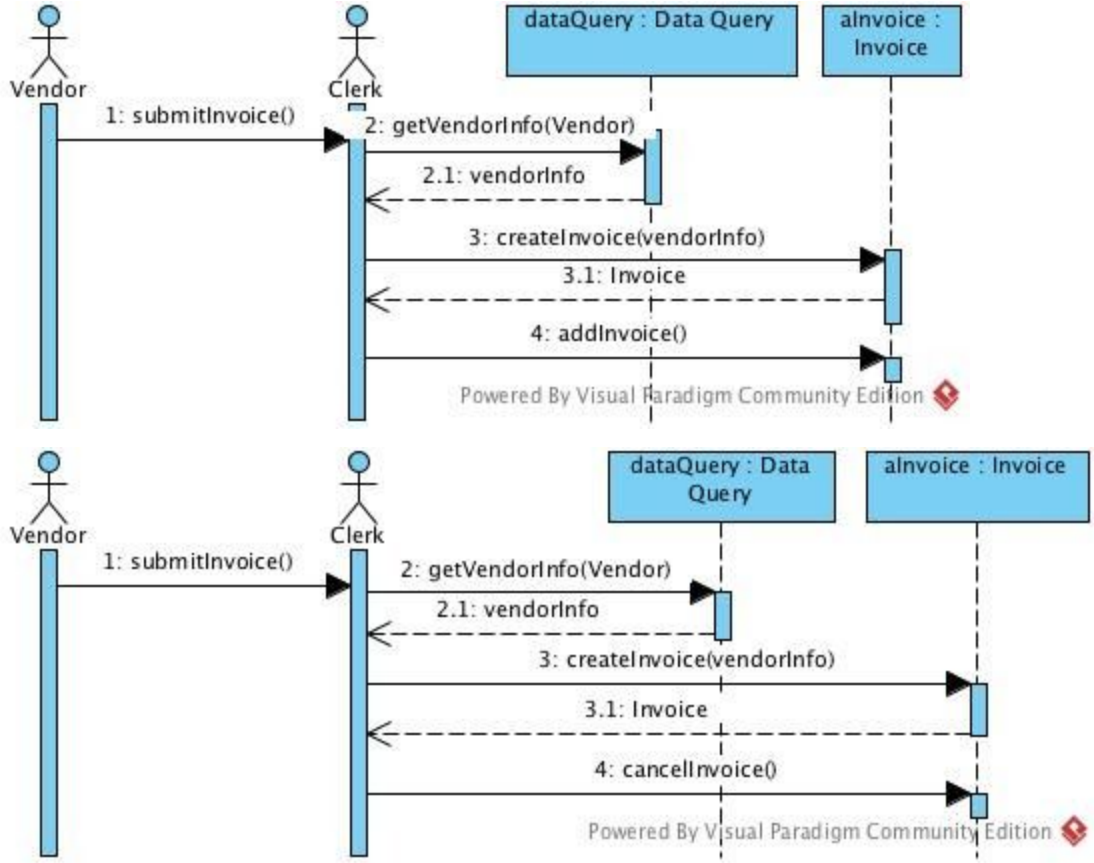
★ Manage Vendor Sequence Diagrams



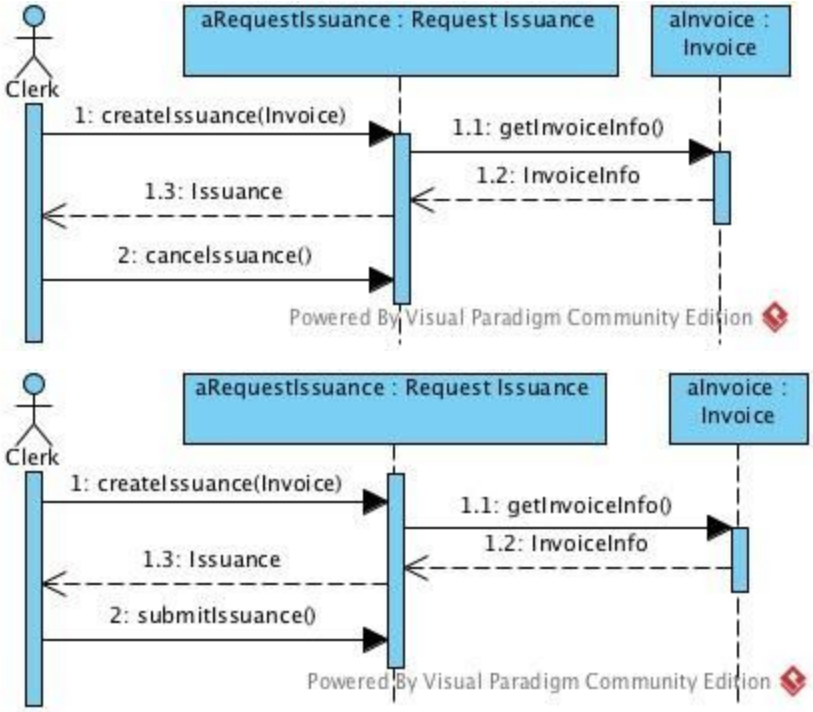




★ Manage Invoice Sequence Diagrams



★ Request Issuance Sequence Diagrams

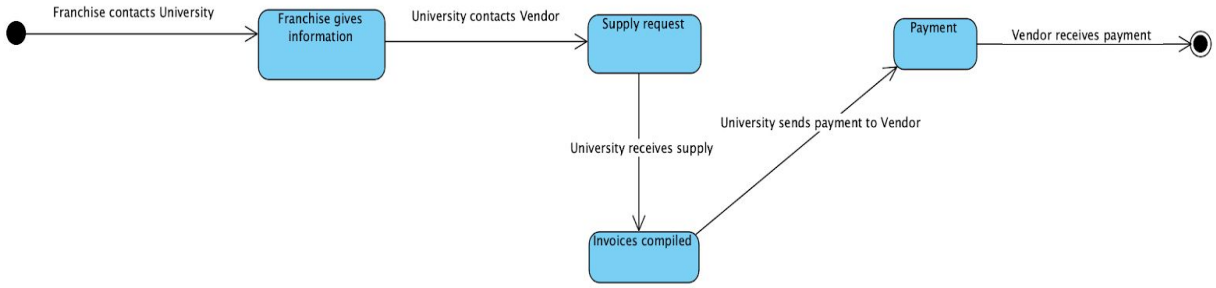


CRUDE Matrix

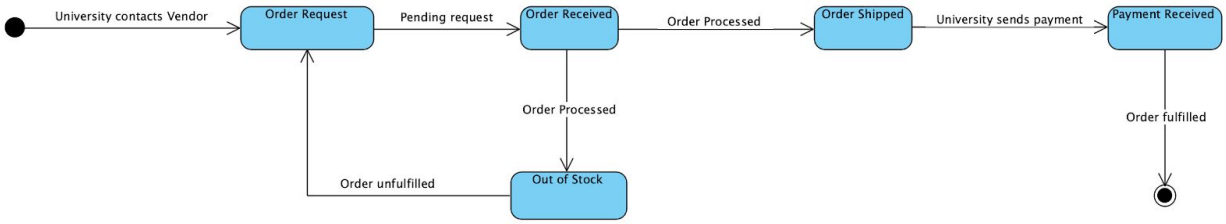
	Clerk Actor	Vendor Actor	Invoice	Issuance	Franchise	Product
Clerk Actor			C,R,U,D,E	C,R,U,D,E		C,R,U,D
Vendor Actor			C,R			R
Invoice	C,R,U,D,E	C,R				
Issuance	C,R,U,D,E					
Franchise						
Product	C,R,U,D	R				

Behavioral State Machines

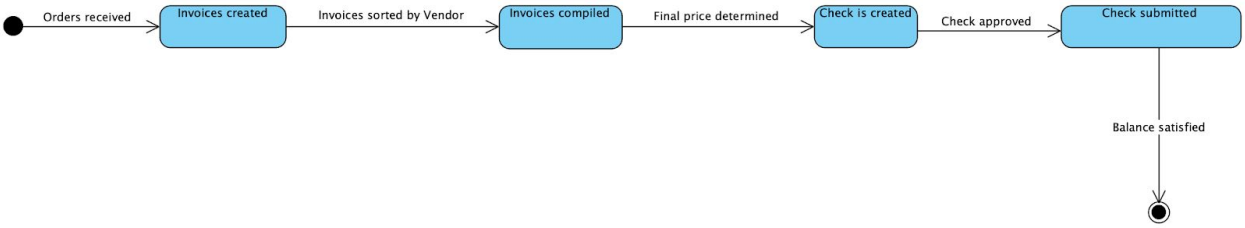
★ Franchise Behavioral State Machine



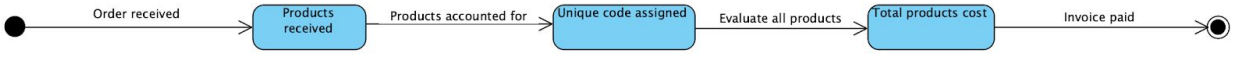
★ Vendor Behavioral State Machine



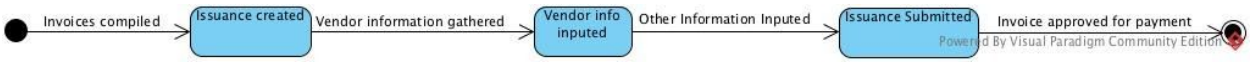
★ Invoice Behavioral State Machine



★ Products Behavioral State Machine



★ Request Issuance Behavioral State Machine



Invariants

★ Vendor Class Invariants

- Name = vendor.getName()
- VendorNum = vendor.getVendorNum()
- Address = vendor.getAddress()
- PhoneNumber = vendor.getPhoneNumber
- Name (1..1) : String
- VendorNum (1..1) : Integer
- Address (1..*) : String
- PhoneNumber (1..1) : String

★ Franchise Class Invariants

- Name = franchise.getName()
- phoneNumber = franchise.getPhoneNumber()
- Address = franchise.getAddress()
- Name (1..1) : String
- phoneNumber (1..1) : String
- Address (1..1) : String

★ Invoice Class Invariants

- VendorName = Invoice.getVendorName()
- ProductCost = Invoice.getProductCost
- InvoiceNumber = Invoice.getInvoiceNumber()
- DateReceived = Invoice.getDateReceived()
- Tax = Invoice.getTax()
- FranchiseName = Invoice.getFranchiseName()
- VendorName (1..1) : String
- ProductCost (1..*) : Double
- InvoiceNumber (1..1) : Integer
- DateReceived (1..1) : String
- Tax (1..1) : Double
- FranchiseName (1..*) : String

★ Product Class Invariants

- ProdNum = Product.getProdNum()
- ProdDesc = Product.getProdDesc()
- Price = Product.getPrice()
- AccountCode = Product.getAccountCode()
- ProductQuantity = Product.getProductQuantity()
- ProdNum (1..1) : Integer
- ProdDesc (1..1) : String
- Price (1..1) : Double
- AccountCode (1..1) : Integer
- ProductQuantity (1..*) : integer

★ Request Issuance Class

- VendorNumber = RequestIssuance.getVendorNumber()
- ProductsDescription = RequestIssuance.getProductsDescription()
- Amount = RequestIssuance.getAmount()
- PaymentDate = RequestIssuance.getPaymentDate()
- IssuanceNumber = RequestIssuance.getIssuanceNumber()
- ApprovalDate = RequestIssuance.getApprovalDate()
- VendorName (1..1) : String
- ProductsDescription (1..1) : String
- Amount (1..*) : Double
- PaymentDate (1..1) : String
- IssuanceNumber (1..1) : Integer
- Approval Date (1..1) : String

Contracts

Method Name: submitIssuance()	Class Name: Request Issuance	ID: 1
Clients (Consumers): Clerk		
Associated Use Cases: Manage Request Issuances		
Description of Responsibilities: Implement the necessary behavior to send an issuance from accounts payable to the vendor via the checks office to solidify payment.		
Arguments Received: None.		
Type of Value Returned: void		
Pre-Conditions: Issuance exists.		
Post-Conditions: The Request Issuance is sent to the clerk's office		

Method Name: calcTotalPayment()	Class Name: Franchise	ID: 2
Clients (Consumers): Vendor		
Associated Use Cases: Manage Invoice		
Description of Responsibilities: Implement the necessary behavior to compile all invoices and submit the grand total to the checks office		
Arguments Received:		
Type of Value Returned: Double		
Pre-Conditions: None.		
Post-Conditions: None.		

Method Name: calcAccountTotal	Class Name: Franchise	ID: 3
Clients (Consumers): Vendor		
Associated Use Cases: Manage Invoice		
Description of Responsibilities: Implement the necessary behavior to calculate all account totals.		
Arguments Received: aProduct : Product		
Type of Value Returned: Double		
Pre-Conditions: Product account exists.		
Post-Conditions: None		

Method Name: getFranchiseTotals()	Class Name: Data Query	ID: 4
Clients (Consumers): Clerk		
Associated Use Cases: Manage Invoice		
Description of Responsibilities: Implement the necessary behavior to compute the total for each Franchise.		
Arguments Received: aFranchise: Franchise		
Type of Value Returned: String		
Pre-Conditions: Franchise exists.		
Post-Conditions: None.		

Method Name: calcVendorTotals()	Class Name: Data Query	ID: 5
Clients (Consumers): Vendor		
Associated Use Cases: Manage Invoice		
Description of Responsibilities: Implement the necessary behavior to compute the total for each Vendor.		
Arguments Received: aVendor : Vendor		
Type of Value Returned: String		
Pre-Conditions: Vendor exists.		
Post-Conditions: None		

Method Name: CalcAccountTotals()	Class Name: Data Query	ID: 6
Clients (Consumers): Clerk		
Associated Use Cases: Manage Invoice		
Description of Responsibilities: Implement the necessary behavior to compute the total for all University Accounts.		
Arguments Received: aInvoice: Invoice		
Type of Value Returned: String		
Pre-Conditions: None		
Post-Conditions: None		

Method Name: getVendorInfo()	Class Name: Data Query	ID: 7
Clients (Consumers): Vendor		
Associated Use Cases: Manage Vendors		
Description of Responsibilities: Vendor is contacted and information is obtained.		
Arguments Received: aVendor: Vendor		
Type of Value Returned: String		
Pre-Conditions: Vendor Exists		
Post-Conditions: None		

Method Name: addInvoice()	Class Name: Invoice	ID: 8
Clients (Consumers): Clerk		
Associated Use Cases: Manage Invoice		
Description of Responsibilities: A new invoice is created assigned to a specific account.		
Arguments Received: None.		
Type of Value Returned: void		
Pre-Conditions: None.		
Post-Conditions: Invoice added to the database.		

Method Name: cancelInvoice()	Class Name: Invoice	ID: 9
Clients (Consumers): Clerk		
Associated Use Cases: Manage Invoice		
Description of Responsibilities: Implement the necessary behavior to allow an invoice to be cancelled.		
Arguments Received: None.		
Type of Value Returned: void		
Pre-Conditions: Invoice exists.		
Post-Conditions: Invoice removed from the system.		

Method Name: DeleteVendor()	Class Name: Vendor	ID: 10
Clients (Consumers): Vendor		
Associated Use Cases: Manage Vendors		
Description of Responsibilities: Implement the necessary behavior to allow a Vendor to be deleted from the system.		
Arguments Received: None.		
Type of Value Returned: void		
Pre-Conditions: Vendor Exists		
Post-Conditions: Vendor removed from the system.		

Method Specifications

Method Name: submitIssuance	Class Name: Request Issuance	ID: 1
Contract ID: 1	Programmer: Brandon Ho	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: Accounts Payable is has compiled needed invoices for a Request Issuance		

Arguments Received: Data Type:	Notes:

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:

Argument Returned: Data Type:	Notes:
void	
Algorithm Specification:	
Misc.Notes:	

Method Name: calcTotalPayment	Class Name: Franchise	ID: 2
Contract ID: 2	Programmer: Brandon Ho	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: A request for totals paid out for a franchise is needed		

Arguments Received: Data Type:	Notes:

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:
anInvoice.getProductCost anInvoice.getVendorName	Double	Gets the total cost of products from an invoice
	String	

Argument Returned: Data Type:	Notes:
Double	The total cost of all products ordered for a franchise
Algorithm Specification: FOR all Invoices DO total = total + anInvoice.getProductCost	
Misc.Notes:	

Method Name: calcAccountTotal	Class Name: Franchise	ID: 3
Contract ID: 3	Programmer: Brandon Ho	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: A request for franchise totals by university account is needed		

Arguments Received: Data Type:	Notes:
Product	

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:
anInvoice.getProduct.getAccountCode	Integer	Gets the Product Account code

Argument Returned: Data Type:	Notes:
Double	The total cost of all products
Algorithm Specification: FOR all Invoices.getProduct DO IF product.AccountCode = accCode THEN total = total + anInvoice.getProductCost	
Misc.Notes:	

Method Name: getFranchiseTotals	Class Name: Data Query	ID: 4
Contract ID: 4	Programmer: Gunnar Biebighauser	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: A request for a total paid out for each franchise is needed		

Arguments Received: Data Type:	Notes:
Franchise	

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:
aFranchise.calcTotalPayment()	Double	Gets total for a franchise

Argument Returned: Data Type:	Notes:
String	The total paid out by each franchise
Algorithm Specification: FOR all Franchises DO aFranchise.calcTotalPayment()	
Misc.Notes:	

Method Name: calcVendorTotals	Class Name: Data Query	ID: 5
Contract ID: 5	Programmer: Gunnar Biebighauser	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: A request for vendor totals is needed		

Arguments Received: Data Type:	Notes:
Vendor	

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:
anInvoice.getVendorName	String	
anInvoice.getProductCost	Double	

Argument Returned: Data Type:	Notes:
String	The total cost of all products
Algorithm Specification: FOR all invoices DO IF anInvoice.getVendorName = aVendor.getName THEN get the total cost of that invoice	
Misc.Notes:	

Method Name: calcAccountTotals	Class Name: Data Query	ID: 6
Contract ID: 6	Programmer: Gunnar Biebighauser	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: A request for vendor totals is needed		

Arguments Received: Data Type:	Notes:
Invoice	

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:
anInvoice.getProduct.getAccountCode	Integer	Gets the account code
aProduct.getPrice	Double	Gets the price

Argument Returned: Data Type:	Notes:
String	The total for each account

Algorithm Specification:
FOR all invoices DO
FOR all products DO
Add the price of the product to each unique account code	

Misc.Notes:	

Method Name: getVendorInfo	Class Name: Data Query	ID: 7
Contract ID: 7	Programmer: Gunnar Biebighauser	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: A request for vendor info is needed		

Arguments Received: Data Type:	Notes:
Vendor	

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:

Argument Returned: Data Type:	Notes:
String	
Algorithm Specification:	
Misc.Notes:	

Method Name: addInvoice	Class Name: Request Issuance	ID: 8
Contract ID: 8	Programmer: Adam Taylor	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: An Invoice is received		

Arguments Received: Data Type:	Notes:

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:

Argument Returned: Data Type:	Notes:
void	
Algorithm Specification:	
Misc.Notes:	

Method Name: cancelInvoice	Class Name: Request Issuance	ID: 9
Contract ID: 9	Programmer: Adam Taylor	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: An Invoice is no longer needed		

Arguments Received: Data Type:	Notes:

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:

Argument Returned: Data Type:	Notes:
void	The invoice created is deleted
Algorithm Specification:	
Misc.Notes:	

Method Name: deleteVendor	Class Name: Request Issuance	ID: 10
Contract ID: 10	Programmer: Adam Taylor	Date Due: 12/10/19
Programming Language: ● Java		
Triggers/Events: A vendor is no longer needed in the system		

Arguments Received: Data Type:	Notes:

Messages Sent & Arguments Passed: ClassName.MethodName:	Argument Data Type:	Notes:

Argument Returned: Data Type:	Notes:
void	
Algorithm Specification:	
Misc.Notes:	

Meeting Reporting Form (1)

Date: 10-24-19 **Group Name: BusyBusinessAnalysts**

Start Time: 12:30PM **End Time: 1:45PM**

Attendees: (List the persons at the meeting – not everyone needs to attend every meeting.)

- **Brandon Ho**
- **Gunnar Biebighauser**
- **Andy Kim**
- **Adam Taylor**
- **Ken Ningcharoen**

Topic: (Use descriptive terms, try to be short and informative)

- **Read project specifications**

Pre-meeting Materials for Review: (Provide title and location)

- **Device to view specifications**

Agenda: (What are you planning to do in this meeting – Keep it short and focused on the topic)

1. **Start Project overview**

Description of Activities:

- **I**ndividually looked over project specifications, after we pointed out possible functions of the system for the use case diagram. We also got familiar with possible objects that may be created.

Action Items: (What needs to be done next – determine who will do it and when they expect it to be done.)

What	Who	By When
Start on User Cases	Adam Taylor	10-27-19
Start on Use Case Diagram	Ken Ningcharoen	10-27-19
Start on User Stories	Andy Kim	10-27-19
Start on Project Problem Outline	Brandon Ho	10-27-19
Start on Activity Diagram	Gunnar Biebighauser	10-27-19

Unresolved Issues: Understanding the problem

Meeting Reporting Form (2)

Date: 10-31-19 **Group Name: BusyBusinessAnalysts**
Start Time: 12:30PM **End Time: 1:45PM**
Attendees: (List the persons at the meeting – not everyone needs to attend every meeting.)

- **Gunnar Biebighauser**
- **Andy Kim**
- **Adam Taylor**
- **Ken Ningcharoen**
- **Brandon Ho**

Topic: (Use descriptive terms, try to be short and informative)

- **Simplifying the project**

Pre-meeting Materials for Review: (Provide title and location)

- **Device to view specifications**
- **Breakdown the project**

Agenda: (What are you planning to do in this meeting – Keep it short and focused on the topic)

1. **Fully understand the project**
2. **Edit Use Cases**
3. **Edit Use Case Diagram**
4. **Edit User Stories**
5. **Edit Project problem outline**
6. **Edit Activity Diagram**

Description of Activities:

- We understood the project better by breaking down and taking detailed notes. We then edited the use cases, use case diagram, user stories, and activity diagram that we individually started.

Action Items: (What needs to be done next – determine who will do it and when they expect it to be done.)

What	Who	By When
Finish editing what we did in class	Everyone	11-2-19
Think ideas about Class diagram	Everyone	11-2-19

Unresolved Issues: (List problems and issues that hinder your progress.)

1. Not 100 percent sure on the Use cases.
2. Still having a hard time understanding the problem fully.

Meeting Reporting Form (3)

Date: 11-3-19 **Group Name: BusyBusinessAnalysts**
Start Time: 4:00PM **End Time: 5:00PM**
Attendees: (List the persons at the meeting – not everyone needs to attend every meeting.)

- **Brandon Ho**
- **Gunnar Biebighauser**
- **Andy Kim**
- **Adam Taylor**
- **Ken Ningcharoen**

Topic: (Use descriptive terms, try to be short and informative)

- **Revising and consolidating diagrams**
- **Class Diagram**

Pre-meeting Materials for Review: (Provide title and location)

- **Device to view specifications**
- **Read over the project again**

Agenda: (What are you planning to do in this meeting – Keep it short and focused on the topic)

1. **Develop Class Diagrams**
2. **Revise draft diagrams**

Description of Activities:

- We revised and went over the Use case and activity diagrams that we had made. We made small changes as we discussed as a group. We also combined the class diagrams that we each created.

Action Items: (What needs to be done next – determine who will do it and when they expect it to be done.)

What	Who	By When
Go over the class diagram	Everyone	11-6-19
Iterate through the Use case diagram	Everyone	11-6-19
Think about ideas about Sequence Diagram and CRUDE Matrix	Everyone	11-6-19

Unresolved Issues: (List problems and issues that hinder your progress.)

1. Not sure of the class diagram
2. Still sort of confused about what needs to be in the consolidating diagrams
3. Difference between the Vendor and the Franchise

Meeting Reporting Form (4)

Date: 11-7-19 **Group Name: BusyBusinessAnalysts**
Start Time: 4:30PM **End Time: 6:00PM**

Attendees: (List the persons at the meeting – not everyone needs to attend every meeting.)

- **Brandon Ho**
- **Gunnar Biebighauser**
- **Andy Kim**
- **Adam Taylor**
- **Ken Ningcharoen**

Topic: (Use descriptive terms, try to be short and informative)

- **Revise Use-Case diagram, Iterate through models**

Pre-meeting Materials for Review: (Provide title and location)

- **Device to view materials**

Agenda: (What are you planning to do in this meeting – Keep it short and focused on the topic)

1. **Revise Use-Case diagram**
2. **Iterate through models**
3. **Talk about the sequence diagram and CRUDE Matrix**

Description of Activities:

- In the meeting, we actually understood the problem fully and changed the use cases that we had. After that, we solidified problem domain and discussed verification of diagrams with the new use-case diagram. We also started iteration of fixing models and diagrams. Finally, we talked about ideas about the sequence diagram and CRUDE Matrix.

Action Items: (What needs to be done next – determine who will do it and when they expect it to be done.)

What	**Who**	**By When**
Fixing functional requirements, creating activity diagram for manage vendors and use-case description	Brandon	11/12/19
Finalizing class diagram and activity diagrams	Gunnar	11/12/19
Iterating through user stories	Andy	11/12/19
Updating and verifying manage reports activity diagram	Ken	11/12/19
Finalizing Class diagram and starting behavioral machine	Adam	11/12/19

Unresolved Issues: (List problems and issues that hinder your progress.)

- What type of reports to generate
- Verify if the Manage Reports Use Case is correctly including the view status of Issuance Report
- Who uses University Accounts?

Meeting Reporting Form (5)

Date: 11-13-19 **Group Name: BusyBusinessAnalysts**
Start Time: 12:30PM **End Time: 1:45PM**
Attendees: (List the persons at the meeting – not everyone needs to attend every meeting.)

- **Brandon Ho**
- **Gunnar Biebighauser**
- **Andy Kim**
- **Adam Taylor**
- **Ken Ningcharoen**

Topic: (Use descriptive terms, try to be short and informative)

- **Problem domain clarification**

Pre-meeting Materials for Review: (Provide title and location)

- **Device to view specifications**

Agenda: (What are you planning to do in this meeting – Keep it short and focused on the topic)

1. **Edit and fix things that needs to be fixed for class diagram**
2. **Edit Sequence Diagram and create CRUDE Matrix**
3. **Edit Behavior State Machine**

Description of Activities:

- We went over the sequence diagrams and CRUDE Matrix that we created and combined them. We clarified things that were confusing for everyone. We also went back to the class diagram to fix things that needs to be fixed. Finally, we edited behavior state machine that we created and combined them.

Action Items: (What needs to be done next – determine who will do it and when they expect it to be done.)

What	Who	By When
Start creating Invariants and start adding them to the class diagram	Andy Kim, Gunnar Biebighauser, Ken Ningcharoen	11-18-19
Start creating Contract and Method Specification	Brandon Ho, Adam Taylor	11-18-19

Unresolved Issues: (List problems and issues that hinder your progress.)
Unsure of our class diagram.

Meeting Reporting Form (6)

Date: 11-18-19 **Group Name: BusyBusinessAnalysts**

Start Time: 5:30PM **End Time: 7:00PM**

Attendees: (List the persons at the meeting – not everyone needs to attend every meeting.)

- **Brandon Ho**
- **Gunnar Biebighauser**
- **Andy Kim**
- **Adam Taylor**
- **Ken Ningcharoen**

Topic: (Use descriptive terms, try to be short and informative)

- **Editing the project**

Pre-meeting Materials for Review: (Provide title and location)

- **Class and Method Design**
- **Device to view specification**

Agenda: (What are you planning to do in this meeting – Keep it short and focused on the topic)

1. **Go over the invariants, Contracts, and Method specification form that we made and make sure they are done correctly.**
2. **Go back to Use case and class diagram and fix any mistakes**
3. **Start making outline for the whole project and how it will be labeled**

Description of Activities:

- In this meeting, we went over the invariants, contracts, and method specification form and fixed any mistakes that were made. We also changed the Use Case diagram by taking out a use case that we thought it wasn't a need and changed things like user stories, use case description, etc.

Action Items: (What needs to be done next – determine who will do it and when they expect it to be done.)

What	Who	By When
Finish making the outline for the project	Ken Ningcharoen	12-10-19
Finalize the Invariants, Contract, and Method Specification part	Everyone	12-10-19

Unresolved Issues: (List problems and issues that hinder your progress.)

- Not 100 percent on our class diagram

Meeting Reporting Form (7)

Date: 12-8-19 **Group Name: BusyBusinessAnalysts**

Start Time: 5:30PM **End Time: 7:30PM**

Attendees: (List the persons at the meeting – not everyone needs to attend every meeting.)

- **Brandon Ho**
- **Gunnar Biebighauser**
- **Andy Kim**
- **Adam Taylor**
- **Ken Ningcharoen**

Topic: (Use descriptive terms, try to be short and informative)

- **Finalizing the Project**

Pre-meeting Materials for Review: (Provide title and location)

- **Device to view specifications**

Agenda: (What are you planning to do in this meeting – Keep it short and focused on the topic)

1. **Go over the invariants, Contracts, and Method specification form that we made and make sure they are done correctly.**
2. **Scan out any mistakes that we made in the project**
3. **Fix any mistakes**
4. **Make sure the project outlined correctly with right page numbers.**

Description of Activities:

- This is the final meeting before we turn our project in. In this meeting, we went over the whole project make sure there were no mistakes. If there were, we went back and fixed it. We made some adjustments to the Invariants and Method Specification that we created.

Action Items: (What needs to be done next – determine who will do it and when they expect it to be done.)

What	Who	By When
Go over the project again individually	Everyone	12-10-19

Unresolved Issues: (List problems and issues that hinder your progress.)
None

Assumptions

- ★ A Request Issuance can be for multiple invoices
- ★ The Accounts Payable Office does not handle paying the vendor
- ★ The check pays for one Request Issuance, which can encompass multiple invoices