AgileWorks

Is Agile Design an Oxymoron?

TIM GUAY, PMI-ACP, CSD, CSP, ICP-ASD, TOGAF CERTIFIED
WWW.AGILEWORKSINC.COM
TIM@AGILEWORKSINC.COM

Objectives

- Learn the importance of just enough design upfront
- Learn key Agile design and architectural concepts and techniques
- Learn how good design and architecture drives business value

Tim Guay, Agile Coach and Trainer

- Agile Practitioner since 2002
- CSD, CSP, PMI-ACP, ICP-ASD, TOGAF Certified, etc.
- Sits on the ICAgile DevOps Certification Committee
- Authored the first Agile Software Design certification course available in North America
- Clients include; JP Morgan Chase, Citigroup, Accenture, Cisco, Intuit, State of Washington, and the Royal Canadian Navy (to name a few).

The Myth that Agile Design is an Oxymoron

- Persistent myth that Agile Design is an oxymoron
- This opinion is a natural reaction against the waterfall Big Design Up Front (BDUF) mentality
- Also results from the misinterpretation of the "The best architectures, requirements, and designs emerge from self-organizing teams" Agile principle as design and architecture will magically emerge as the team codes
- Reinforced by the emphasis on deferring (design) decisions until the last responsible moment as well as on the emphasis on emergent architecture and design

The Dangers of this Myth

- Code that is riddled with technical debt
- Impossible to implement DevOps effectively
- Brittle, hard-to-understand and modify code, design and architecture that requires heroics to keep it working
- Resulting in higher TCO and reduced business agility
- Undermines the business value to be delivered

The Dangers of this Myth

- Need for significant code, design, and architectural refactorings in order to modify or maintain the system
- No time for innovation as we have to deliver by the end it the sprint
- Architecture reflects the Big Ball of Mud antipattern
- Nobody really understands the system

Coding without doing any design work is not Agile!

it is hacking, pure and simple, and will result in an instant legacy system…

How Do We Approach Agile Design?

Continuous attention to technical excellence and good design enhances agility

- Maintain focus on business value as the key driver for the design
- Develop a shared and collective knowledge of the system
- The code is co-owned by the designer via their design and the design by the developers via their code
- Emphasize excellence in design
- Do only just-enough design that will ensure a coherent Fit-For-Purpose design
- Create the simplest architecture and design that works

How Do We Approach Agile Design?

Good Agile Design results in

- A robust, scalable, and modifiable architecture and design
- A design that results in testable, maintainable and reusable code
- A design that results in code and tests that are amiable to Continuous Integration (CI) and Continuous Deployment (CD)
- Optimized for delivery of optimal business value and business agility
- Higher ROI and lower TCO

Intentional Versus Emergent Architecture and Design

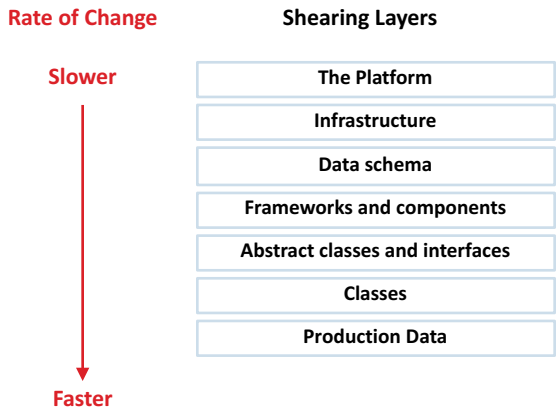
- In many Agile circles there is an over-emphasis on emergent architecture and design
- With emergent design, developers starts delivering functionality and lets the design emerge from the ensuing code
- As we have seen, we ignore intentional design and architecture at our peril
- Creating an Intentional Architecture and Design allows us to;
  - Avoid large-scale and unnecessary architectural and design rework
  - Manages the risk of adverse Impact on currently deployed systems and users
  - Focuses on providing the architectural and design underpinning for delivery of the business value
  - Improves usability, extensibility, reliability, performance and maintenance

Intentional Architecture and Design

Intentional Architecture is developed by

- Levering common architectural patterns, constraints and implementation technologies
- Creating and maintaining the architectural runway
  - Provides sufficient infrastructure to support the incorporation of near-term product backlog items without needing potentially destabilizing refactorings
  - Runway should support six months worth of product backlog
- Encouraging innovation by
  - Setting aside time for researching the best solution and running architecture & design spikes
  - Proving out the design using a Minimum Viable Architecture (MVA)

Intentional Design and Shearing Layers

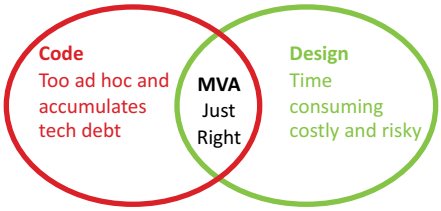


Intentional Design and Shearing Layers

- Understand the business goals and requirements
- Determine what design will support the business goals and requirements
- Understand the rate the layers change
- Design so artifacts that change at similar rates are together
- Ensure the layers are loosely coupled
- Plan for leveraging automation, layering, design patterns, and design techniques
- Use architecture spikes to understand the impact of your architectural and design decisions on the shearing layers as well as on the business architecture

Minimum Viable Architecture

- Contains the intentional architecture – and supports evolutionary architecture
- Balance between jumping into coding and upfront architecture
- Supports the business vision
- Helps avoid the Big Ball of Mud antipattern
- Addresses the shearing layers
- Mitigates architectural risk
- Implemented via a walking skeleton



Model Storming

A form of brainstorming designed to quickly develop high-level architectures and designs

- Collaborative: involves team, customers, and SMEs
- Driven by the customer's business requirements
- Artifacts are low-fidelity
- Focus is breadth over depth
- Produces a holistic view that is understandable by both business and IT
- That will generate a design that delivers the desired business value
- We will briefly touch on two techniques; CRC Cards and Domain-Driven Design

CRC Cards

- Stands for Class-Responsibility-Collaborator
- Collaborative modeling technique brings together customers and the team
- Uses index cards that capture the CRC information
- Should be physical cards if at all possible
- Created during a collaborative workshop that identifies classes and scenarios that model system behavior
- Scenarios are role-played by the workshop participants

CRC Cards

- Stands for Class-Responsibility-Collaborator
- Class is explained to the customer as some 'thing' that exists in the business domain; such as a customer or and order
- Responsibility as an action or activity that the 'thing' does; such as a customer places an order
- Collaborator is any other 'thing' that the first 'thing' need help from

CRC Cards

Class Name	
Responsibility	**Collaborators**

CRC Cards

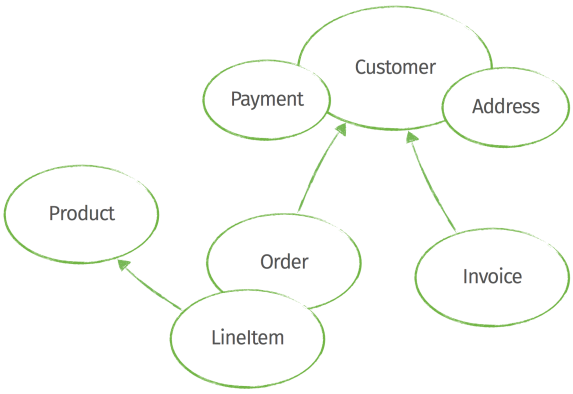
CRC Advantages

- Low-fidelity makes collaboration easy
- Can use the cards anywhere
- Inexpensive
- Can even simplify layout further for initial Design-in-the-Large workshops
- Useful for helping users understand the object-oriented paradigm

Domain-Driven Design

- Collaborative modeling technique brings together customers and the team
- Focuses on creating a conceptual model the core business domain and the business problem to be solved
- Concepts include;
  - Context – the setting that determines the meaning of a word or statement
  - Domain – a area of knowledge
  - Model – an abstract description of the domain
  - Ubiquitous Language – a language that describes the domain model that is understood by both the customers and team

Domain-Driven Design



Behavior-Driven Development (BDD)

- Three Amigos collaboration
  - Folks from business, development, and testing
  - Meet as needed, but all three should be involved in ATDD discussions
  - Use domain-specific language to keep everyone on the same page
- Focus is on business value by understanding the behaviors that support the business goals for this user story
- As well as the acceptance criteria that will be used to validate that the business value has been realized

Behavior-Driven Development (BDD)

- Design thinking: The Given-When-Then structure
  - Guides the design of the application flow and state changes
  - Helps identify the inputs, processing, and outputs
  - Refine the low-level design prior to coding
- Design thinking: Scenarios
  - The ATDD process requires the creation of a scenario for every path
    - This aids in our visualizing and determining the modularization of our design

Behavior-Driven Development

- A BDD spec starts with a short narrative, with the standard user story format often used
- Then scenario is described for each behavior path (think use cases) using a Given-When-Then format
  - Given is the starting state
  - When is the event trigger or input
  - Then is the end state or output
  - Use the And keyword to add extra conditions
- Sample input and expected output can also be included

Behavior-Driven Development

Feature: Calculate Shipping
As a Online Shopper want to calculate shipping
so that I know how much my shipping is
Rules:
If order is equal to or greater than than $30 the shipping is free

Scenario Outline: Order greater than 3$0
Given: Calculate Shipping Clicked
When: <0rder> greater than $29.99
Then: Display <shipping>

Examples
| order | shipping |
| 30 | $0 |

We have covered our objectives

- Learn the importance of just enough design upfront
- Learn key Agile design and architectural concepts and techniques
- Learn how good design and architecture drives business value

Time for

Questions and Discussion