How the SDK Can Support the Broad Spectrum of Development on Blackbaud CRM

- Matt Hall (matt.hall@omaticsoftware.com)
  - Senior Technical Consultant
    - Omatic Software

Agenda

- Introductions
- Omatic Overview
- BBCRM Experience
- Feature Presentation - Complex SDK Customizations
- Q/A

Omatic Software

- Charleston, SC Headquarters
- Founded in 2002
- Blackbaud Technology Partner
- Data integration focus
- Commitment to Nonprofits
- Software Development
  - Product Development
  - Professional Services
  - Support
- Over 2,700 clients worldwide
- 5-time, Inc. 500|5000 Honoree

BBCRM Experience

Professional Services

- Customization
- SDK Training
- System Integration
- Database Migrations

Partnerships

- Blackbaud
- Third Party Systems

Technology

- Data Integration
  - Online
  - P2P
  - Marketing
  - HR
  - Document Management
  - Post to GL/Finance
  - Student Information Systems
- Event Management
- Marketing Automation
- Reporting/Analytics
- Program Automation
  - Grateful Patient
  - Employee Giving

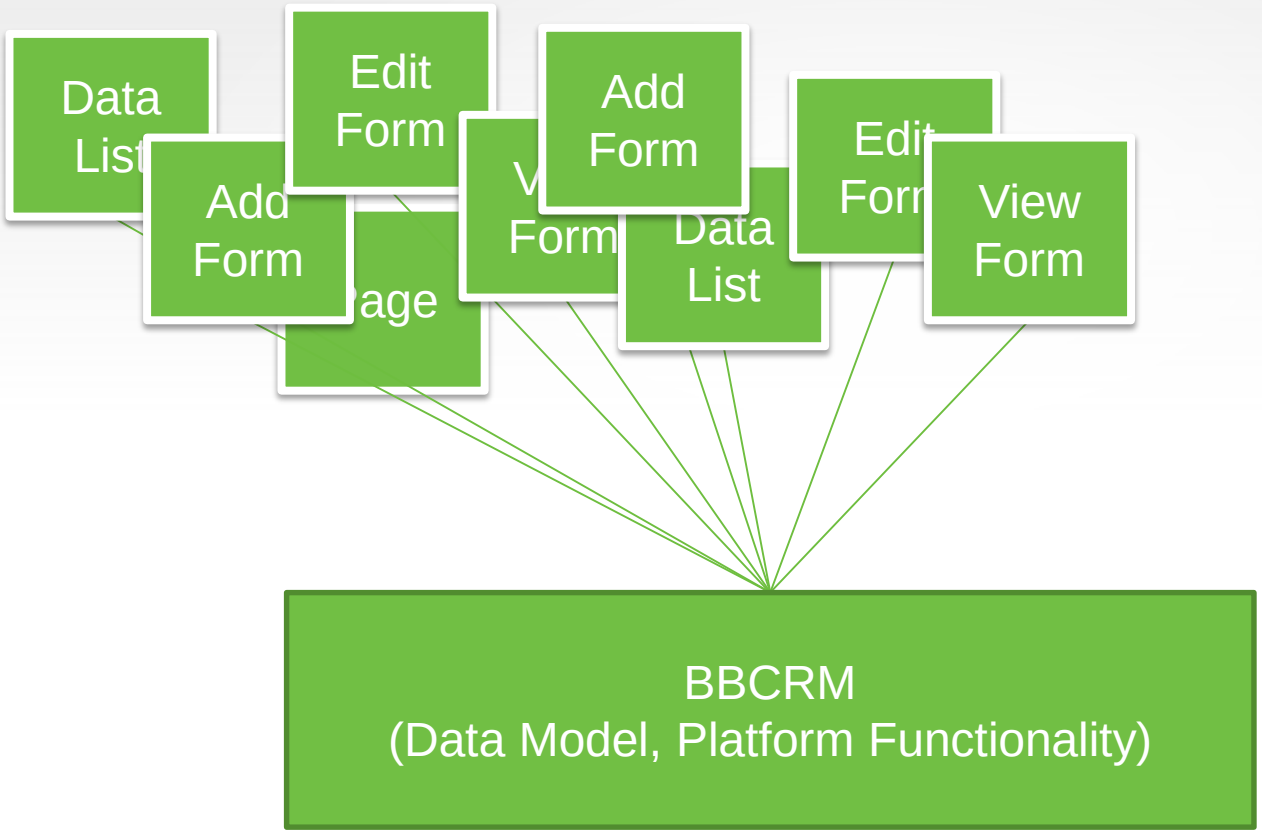
Introductions
Matt Hall – Senior Technical Consultant

- University of Connecticut
  - Senior Systems Architect
  - BBCRM Implementation
  - Integrations and Customizations
- Blackbaud
  - Technical Principal Consultant
  - Project Scoping, Design, Customization and Implementation
- Georgia Tech Alumni Association
  - Senior Programmer
- Development Background
  - .NET (C#, VB.NET), SQL/T-SQL, Microsoft SQL Server Toolset (SSIS, SSRS), ASP.NET, Web
  - Languages (HTML/CSS/JavaScript),
  - Web Services, Enterprise Application Architecture, CRM, Software Development Lifecycle (SDLC), Agile Methodologies, Build and Automation Processes

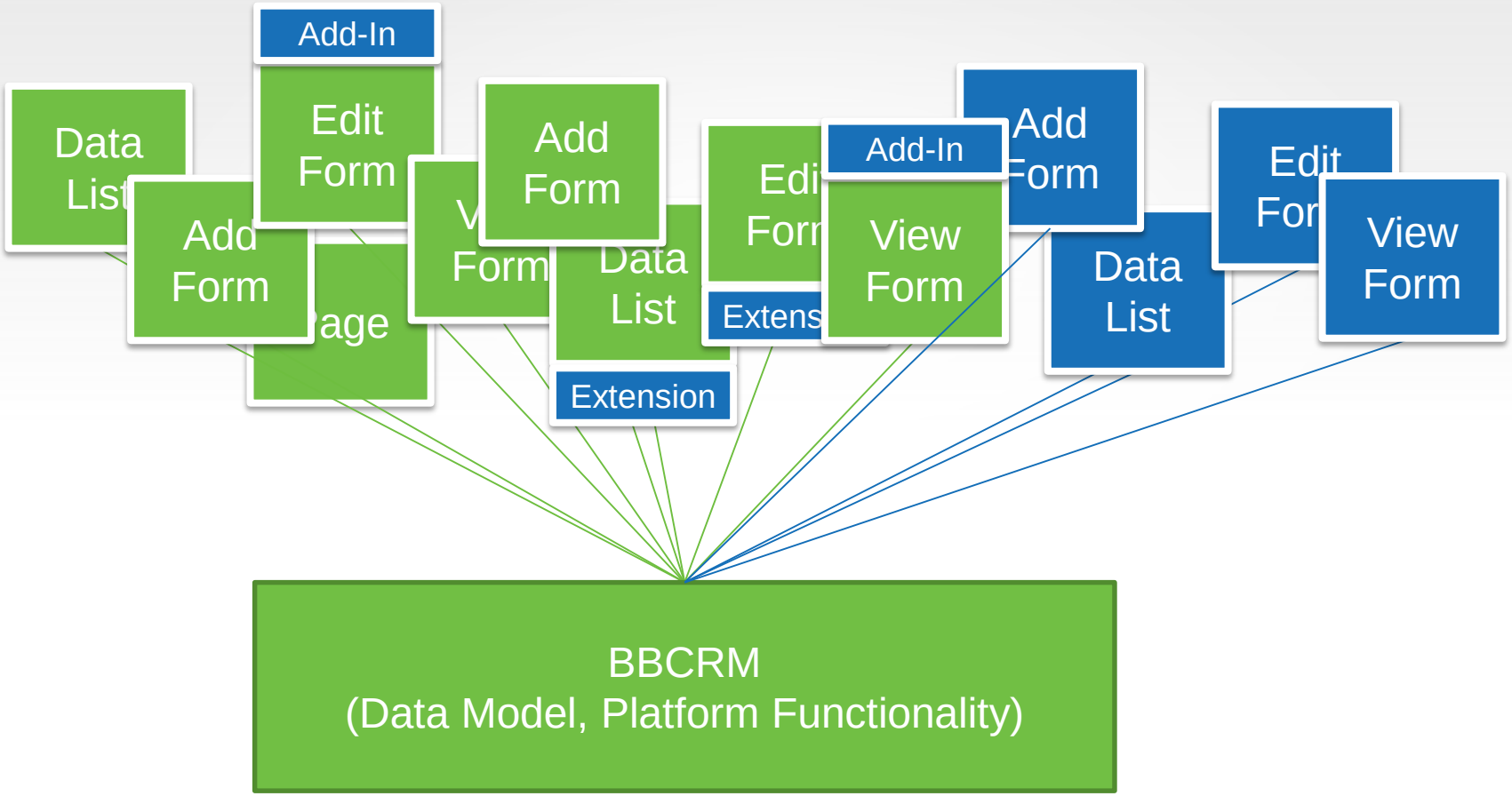
BBCRM SDK and Complex Customizations

- Define: “Lightweight” vs “Complex”
- Limits of lightweight customizations
- Managing complex customizations

Define: Lightweight vs Complex



Define: Lightweight vs Complex

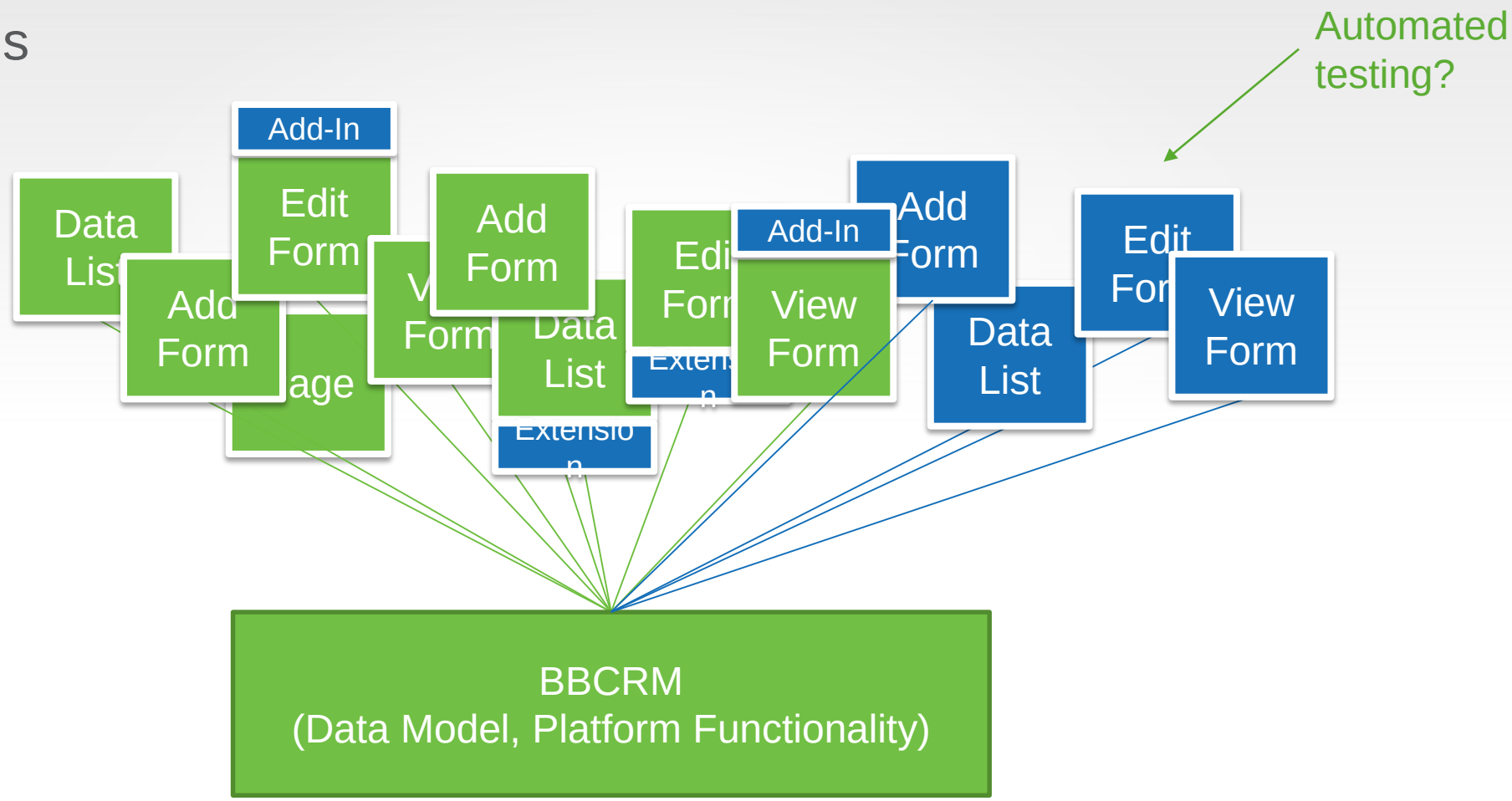


Define: Lightweight vs Complex

- Customize at the surface
  - Developer can augment, not reinvent
- Lightweight = avoid a lot of details
  - Domain (aka object) model
  - UI details
  - App/DB connection (setup, data get/save, teardown)

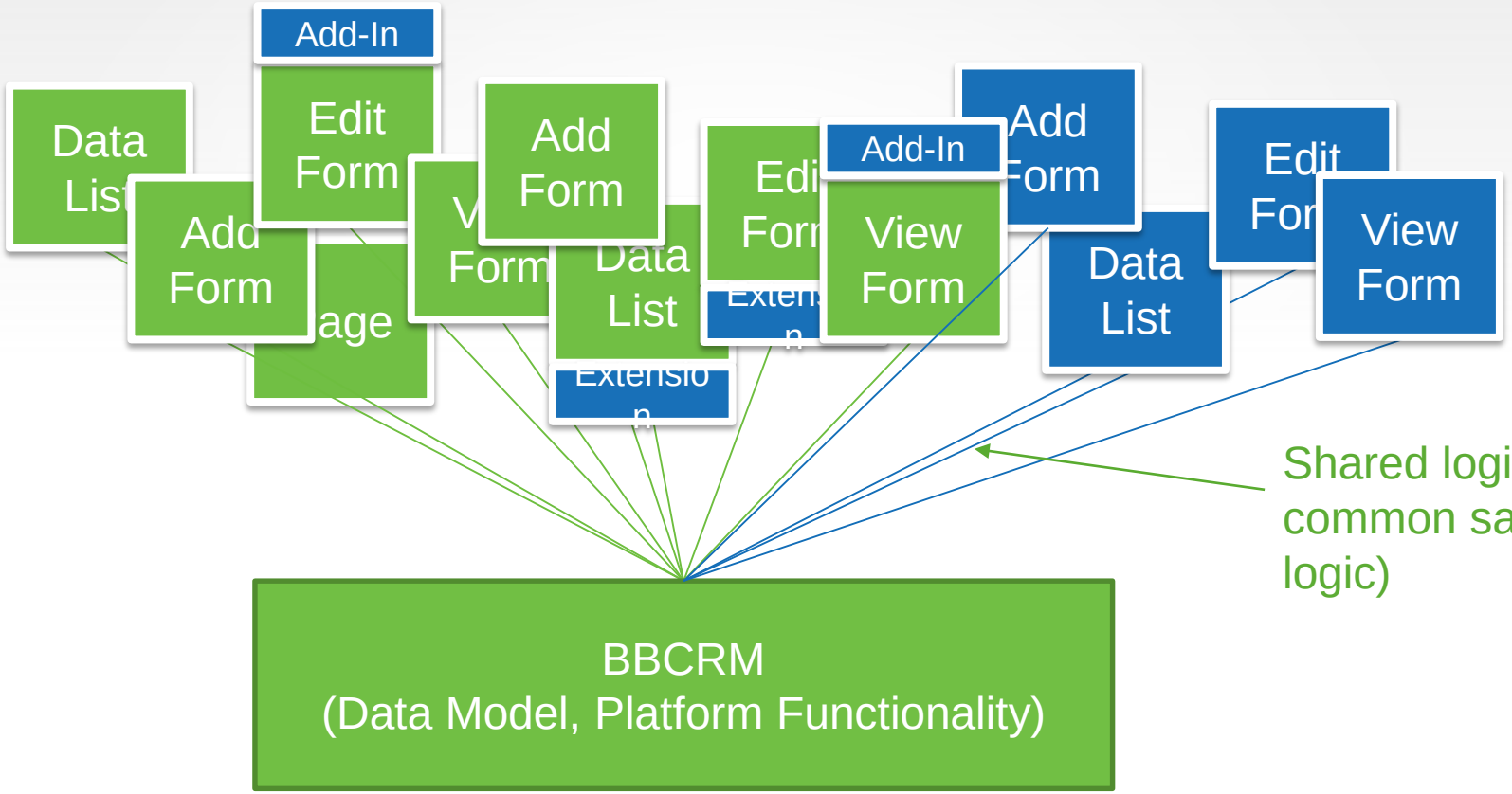
Limits of Lightweight Customizations

- Limits to this



Limits of Lightweight Customizations

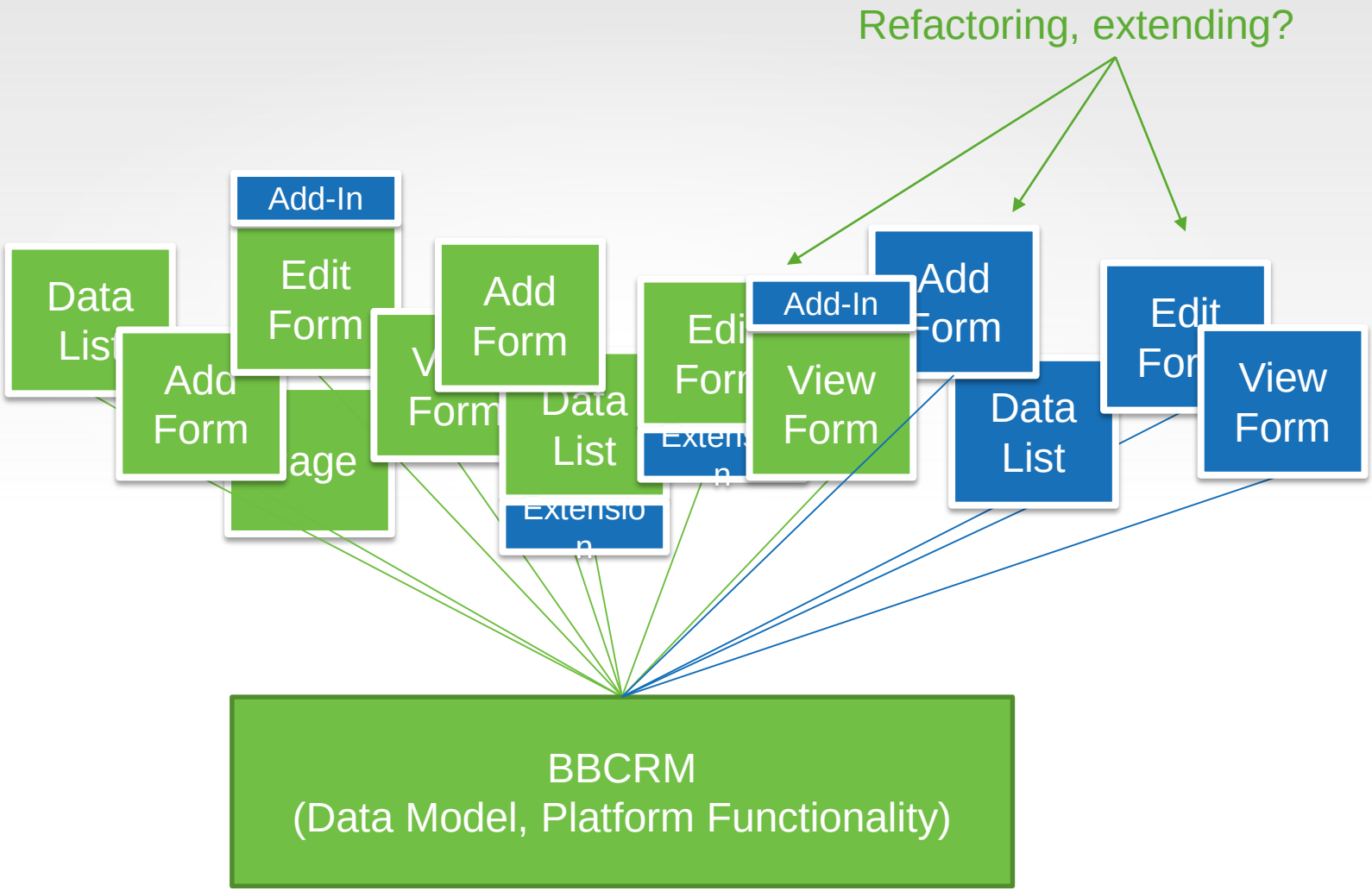
- Limits to this



Shared logic? (Validation, common save/retrieve logic)

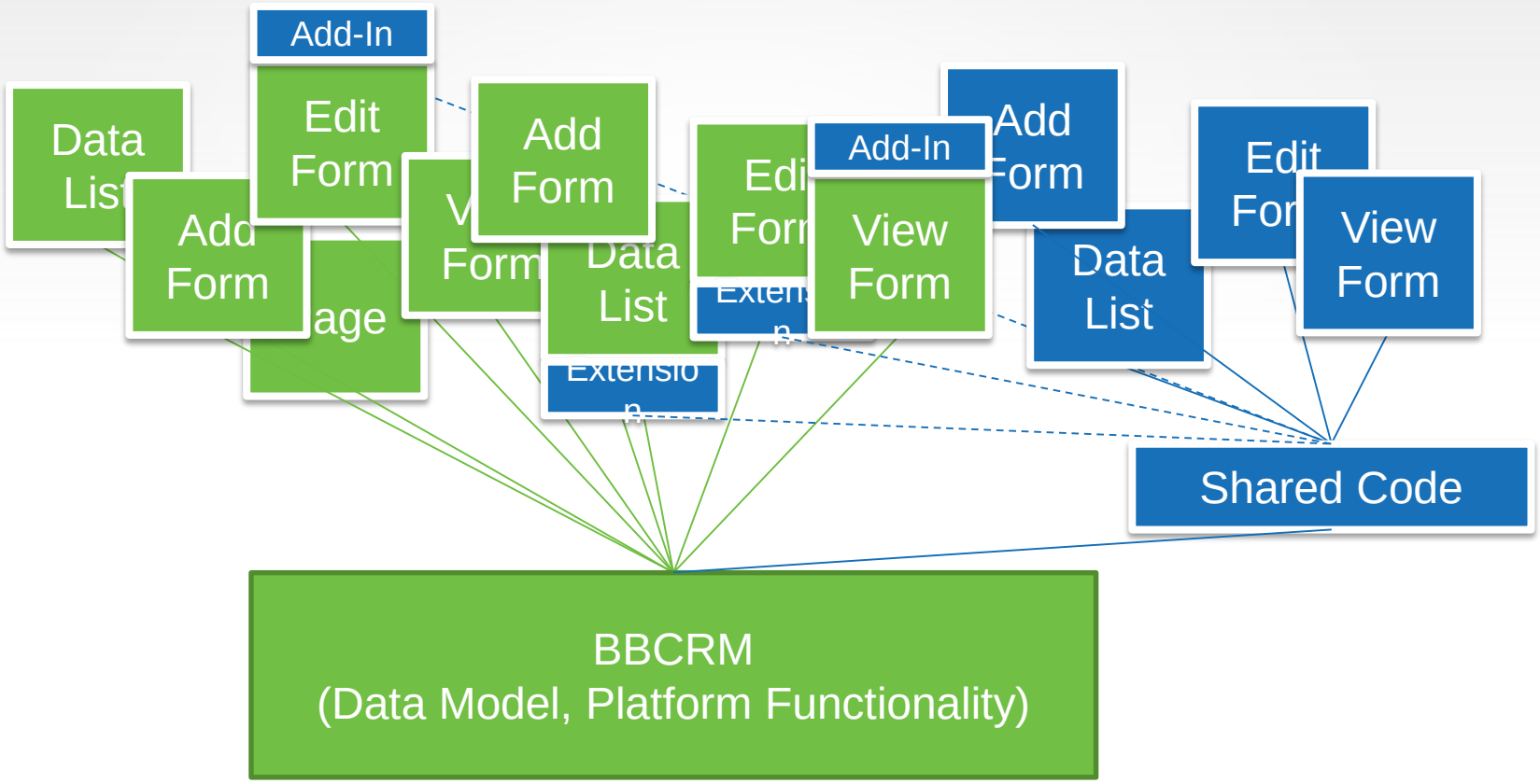
Limits of Lightweight Customizations

- Limits to this



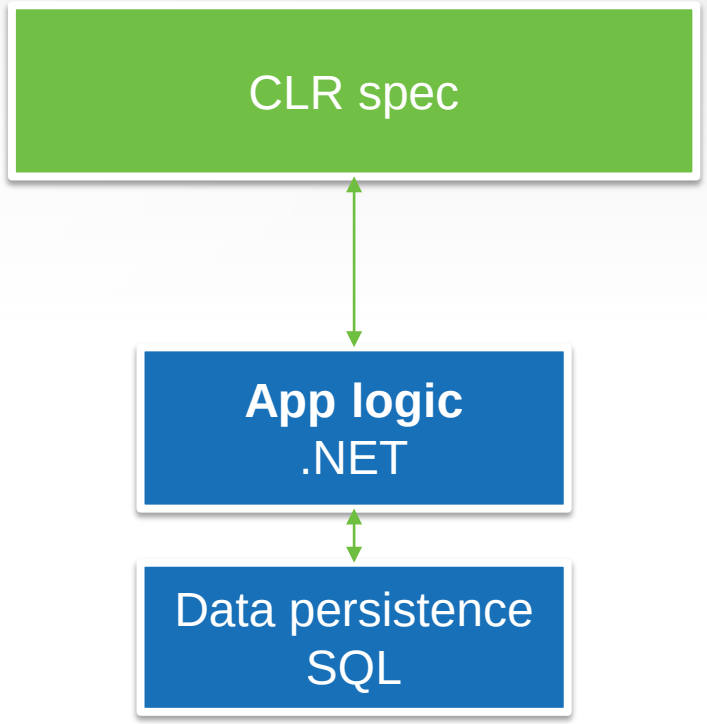
Limits of Lightweight Customizations

- Refactor shared functionality—at cost of complexity



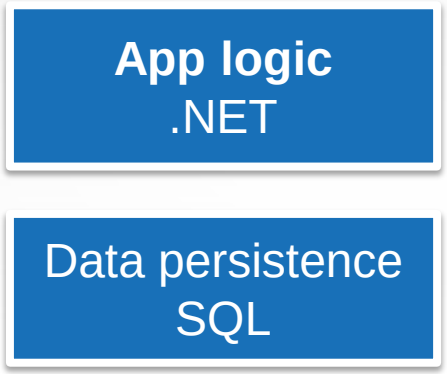
Managing Complex Customizations

- Move logic away from SQL towards CLR (VB.NET or C#)
- Role of SQL:
  - Data persistence (basic CRUD)
  - Set-based operations
  - NOT complex business logic
- Role of CLR
  - Domain model with business rules
  - Abstract away data persistence
- Role of SDK
  - Provide access to domain model



Managing Complex Customizations

- Why?
  - Testability
  - Reusability
  - Maintainability



Decomposed, unit testable objects. Decoupled from DB, BBCRM.

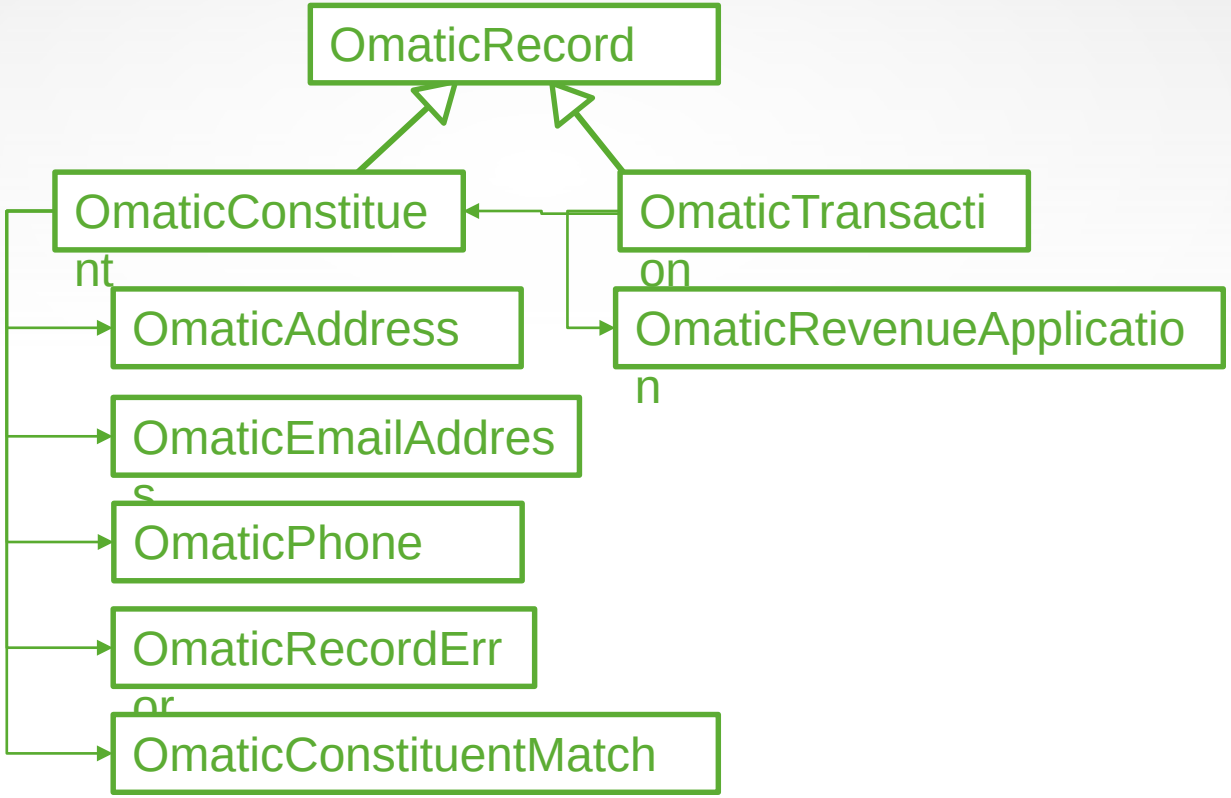
Focus on saving, retrieving objects used in app logic (i.e. ORM). Decoupled from app logic, BBCRM

Managing Complex Customizations

- For example…imported record validation
  - Needs to be testable
    - Check valid, invalid conditions
  - Needs to be reusable
    - Validate on import, user modification, record commit
  - Needs to be modular
    - Common validation routines, organization-specific, data source-specific
- Examples:
  - Ensure imported code table values exist
  - Check for required fields
  - Don’t add bio data if constituent record is matched and already exists

Managing Complex Customizations

- Approach:
  - Domain model in .NET objects



Managing Complex Customizations

- Approach:
  - Data persistence hidden behind interface
  - IDataRepository
    - void SaveConstituent(OmaticConstituent record);
    - OmaticConstituent GetConstituentById(Guid id);
    - OmaticConstituent GetByAlternateId(string alternateIdType, string alternateId);

Managing Complex Customizations

- Approach:
  - Interface for validating records:
    - IRecordValidator
      - IEnumerable<OmaticRecordError> Validate(OmaticConstituent record)
  - Validation runner:
    - For a given OmaticConstituent, iterates collection of IRecordValidator objects, concatenates list of OmaticRecordError objects

Managing Complex Customizations

- Composing things

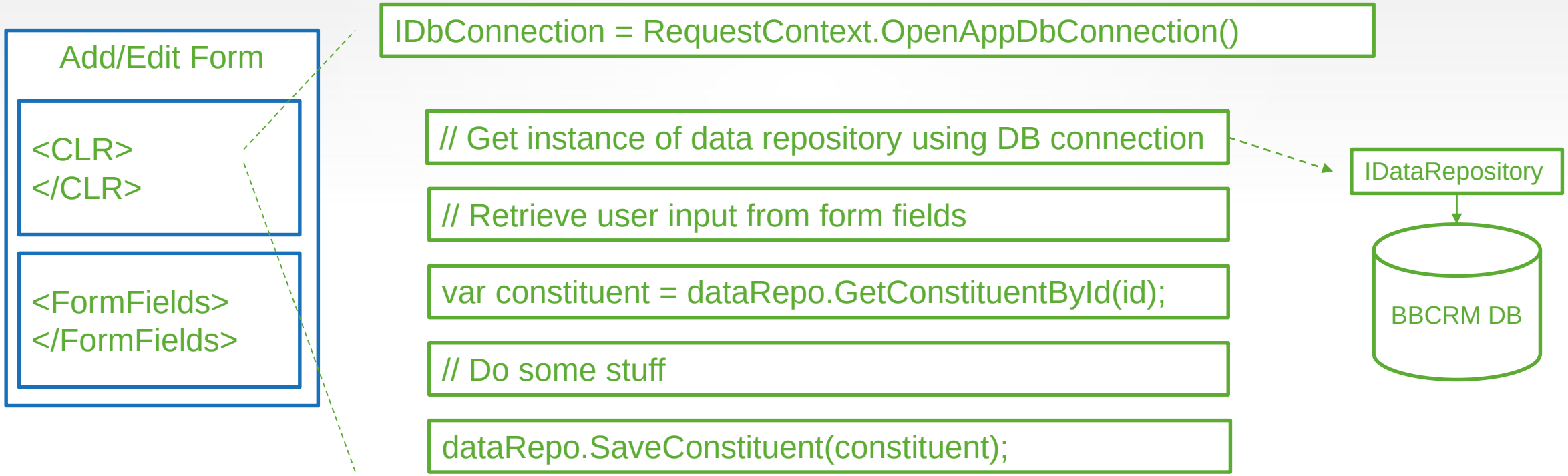
Dependency injection can automate composition of complex graph of objects

```
class DuplicateBioDataValidator : IRecordValidator

    // constructor
    public DuplicateBioDataValidator(IDataRepository dataRepo)
```

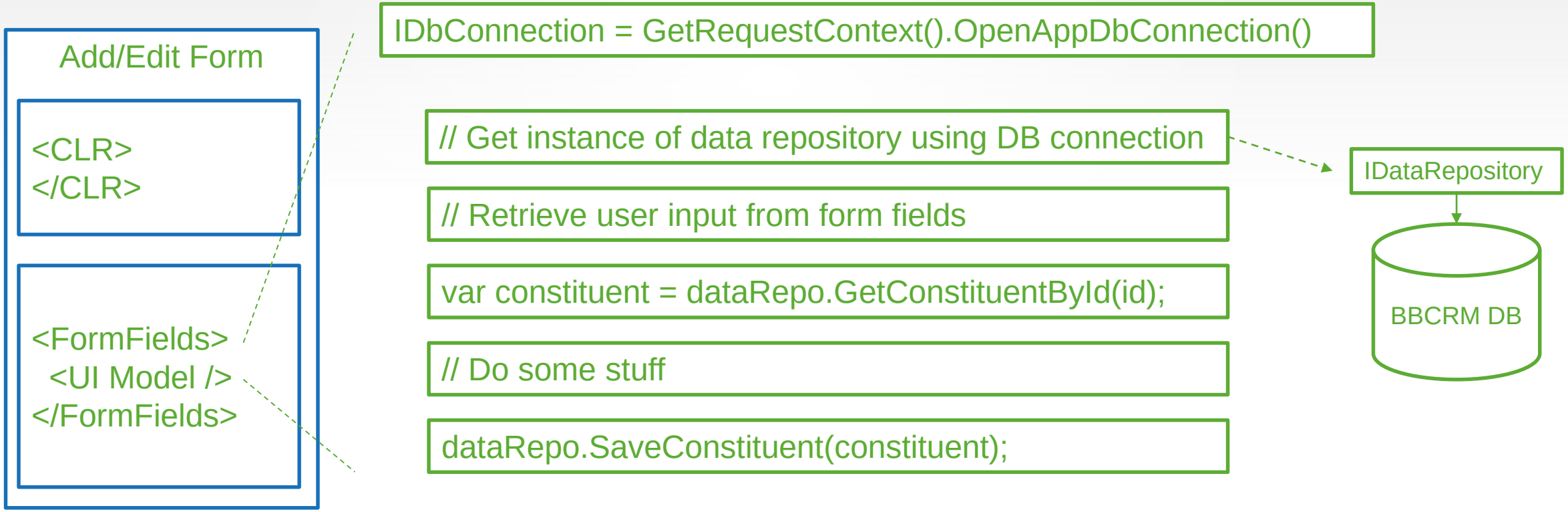
Managing Complex Customizations

- Plugging this in to the BBCRM SDK: Add/Edit Forms



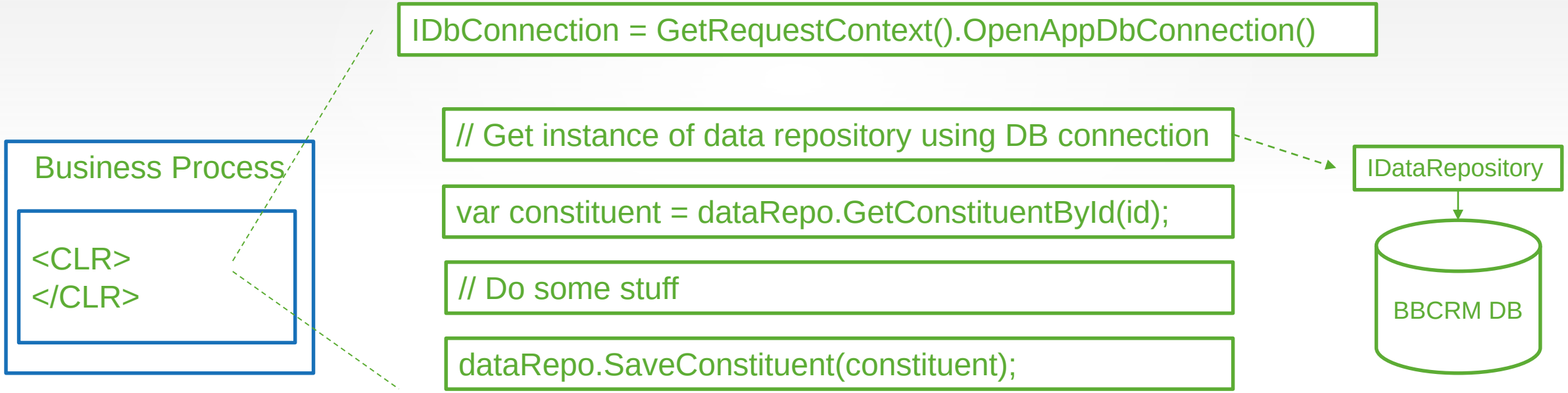
Managing Complex Customizations

- Plugging this in to the BBCRM SDK: UI Models



Managing Complex Customizations

- Plugging this in to the BBCRM SDK: Business Processes



Managing Complex Customizations

- Why? Original goals:
- Testable
  - DEBUGGER SUPPORT!!
  - Minimal dependencies
    - Mock data persistence interface, where necessary
  - Small, tightly focused tests
- Reusable
  - Execute at various points (on import, on change, on commit)
- Maintainable
  - Small units of logic
  - Low coupling

Managing Complex Customizations

- In practice:
  - 700+ unit tests
  - High degree of flexibility
  - Low incidence of BBCRM upgrade/patch breakage