Course Outline

Domain Driven Design and Event Storming

Course Summary

Description

Much of the software being developed or upgraded today is “mission critical,” an organization becomes so dependent on their software infrastructure that when the software fails, the organization can no longer function. However, when software become mission critical it also becomes very complex and difficult to work with. The major contributing factor to the complexity of the mission critical systems is the complexity of the real world business domain which the software automates. Studies on the failures of both IT systems and projects conclude that the inability to manage domain complexity is the major root cause of the failures.

Domain Driven Design (DDD) was introduced by Eric Evans as a methodology to enable developers to create domain models of complex domains based on an intensive iterative study of the domain and refinements of a domain model. DDD, which originally focused only on the static structures in the domain, is complemented by Event Storming (ES) which integrates how the dynamic processes interact with the domain structures by analyzing the events that can occur in the domain and the domain’s responses to those events. The modern approach to DDD now incorporates the ES process as part of its modeling cycle.

The course starts with a look at the complexity problem and the overall solution provided by the DDD-ES approach. This is supplemented with an overview of domain modeling best practices from both a structural and dynamic perspective. The basic model elements or interfaces are covered in detail – factories, repositories, entities, aggregates, value objects etc. Covered in detail is the knowledge distillation or crunching process, the core investigative technique of DDD-ES and how this process is used to create robust and elegant models.

The course includes more recent work that was not part of the original formulations of DDD and ES such as the process re-engineering aspects of the domain, the inversion of control design principle and other patterns of implementation that have become part of the domain driven design approach since the original Evans material was presented.. Also included is an overview of how other techniques like the Stanford Design Process, Agile, DevOps, Lean Engineering and Agile testing integrate with the domain driven design process.

Since the course deals with modeling, there will be a continuous modeling exercise that will be used to illustrate all of the principles and concepts presented in the course

A fourth day is available as an add-on to the basic course which focuses on moving from design to code and is intended for developers

Topics

- Complexity and Design
- Domains, Architectures, and Knowledge Crunching
- Domains, Contexts, and Ubiquitous Language
- Entities, Value Objects, and Aggregates
- Event Storming
- Factories, Repositories, and Events
- Services and Specifications
- Moving from Model to Design
- Re-engineering

Course Summary (cont'd)

Audience

This course is targeted at developers, business analysts, and domain experts who need to work collaboratively to build applications based on complex domains.

Prerequisites

A familiarity with data modeling or object oriented domain modeling is helpful, as well as an exposure to program design using interfaces, although these are not strictly necessary.

Duration

Three days

Domain Driven Design and Event Storming

Course Outline

I. *Complexity and Design*
- A. Complexity – the motivating problem
- B. Sources of complexity – domain, design and environment
- C. Overview of the DDD-ES approach
- D. The DDD Quadrants
- E. ES: processes, scenarios, events and responses.
- F. Architectural and Functional layers and dependencies

II. *Domains, Architectures, and Knowledge Crunching*
- A. Domains and domain models
- B. Knowledge crunching and distillation
- C. Breakthrough iterations
- D. Integration with Stanford Design process and Lean technologies
- E. Abstraction and Dependency Inversion Principle
- F. Types of complexity in homogeneous and heterogeneous domains
- G. Modeling Principles – Dynamic and Static

III. *Domains, Contexts, and Ubiquitous Language*
- A. Identifying sub-domains and why they exist
- B. Bounded contexts and modeling
- C. Ubiquitous Language
- D. Continuous Integration
- E. Processes processes and events within and across bounded contexts.

IV. *Entities, Value Objects, and Aggregates*
- A. More modeling concepts
- B. Entities and identifiers
- C. Value objects
- D. Aggregates
- E. Aggregate invariants
- F. Aggregate interfaces
- G. Conceptual contours
- H. Context maps and shared kernels

V. *Event Storming*
- A. Scenarios, story mapping and data flows.
- B. Commands, events, aggregates and read models
- C. Model, View Controller and related architectures
- D. Actions and responsibilities
- E. Binding events and actions to aggregates
- F. Alignment of structure and function

VI. *Factories, Repositories, and Events*
- A. Object lifecycles
- B. Factories, factory sites, and factory interfaces
- C. Processing invariants
- D. Repositories, queries and repository interfaces
- E. Events – domain versus versus application events
- F. Modeling event processing with aggregate states
- G. Modeling reactive processes
- H. Side-effect free functions
- I. Stand-alone classes
- J. Anti-corruption layer

VII. *Services and Specifications*
- A. Services – when to use them
- B. Inversion of Control and Dependency Inversion
- C. Specifications and business rules
- D. Modules
- E. Assertions
- F. Closure of Operations
- G. Server types

VIII. *Moving from Model to Design*
- A. Implementation patterns for aggregates
- B. Implementation patters for factories and repositories
- C. Implementation patterns for services and specification
- D. Implementing events and processes
- E. implementing a layered architecture
- F. Best practices for implementation

IX. *Re-engineering*
- A. Exploratory testing and analysis
- B. The problem of Legacy structures and processes
- C. Re-engineering the domain
- D. Event storming to develop acceptance tests
- E. Test Driven Architectural and Functional redesign
- F. Risk analysis and cost benefit analysis
- G. Using the Open-Close principle
- H. Implementing design best practices using DDD-ES