Abstract and cover page

Table of Contents

[1 Introduction 1](#_Toc389915697)

[2 Design Methodology 1](#_Toc389915698)

[3 Domain Driven Design 1](#_Toc389915699)

[3.1 Components of Domain Driven Design 1](#_Toc389915700)

[3.1.1 Domains 1](#_Toc389915701)

[3.1.2 Sub Domains 1](#_Toc389915702)

[3.1.3 Generic Sub Domains 1](#_Toc389915703)

[3.1.4 Bounded Contexts 1](#_Toc389915704)

[3.1.5 Aggregates, Entities and Value Objects 2](#_Toc389915705)

[3.1.6 Repositories 2](#_Toc389915706)

[4 The Strategic Vision and Design of APS 2](#_Toc389915707)

[4.1 APS Domains and Responsibility Decomposition 2](#_Toc389915708)

[4.1.1 Core Domain 2](#_Toc389915709)

[4.1.2 Sub Domains 2](#_Toc389915710)

[4.1.3 Generic Sub Domains 2](#_Toc389915711)

[4.1.4 Integrating the different domains 2](#_Toc389915712)

[4.1.5 Queries, Single Responsibility Principle and Interface Segregation principle 3](#_Toc389915713)

[5 Tactical Design 3](#_Toc389915714)

[5.1 Domain and feature allocation 3](#_Toc389915715)

[5.2 Project structure 3](#_Toc389915716)

[5.3 Continuous Integration and Development Strategy 3](#_Toc389915717)

[5.4 Specifications by example and Test Driven Development 3](#_Toc389915718)

[5.5 Feature integration specification collaboration 3](#_Toc389915719)

[5.6 Object Orientation Principles 3](#_Toc389915720)

[5.7 Peer Programming and Code Review 4](#_Toc389915721)

[6 Challenges and successes 4](#_Toc389915722)

[7 Conclusion 4](#_Toc389915723)

[8 References 5](#_Toc389915724)

[9 Appendix 6](#_Toc389915725)

[A - Definition of terms or concepts used within the APS system: 6](#_Toc389915726)

# Introduction

, mention how we took a DDD design methodology, which incorporates a strategic design, with an explanation of the components of DDD, using our overarching architecture and how all the systems fit together to illustrate.

Talk about the implementation of our solution at a tactical design level (but not too in depth )

# Design Methodology

The following sections.. will describe all the sections to come as well as detailing how a Domain Driven Design(DDD) approach was taken for determining the design and development strategy.

This was done by doing a high level strategic design first to see if the scope of work fit a level of complexity/triviality necessitating DDD. After the decision was taken to use a DDD approach, the development methodology that was taken was one of TDD using the Specification By Example technique.

Cite how these concepts are known to work well together especially in a DDD context

Discuss creating a ubiquitous language and how a set of definitions (see appendix A ) was created.

Discuss how using the responsibility breakdown simplified the process of work allocation and implementation, which allows for developer swap out if it was needed.

Discuss how the design was not data centric, but domain and functionality design.[ref?]

# Domain Driven Design

Discuss DDD:

What is DDD, naming the components?

Why we chose to take a DDD approach to our project (What is good about DDD and what is not) our expected complexity in the future, and focusing the departments

Discuss development being iterative and test driven, collaborative

When to use DDD, when not to - Lessons we have learnt about it.

## Components of Domain Driven Design

### Domains/Core Domain

### Sub Domains

### Generic Sub Domains

### Bounded Contexts

Explain what they are and site an example

*Bounded Contexts needn't be organized solely by the functional area of an application. They're very useful in dividing a system to achieve desired architectural examples. The classic example of this approach is an application that has both a robust transactional footprint and a portfolio of reports.* [1]

### Aggregates, Entities and Value Objects

Explain what they are and site an example explain

### Repositories

Explain the repository pattern and how are using it

# The Strategic Vision and Design of APS

## APS Domains and Responsibility Decomposition

Discuss breaking the requirements into separate areas of responsibilities or departments/domains (explaining their types) that would deal with data/domain models in their own way – at a high level

### Core Domain

Discuss the decision that was taken of the core business proposition being that of collating and creating statements from multiple providers

Discuss function and responsibilities at a high level and how any changes of how we do business should flow out from there.

### Sub Domains

Discuss that customer registration and business company creation are sub domains and could potentially be separated into their own systems depending on how customers and billing companies grow in the future.

We wanted to allow customer registration and company additions as well as scraping to be scalable and autonomous.

Discuss function and responsibilities at a high level

### Generic Sub Domains

3rd Part scraping component – should we ever want a different one, we are not coupled to it.

Data persistence is also generic by means of the repository pattern. Discuss function and responsibilities at a high level

### Integrating the different domains

Due each of the Domains and their Application Services running autonomously an integration mechanism was required that would provide cross-domain logging, as well as a common language spoken between systems.

Discuss loose coupling

*The core principle behind loose coupling is to reduce the assumptions two parties (components, applications, services, programs, users) make about each other when they exchange information. The more assumptions two parties make about each other and the common protocol, the more efficient the communication can be, but the less tolerant the solution is of interruptions or changes because the parties are tightly coupled to each other.* [2]

Discuss the interaction requirements and how we planned to integrate the different systems once they we split into their own

Explain the event integration service – how it works, how it can be swapped to another integration mechanism

### Queries, Single Responsibility Principle and Interface Segregation principle

Explain the “queries” and how we use the DTO Pattern. [3] Show how they also adhere to single responsibility

### CQS?

# Tactical Design

## Domain and feature allocation

The team had sessions to discuss and define the strategic vision of the project. Following that the team functionally decomposed the strategic vision using the requirements in the brief into differing high level responsibilities and interactions using a non-data-centric-driven approach, but a function and responsibility approach, keeping the reporting/auditing requirements in mind. The responsibilities were then grouped to create sub-domains or areas of expertise. Then features were defined within these groupings allocating effort weighting and priorities to them. The features were the allocated among the team members.

Table 1: Member Responsibilities

|  |  |
| --- | --- |
| **Member** | **Responsibility** |
| Kgang | Scrape Session Data Interpreter |
| Kgang | Scrape Session Data Validator |
| Carlos | Scrape Session Failure Handler |
| Carlos | Customer Registration and Maintenance |
| Wynand | Scrape Session Scheduler |
| Wynand | Customer Account Statement UI |
| Grant | Customer Billing Account Statement Composer |
| Grant | Billing Company Maintenance |
| Jignesh | Auditor |
| Jignesh | Scrape Session |

## Project structure

The APS Solution, can be broken down into 6 categories, being; Unit Tests, Fakes, Domain Models [4], Application Services, the Integration and Published Language [4] as well as common Classes.

A short description and/or reasoning for each of the categories is as follows:

### Unit Tests

All Unit Test Projects are separated from the Classes being tested, so that when the APS Solution no longer is a Prototype, the solution can be deployed without them based on a customised Build Configuration [5].

### Fakes

The Fakes project is a collection of placeholder Implementations of all of the Aggregate Repositories in the solution. These placeholders facilitate testing functionality acting as rudimentary in-memory collection stores. The intention was to design the solution around the data, but around the Domain requirements, with the understanding the data structure will follow.

### Domain Models

The Domain Model projects are what was described previously in section 4.1. They house Domain Aggregates, Entities and Value Objects performing a specific business function.

Examples are of managing Customers, managing Billing Companies or even running the *Scrape Sessions.*

Each project is isolated for any Software Developer who is working on the Domain Model to work independently of anything else.

### Application Services

An Application Service in the context of the APS Solution is a Class and project that has the responsibility of integrating and coordinating a corresponding Domain Model Project to other Domains, by reacting to Events internal and external to the Domain Model Project.

An example of this is the *Aps.Customer.ApplicationService* project.

The main purpose of this Service is to tell the Customer Aggregate to perform certain actions based on Integration Events being raised from other Domains, such as telling the Domain Model to store a reference to an *Account Statement* when an *Account Statement* has been *composed* and then stored the *Aps.AccountStatements* Domain*.*

Other Domains are notified using the *Event Integration Service* from the Application Serviceof changes within the Customer Aggregate, such as when a new *Billing Company Account* has been *added* to the Customer Aggregate.

The Application Service could have be placed in the same project as the corresponding Domain Model project, however, the decision was taken to separate these to allow the Domain Model logic to be isolated from the processing logic for ease of development and future Domain Model or Application Service extension.

### Integration and Published Language

The *Aps.Integration* project is analogous to an *Open Host Service* [4] as defined by Eric Evans employing the DTO Pattern [4] with specific queries and common Classes to share data in a unified way amongst differing Domains, without exposing the internal Classes of the Domains.

The project also contains the Event Integration Service which allows publishing of and subscribing to a common set of events [pub/sub – event store]

### Common Classes

The common classes or *Aps.DomainBase* project contains Base Classes used by all Domains as well as the *Caliburn.Micro Event Aggregator* allowing each Domain to have the ability to *channel events from multiple objects into a single object to simplify registration for clients* [6].

Currently our Domains have a single Aggregate, but should the Domain grow to more, integrating the Application Service is simplified, and done in a uniform manner. The Event Aggregator allows for decoupling the handling service from the Event raiser, as well as allows for asynchronous event handling.

## Continuous Integration and Development Strategy

Iterative development

Github ( point to url for Joshua to see )

Problems encountered and resolutions taken (see section 6)

## Specifications by example and Test Driven Development

In recent years delivery speed has been become a key component is Software Development [7]. When creating effective Requirement Documentation the goal is to create just enough Documentation at the right time and for the correct audience [8]. Since a proof of concept was being developed a decision was taken that the requirements gathering technique selected needed to be lightweight, easy to maintain and verifiable. Three requirement gathering techniques were investigated, these included Use Cases, User Stories and Specification by Example. The first technique that did not meet the stated requirements was Use Cases due to the fact that they are hard to write and maintain. Another disadvantage was due to the fact that they could not be verified directly on the system created to make sure all requirements were met.

The next technique that was evaluated was that of writing User Stories. This technique met the requirements of needing to be lightweight and easy to maintain. User Stories are also however not verifiable and do not directly test the system. In the end Specification by Example was selected due to the fact that it is lightweight, easy to maintain and verifiable. It allowed for changes to be implemented more efficiently, this was due to having a reliable source of information on system functionality [7]. It also allowed for a higher quality product, this was due to expectations being clearly defined through the use of examples [7]. Another benefit was that it ensured a shared understanding of all expectations by all team members [7].

Each team member was assigned a part of the problem domain and had to create Specifications by using examples about the domain. These specifications were then reviewed by other team members. This led to the team having a shared understanding of the system requirements. Specification by Example provided the team the right documentation and the right time, this fit in well with the iterative development approach.

Test Driven Development (TDD) is the process were a developer writes a failing Unit Test. The second step is to write some code to make the test past. Lastly the developer refactors the code to be as simple an implementation of the tested feature as possible [9]. Features of the system were developed using a Test Driven Development approach. This approach gave the team the ability to test if the implementation works as well as if the design is well structured [9].

Using a Test Driven Development approach afforded the team the following benefits. Writing Unit Tests forced the Developer to have acceptance criteria of what is the definition of done [9]. The approach assisted in driving out a loosely coupled component due to the fact that it had to be written in a way that allowed it to be tested in isolation. The team was able to refactor system changes and have immediate feedback in the form of test results on whether the changes have had any undesired effects.

Discuss how features were tackled by means of specifications by example with 1 or two examples

## Feature integration specification collaboration

Discuss how TDD and integration took place by means of specification by example.

e.g. As a scheduling engine I need create default scheduling when a customer adds a billingcompanyaccount. This allowing integration to be facilitated without actually integrating. Compare this to interface contracts

## Object Orientation Principles

Mention how the implementation follows good OO principles

Dependency injection for DIP – relying on abstractions vs concretions

Single Responsibility – one reason to change examples

ISP – queries tailored to consumers at the lowest level.

## Peer Programming and Code Reviews

### Integration sessions

### Code / Test Reviews

### 2 Peers Programming

# Challenges and successes

# Conclusion

Was DDD good? Was TDD good? Where can we improve on our design?

# References

|  |  |
| --- | --- |
| [1] | D. Laribee, “An Introduction To Domain-Driven Design,” Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/en-us/magazine/dd419654.aspx. [Accessed 03 06 2014]. |
| [2] | G. Hohpe and B. Woolf, Enterprise Integration Patterns - Designing, Building, And Deploying Messaging Solutions, Addision Wesley, p. 38. |
| [3] | M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee and R. Stafford, Patterns of Enterprise Application Architecture, Addison Wesley, 2002, pp. 347-351. |
| [4] | E. Evans and M. Fowler, Domain-Driven Design -Tackling Complexity in the Heart of Software, Prentice Hall, 2003. |
| [5] | Microsoft, “Understanding Build Configurations,” Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/en-us/library/kkz9kefa.aspx. [Accessed 09 06 2014]. |
| [6] | M. Fowler, “Event Aggregator,” [Online]. Available: http://martinfowler.com/eaaDev/EventAggregator.html. [Accessed 09 06 2014]. |
| [7] | G. Adzic, Specification by Example: How Successful Teams Deliver the Right Software, Manning Publications, 2010. |
| [8] | [Online]. Available: http://www.agilemodeling.com/essays/agileDocumentation.htm. [Accessed 10 06 2014]. |
| [9] | S. Freeman and N. Pryce, Growing Object-Oriented Software, Guided by Tests, Addison-Wesley Professional, 2009. |
| [10] | Microsoft, “Class Library,” Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/en-us/library/d11h6832(v=vs.71).aspx. [Accessed 09 06 2014]. |

# Appendix

## A - Definition of terms or concepts used within the APS system:

|  |  |
| --- | --- |
| **Term/Concept** | **Definition** |
| Customer | Person or persons who register as a customer of the APS system |
| Customer Registration | Details of customer used/stored on APS |
| Billing Company | Business that APS interacts with to retrieve customer statements from on behalf of customers |
| Customer Billing Account | Credentials and information pertaining to the account information as held by a customer at a billing company |
| Scrape Session | Process or workflow used by APS to collect, Interpret, Validate and compose statements for a customer from a billing company |
| Scrape Session Data | Information received from a billing company via the scraper for a customer |
| Scrape Session Converter | Conversion of scrape session data into an APS specific format determining success or failure of the scrape session. |
| Scrape Session Failure Handling | Processing of different errors that could be returned in the Scrape Session data |
| Scrape Session Data Pairs | Key value pairs of data returned from the billing company when scraping converted into the APS format |
| Scrape Session Validation | Process of taking the Scrape Session Data Pairs and analysing them for inconsistencies and performing differing forms of integrity checking |
| Customer Billing Account Statement Composition | Creation of a customer statement from valid Scrape Session Data Pairs |
| Scrape Session Queued | Defines that a Scrape Session has been stored for later triggering |
| Scrape Session Scheduler | Means by which Scrape Sessions are stored and retrieved for execution |
| Static page on front end | Non-customer interactive web page which may/may not pull data from a data storage mechanism and display to a customer |

## B – Domain Integration Diagram

