DRAFT - ICT Project Guidance

Custom System Component Design

Version:

0.1

Author:

Sky Sigal, Solution Architect

## Description

This document describes the structuring of components for a system to facilitate incremental changes while having fast delivery cycles, by having prescribed locations features for new features as they are built.

## Synopsis

Non-trivial information systems that are expected to become value adding assets versus becoming maintenance and reputational liabilities follow component layout patterns that have been developed from 60 years of computer engineering experience as opposed to delivery fads.

## Contents

[Description 1](#_Toc145232973)

[Synopsis 1](#_Toc145232974)

[Contents 2](#_Toc145232975)

[Synopsis 3](#_Toc145232976)

[Introduction 3](#_Toc145232977)

[Heading Level 3 3](#_Toc145232978)

[Heading Level 4 3](#_Toc145232979)

[Appendices 4](#_Toc145232980)

[Appendix A - Document Information 4](#_Toc145232981)

[Images 4](#_Toc145232982)

[Tables 4](#_Toc145232983)

[References 4](#_Toc145232984)

[Review Distribution 4](#_Toc145232985)

[Audience 4](#_Toc145232986)

[Structure 4](#_Toc145232987)

[Diagrams 4](#_Toc145232988)

[Terms 5](#_Toc145232989)

## Introduction

BOSSCARD/ RAID: Background [], Objective, Options, Scope[In/Out], Stakeholders [Users], Constraints, Assumptions, Risks, Dependencies, Decisions, Deliverables.

## Risks

Most IT Projects fail to deliver to expectations.

## Issue

While a fair share of the reasons for failure can be attributed to poor stakeholder, persona and user analysis, leading to poor requirement gathering, a significant share of failure is attributable to poor architecture and layout of components such that adds friction, impedes change, often leading to a need to implement costly redesigns, halting delivery, impacting both cost and reputation.

## Resolution

This guidance document provides a short but usable guidance on how to structure a system’s components that will permit the early delivery, while permitting changes later that will mitigate a later need for redesign.

## Patterns

Before addressing system design, it is important to outline the Patterns that influence the design, and anti-patterns that the design avoids using.

### Anti-Patterns

#### Misunderstood Agile

Agile was developed by some of the best senior software developers of their time, working on tremendously impactful projects, at forefront of software development, and being paid tremendous salaries, within extremely well-funded startups and organisations. Their years of experience meant that not only were they good, but had been exposed to many patterns in different environments. The team that developed the manifesto’s principles had no beginners or intermediate developers on it that needed, nor other stakeholders than developers.

In contrast, we take it as an acceptable characterisation that the project you are working on – probably an enterprise system -- is not as well funded, with anywhere near the same autonomy, and the teams assembled is not composed of the brightest and best of their generation.

#### Working Software Over Comprehensive Documentation

It is a recurring myth amongst intermediate developers that a sign of excellent in craftsmanship is the ability to not need direction, nor need to leave direction for others. That the code itself is sufficient to explain its purpose.

This is a flawed perspective in several aspects:

* it doesn’t account for the reality that healthy teams avoid being composed of only highly paid senior developers, and instead – for business continuity reasons -- include the intake and mentoring of junior (cheaper) talent and expose them to their ability to non-trivial work,
* developers overrate their own abilities, considering that what they develop cannot be improved on much, and by extension should be quickly understandable.   
  Whereas the reality is that their code is only one of many options, so concentration and reframing is often required to understand the code.
* Code is for computers. Comments is for humans.  
  While professional developers become fluent in code, it remains a form of communication destined for computers, not humans, and therefore is time consuming to concentrate to understand its purpose. Concentration and time cost money.

# Component Design

## Domain Driven Design

Domain Driven Design (DDD) is a methodology used to organise system logic into logical groups to improve maintainability and modularity of the system over its whole service lifespan.

The basic approach is the organisation of a service into the following groupings:

* Presentation: the presentation of information to end users
* Interface: the technology dependent interface to the system.
* Application: the presentation technology agnostic layer that orchestrates calls to both the Infrastructure Domain and Business Domain
* Common Models:
* Infrastructure Domain: the orchestration of dependency technology services (eg: Identity, Object Mapping, Caching, Data Storage, Searching, Notification, Malware Detection, etc.)
* Business Domain: the assembly of business specific logic.

Key aspects to note of the approach are:

* the Business Domain is kept strictly separate from the Infrastructure Domain, such that neither assembly references the other.
* Models needed outside of either the infrastructure or business domain assemblies are placed in the Common assembly.

Appendices

Appendix A - Document Information

### Images

[Figure 1: TODO Image 2](#_Toc144995112)

### Tables

[Table 1: TODO Table 3](#_Toc145048484)

[Table 2: TODO Table 2 3](#_Toc145048485)

### References

**There are no sources in the current document.**

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
|  |  |
|  |  |
|  |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Terms

Refer to the project’s Glossary.

##### IT

: acronym for Information, using Technology to automate and facilitate its management.

##### ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. IT is a subset of ICT.