TEST-DRIVEN DATABASE DEVELOPMENT

Unlocking Agility

MAX GUERNSEY, III
Foreword by SCOTT BAIN

NetObjectives
Lean-Agile Series

Test-Driven Database Development

Test-Driven Database Development: Unlocking Agility

Table of Contents

Contents
Foreword
Preface
Acknowledgments
About the Author
Chapter 1 Why, Who, and What
- Why
  - Agility Progressively Invades Domains Every Day
  - Agility Cannot Work Without TDD
  - TDD in the Database World Is a Challenge
- Who
  - TDD and OOP
  - Applications and Databases
- What
  - Databases Are Objects
  - TDD Works on Classes, Not Objects
  - We Need Classes of Databases
- Summary

Chapter 2 Establishing a Class of Databases
- The Classs Role in TDD
  - A Reliable Instantiation Process
  - Tests Check Objects

Table of Contents

- Classes in Object-Oriented Programming Languages
  - Making Classes Is Easy: Just Make New Objects
  - One Path: Destroy If Necessary
- Classes of Databases
  - Two Paths: Create or Change
  - The Hard Part: Unifying the Two Paths
  - Real Database Growth
  - How About Making Every Database Build Like Production Databases?
  - All DBs Would Follow the Exact Same Path
- Incremental Build
  - Document Each Database Change
  - Identify Current Version
  - Apply Changes in Order as Needed
- Implementation
  - Requirements
  - Pseudocode Database Instantiation Mechanism
  - Pseudocode Input
- Summary

Chapter 3 A Little TDD

- The Test-First Technique
  - Write the Test
  - Stub Out Enough to See a Failure
  - See the Test Pass
  - Repeat
- Tests as Specifications
  - Tests Arent Tests, They Are Specifications
  - Tests Arent Specifications, They Are Tests
  - Tests Are Executable Specifications
  - Incremental Design

Table of Contents

- Building Good Specifications
  - Specify Behavior, Not Structure
  - Drive Design In from Without, Not the Other Way Around
  - Defining the Design Inside Out
  - Defining the Design Outside In
- Summary

Chapter 4 Safely Changing Design

- What Is Safe?
  - Breaking a Contract Is a Little Bad
  - Losing Data Will Probably Get You Fired
  - Not Changing Design Is Also Dangerous
- Solution: Transition Testing
  - Test-Driving Instantiation
  - Transition Testing Creation
  - Transition Testing Addition
  - Transition Testing Metamorphosis
  - Why Not Use the Public Interface?
- Transition Safeguards
  - Read/Read Transition Tests
  - Run by the Class of Databases on Every Upgrade
  - Backup and Rollback on Fail
  - Making Transition Tests Leverage Transition Safeguards
- Summary

Chapter 5 Enforcing Interface

- Interface Strength
  - Stronger Coupling Languages
  - Weaker Coupling Languages
  - The Common Thread
  - Coupling to Database Classes

Table of Contents

The Problem Is Duplication

Client-Object-Like Enforcement

Creating Demand for a DatabaseDesign Class

Specifying the DatabaseDesign Class

Getting Rid of Duplication with Multiple Client Platforms

What Happens When Coupling Goes Bad?

Eliminating Duplication Between Database Build and Client Code

Decoupling Implementation from Design

Sticking Point: Change

Designs Change Over Time

Document All Versions of Design

Couple to the Correct Version of the Design

Sticking Point: Coupling

Various Clients Couple to Various Versions

Having to Change Everything All the Time Is Duplication, Too

Introducing the Lens Concept

Virtual Lenses

The Current Lens

The New Lens

Summary

Chapter 6 Defining Behaviors

A New Group of Problems

No Encapsulation

Hide Everything

Business Logic in the Database

Knowledge, Information, and Behavior

Information

Knowledge

Behavior

Table of Contents

- Outside-In Development
  - Defining the Test
  - Growing Interface
  - Growing Behavior and Structures
- Justification by Specification
  - Work Against Present Requirements, Not Future
  - Build in Increments
  - Limit Access to What Is Specified
- Summary

Chapter 7 Building for Maintainability

- Never Worry About the Future
  - Look for Opportunities in the Now
  - Design to Information
  - Translate Info and Knowledge with Behavior
- Guard Knowledge with Fervor and Zeal
  - Not Changing Is the Most Dangerous Choice
  - Keep Your Design Natural
- Deal with the Future When It Happens
  - Define New Design
  - Introduce Minimal Changes
  - Get Tests Passing
  - Stop, Think, Refactor
- Summary

Chapter 8 Error and Remediation

- Kinds of Errors
  - Axis: Is the Error Good or Bad?
  - Axis: Is the Error Released or Not?
- Dealing with Good Errors

Table of Contents

Just Fix It

Document Behavior Now

Trace Feature Back to Its Genesis

Dealing with Bad Errors

Unreleased Errors

Released Errors

Catastrophic Errors

Summary

Chapter 9 Design

Structures Versus Design

Structures: Execution Details

Tests and Class Information

What Is Design?

Buckets of Concepts

Mandatory Part of True TDD

Composition and Aggregation

Composition: One Thing with Multiple Parts

Aggregation: Connecting Distinct Things

Reuse

Avoid Developing the Same Thing Twice

Reuse by Composition or Aggregation

Abstraction

Identifying Opportunities for Abstraction

Encapsulating Behaviors

Finding Ways to Allow Variation in Dependencies

Dealing with the Time Problem

Summary

Chapter 10 Mocking

Table of Contents

- Testing Individual Behaviors
  - Why Encapsulate
  - Tests Test Everything Not Under Their Control
  - Controlling Irrelevant Behaviors from Tests
  - Mocking Controls Behaviors
- Mocking in Object-Oriented Programming
  - Setup
  - Decoupling
  - Isolation
  - Integration
- Mocking in Database Design
  - Example Problem
  - Example Solution
  - Composition
  - Aggregation
  - Designing for Testability
- Summary

Chapter 11 Refactoring

- What Refactoring Is
  - Changing Design Without Changing Behavior
  - In the Context of Passing Tests
- Lower and Higher Risk Design Changes
  - Lower Risk: Changing Class-Level Design
  - Medium Risk: Rearranging Behavior Logic
  - Higher Risk: Altering Knowledge Containers
  - This Is Not an Invitation to Skip Testing
- Summary

Chapter 12 Legacy Databases

- Promoting to a Class

Table of Contents

Deducing Initial Version

Pinning the Transition Behavior with Tests

Controlling Coupling

Identifying and Locking Down to Existing Uses

Encapsulating on Demand

Controlling Change

Test-Driving New Behaviors

Pinning Construction on Demand

Pinning Behavior on Demand

Implementing New Behavior

Finding Seams and Components

Finding Seams

Encapsulating Components

Summary

Chapter 13 The Façade Pattern

Encapsulation with a Façade

Explanation of Façade Pattern

New Test-Driven Façade Database

Compositional Alternative

To Encapsulate or Not

Strangling the Old Interface

Transferring Changing Behaviors to Façade

Removing Access and Features When No Longer Needed

Test-Driving Behaviors in the Façade Database

Exposing Legacy Behaviors

Another Way to Do It

New Behaviors

Summary

Chapter 14 Variations

Table of Contents

- Having a Class Is ImportantImplementation Is Not
- Scenario: Skipping Steps
  - Problem
  - Solution
  - The Right Amount of Work
- Scenario: Deviations
  - Problem
  - Solution
  - Solution Applied
- Common Solution
- Summary

Chapter 15 Other Applications

- XML
  - Encapsulation
  - XSD Schemas
  - XSLT Transitions
  - Transition Test XSLT Changes
- File Systems and Other Object Directories
  - Transition Test File System Manipulations
  - Shell Script Transitions
- Data Objects
  - Class Definitions Are Schemas
  - Transition Test the Ugrader Class
  - Code Transitions
- Summary and Send Off

Index