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

### 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. Note that billing companies doesn’t have its own application service as it doesn’t send events out at all, but should for deleting..

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

# Tactical Design

## Domain and feature allocation

Describe who did what at a high level and note these sections will be discussed in detail in the individual reports.

## Project structure

Explain solution and project structure

## Continuous Integration and Development Strategy

Iterative development

Github ( point to url for Joshua to see )

Problems encountered and resolutions taken (see section 7)

## Specifications by example and Test Driven Development

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.

## Object Orientation Principles

Mention how the implementation follows good OO principles

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

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

# 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. |

# 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

