Boundaries

Display → Business Logic → Data

Presentation / Display Layer

- Displaying data to the user, receiving input, passing input on to the application
- What logic goes in this layer?
  - Display logic
  - Formatting logic
  - User input validation (with a caveat, try to use the business layer if possible)
- What logic does not go in this layer?
  - Deciding what data to display based on contents of the data
  - Saving / loading data in the data layer
  - Altering data
- Goal:
  - Try to have your presentation layer extremely simple / dumb
  - By keeping the layer dumb, you can swap it for any technology

Business Logic / Rules Layer

- Part of the program that encodes business rules that determine data creation and alteration, the connection between data display and data persistence
  - How business objects interact with each other
  - Rules about data (how it can be altered, valid values, valid operations)
  - How data is organized (ownership, hierarchy)
- What goes in this layer?
  - Anything not concerned with display or persistence of data
  - Logic related to the reason your application even exists (what problem does your application solve? The solution is in this layer)
- What does not go in this layer?
  - Decisions about HOW to store data (compressed, storage technology, etc…)
  - Decisions about WHERE to store data
  - Altering data to affect how it is displayed (adding display logic to it)

Data Layer / Persistent Storage

- Provides persistent storage of some kind (RDBMS, NoSQL, file system, etc.)
  - Saving data
  - Loading data
  - Alteration of data for storage (serialization, compression, etc…)
- What goes in this layer?
  - Data, the more raw the better
  - Data separated into its smallest elements
  - Ideally in third normal form
- What doesn’t go in this layer?
  - Not data
  - Application logic
  - Display information (data wrapped in display info)
  - Rule information stored with data

Is This A Good Idea? (Display Layer)

```
<html>
    <body>
    <h1>Student Record</h1>
    <div>ID: <%= student.getId()%></div>
    <div>First Name: <%= student.getFirstName()%></div>
    <div>Last Name: <%= student.getLastName()%></div>
    <div>Is Foreign Student: <%= student.getCountry() != "Canada"></div>
    </body>
</html>
```

Is This A Good Idea? (Business Layer)

```
Sub UpdateCEUsersSaveInfo(strCEUserName)
  Dim rs, sql, bSaveAddress
  sql = "SELECT * " &
        "FROM CEUsers C " &
        "INNER JOIN CEGroups CG ON C.SubscriberID = CG.SubscriberID " &
        "WHERE C.CEUserName = '" & Replace(strCEUserName, "'", "''") & "'"
  Set rs = objConn.Execute(sql)
  If Not rs.EOF Then
    bSaveAddress = True
    sql = "UPDATE CEUsers SET SaveName = 1"
    Do While Not rs.EOF
      If rs("MembersMustPay") Then
        bSaveAddress = False
        Exit Do
      End If
      rs.MoveNext
    Loop
    If bSaveAddress Then
      sql = sql & ", SaveAddress = 1"
    End If
    sql = sql & " WHERE CEUserName = '" & Replace(strCEUserName, "'", "''") & "'"
    Set rs = Nothing
    objConn.Execute(sql)
  End Sub
```

Is This A Good Idea? (Data Layer)

```
CREATE TABLE University (
    Name VARCHAR(MAX),
    DisplayHeader VARCHAR(MAX),
    Address VARCHAR(MAX),
    ContactInfo VARCHAR(MAX)
)
```

```
<html>
  <h1><%= university.GetName() %></h1>
  <%= university.GetDisplayHeader() %>
  <h2>Address:</h2>
  <%= university.GetAddress() %>
  <h2>Contact Info:</h2>
  <%= university.GetContactInfo() %>
</html>
```

```
INSERT INTO University (Name, DisplayHeader, Address, ContactInfo)
VALUES (
  'Dalhousie',
  '<b>Dalhousie is great!</b>',
  '<br/><h3>1 Street</h3><br/><h3>Halifax, Nova Scotia</h3>',
  '<br/><h3>info@dal.ca</h3><br/>'
);
```

Why Not?

- Inflexible, cannot swap solutions in layers
- Maybe it works when you first write it, but what about when something changes?
  - You can't safely make changes in a layer, without worrying about breaking other layers
  - **Rigidity is a disease, it spreads and kills your business**
    - A major customer is considering renewing their subscription to your company's products, they don't like something about your application, but rigidity prevents you from changing it. You lose that customer.
    - A competitor implements something that changes the game, if you don't follow suit you go out of business.
    - A new law or regulation comes into effect, you must meet the regulation or suffer massive fines or be shut down
- Let's look at some examples

Rob’s Company - Major Customers

- SAML
  - One large customer requested integration between their learning management system and ours via SAML authentication (them authenticating the user, then passing a token to us asking us to accept their authentication). We couldn’t make this happen, customer decided to not renew their subscription
  - We’re constantly faced with a cost analysis: do this difficult task that will cost us a lot of money (time spent on wages) due to our system rigidity vs. lose a customer worth X dollars
- Walmart
  - Maybe 30% of our business
  - Currently in negotiations for renewal of their subscriptions
  - They want changes:
    - Want our sites to load faster
    - Want us to implement custom courses for them that don’t fit our current systems
    - Have threatened that they will not work with 3rd party vendors that use Amazon cloud services

Competition - Failure / Inability To Innovate

- Adapt or Die:
  - Like evolution, fundamental rule in business. You cannot stagnate, you must be at the front of the pack, not the back
- Blockbuster:
  - Should have been Netflix
  - Would it have been easy to go digital? Imagine the struggle trying to scale.
    - Netflix started by mailing DVDs around
    - Cable companies started streaming / on-demand
    - Too little too late from Blockbuster
- Sears:
  - Should have been Amazon
- RIM (Blackberry):
  - Should have been Apple / iPhone
  - Even today they still aren’t caught up? Why? Probably rigidity.

Regulations & Laws

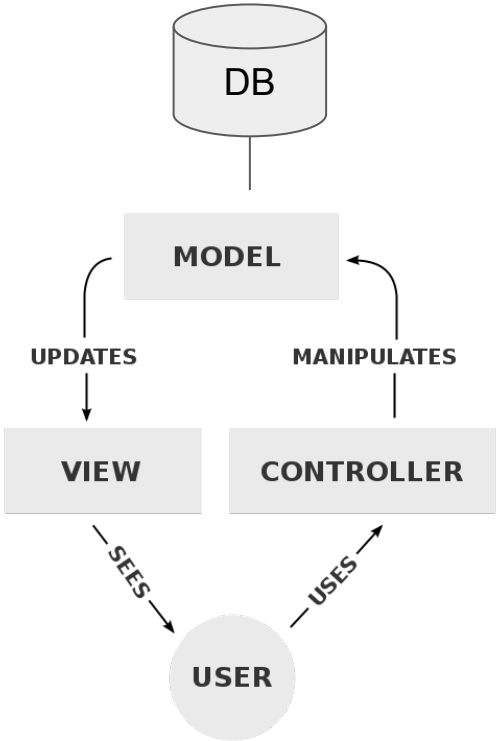
- Large cases against large companies:
  - Google vs. Getty Images
  - Microsoft vs. United States re: Internet Explorer bundling
- Uber:
  - Drivers not employees but contractors, if they lose that imagine how their systems will need to change to adjust to hours worked / SIN data, they will need to connect their systems that track driver income with tax / HR software
- American Disabilities Act:
  - Companies make large web apps, then they get a government contract (e.g. airline flying government employees around), they must then comply with ADA and retrofit
- General Data Protection Regulation (GDPR):
  - Anyone who does anything with Europe now has to retrofit all of their data systems to be able to delete all user data on request, imagine the hours across the internet that went into satisfying this new law

Solution? Separation of Concerns

- Achieve Cohesion
  - Bundle modules into their concerns using the cohesion principles
    - Presentation / Display
    - Business Logic / Rules
    - Data Layer / Persistent Storage
- Avoid Coupling
  - You will be faced with a choice: get your program working quickly vs. prevent coupling and keep your modules loose so that you can swap modules in and out
    - Choose the difficult path, what you're writing either has value or it doesn't, if it has no value why are you writing it? Go find an existing solution.
    - If it has value, write it for the long term to maximize re-use and flexibility. Otherwise don't write it!
- Most common solution: Three-Tier Architecture (presentation, business, data)

Model - View - Controller (MVC)

- Model: Domain logic, expresses the behaviour of the app in terms of the problem domain
  - Persists the model of data in the data layer if necessary
- View: Output representation of information
  - (e.g. HTML in web app)
- Controller: Accepts input converts it to commands for the model
- Language examples:
  - Java: Struts, Spring, etc…
  - Python: Legos
  - ASP.Net MVC
  - These systems give you the structure to start from and help you maintain cohesion and put your code in the right place



MVC Solution To Earlier Examples (Display Layer)

```
<html>
    <body>
        <h1>Student Record</h1>
        <!-- Notice how everything here is dumb and simple getters -->
        <div>ID: <%= student.getID()%></div>
        <div>First Name: <%= student.getID()%></div>
        <div>Last Name: <%= student.getID()%></div>
        <div>Student Type: <%= student.getIsForeignStudent()%></div>
    </body>
</html>

public class student {
    private String country;
    public String getIsForeignStudent() {
        if (country.equals("Canada")) { return "Resident"; }
        else {return "Foreign"; }
    }
}
```

MVC Solution To Earlier Examples (Business Layer)

```
Sub UpdateCEUsersSaveInfo(strCEUserName)
  Dim user = CEDataLayer.LoadUser(strCEUserName)
  user.SaveName = True
  If Not user.MembersMustPay Then
   user.SaveAddress = True
  End If
  CEDataLayer.UpdateUser(user)
End Sub
```

MVC Solution To Earlier Examples (Data Layer)

```
CREATE TABLE University(
    ID BIGINT,
    Name VARCHAR(MAX),
    Slogan VARCHAR(MAX)
)
CREATE TABLE ContactInfo(
    ID BIGINT,
    Street VARCHAR(MAX),
    City VARCHAR(MAX),
    Province VARCHAR(MAX),
    Email VARCHAR(MAX)
)
```

```
<html>
  <h1><%= university.GetName() %></h1>
  <br/><b><%= university.GetSlogan() %></b>
  <h2>Address:</h2>
  <br/><h3><%= university.GetStreet() %></h3>
  <br/>
  <h3>
    <%= university.GetCity()%>,
    <%= university.GetProvince()%></h3>
  <h2>Contact Info:</h2>
  <br/><h3><%= university.GetEmail() %></h3>
</html>
```

```
INSERT INTO University (Name, Slogan)
VALUES ('Dalhousie', 'Dalhousie is great!');

INSERT INTO ContactInfo (ID, Street, City, Province, Email)
VALUES ('1 Street', 'Halifax', 'Nova Scotia', 'info@dal.ca');
```

N-Tier

- Don’t need to restrict yourself to 3 layers, there are other architectures:
  - Service Layer:
    - A layer between other layers (presentation layer and business logic layer, data and business logic)
    - Allows for transmission of input / output across a service to the business logic running somewhere else (e.g. SSL, load balancers, printing, file systems, custom hardware)
  - Thin Clients:
    - Dumb terminals that provide only user input / presentation
  - Rich Clients:
    - Flash: Very powerful client-side logic
    - Video games: Games that allow mods / scripting add-ons (e.g. World Of Warcraft)

Where Should Heavy Duty Data Processing Go?

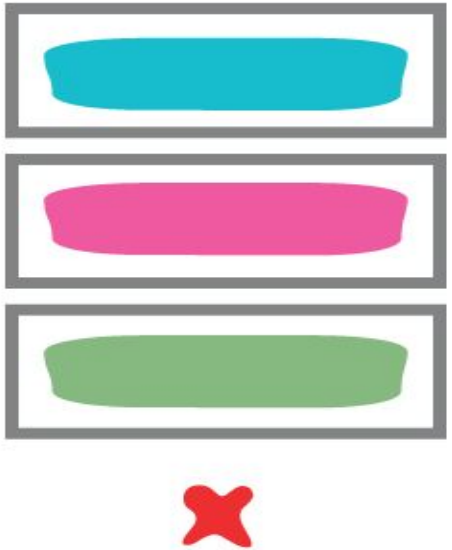
- Business logic in the DB: Good or Bad?
  - **Bad:**
    - Locks you into a specific RDBMS
    - Splits logic into multiple places
    - Harder to debug database logic
  - **Good:**
    - RDBMS are powerful tools capable of insanely powerful operations revolving around sets of data
    - Potentially more scalable (spin up more servers)
- Performance: Consider this last, focus on maintainable / cohesive code. Optimize when performance is for sure a problem
- Understandability: SQL is not the simplest language in the world, people do not think in terms of sets
- Testability: Possible, yet unlikely you will use test-driven development in SQL

Rob’s Advice - DB / Business Layer

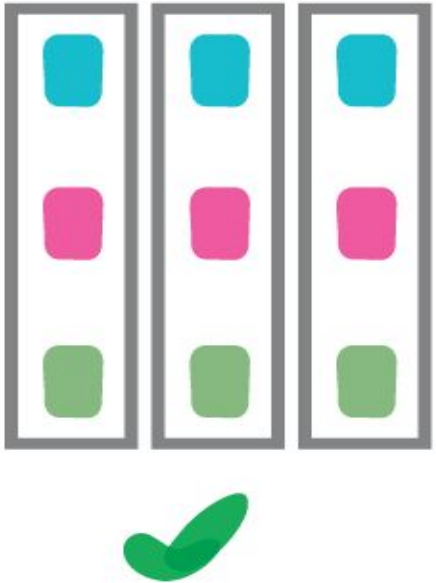
- Start with **no** logic in the DB:
  - Only exception is sorting, if you are sure your data will be sorted you should default it to its most common sorting arrangement on return from the DB.
- ZERO raw SQL in your business layer:
  - Use stored procedures / functions to load and save objects
  - Protects you from schema changes in your database
- When and if you have performance issues:
  - Use profiling tools to ensure that processing large volumes of data in memory is for sure the problem and not something else. Assumptions are the curse of our industry, be sure.
  - Optimize in stages. Find the largest set of data that gets worked on, move that to a stored procedure or function, re-analyze your performance, repeat.

Divide Layers, By Module

- Don’t create monolithic modules for each layer, still separate by smart boundaries (probably driven by business layer categories)
- Full-stack:
  - For each category the ability to store data, manipulate the date and present the data
  - Within the stack the ability to swap how any of those three is done



Don’t use layers as the top level modules in a complex application...



... instead make your top level modules be full-stack

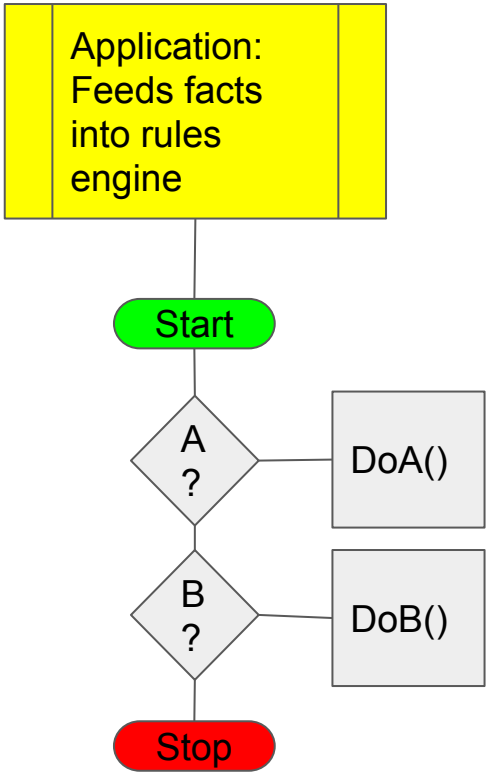
https://martinfowler.com/bliki/PresentationDomainDataLayering.html

Tools That Help You Construct Layers

Note: **None** of these are allowed in your projects. They would make it so you barely had to write any code, good in the real world, bad in a class about writing code.

Business Rules Engines (BRE)

- Allow anyone in your business to define the rules of the business
  - Technical skills not required, only understanding of the available inputs into conditions and the results or actions to take when conditions are met
  - Often have graphical editors to see the rules and debug / test
- Drools:
  - Rules: The “if” part of the rule
  - Facts: The conditional part of the rule
  - Activations: The “then” part of the rule
  - Session: Insert facts into a session until rules are triggered
  - Agenda: The logical place where activations are waiting to be fired, attached to rules



Object-Relational Mapping

- Turn your database into objects, have the database work of loading and storing those objects done for you
- ORM Tools:
  - Hibernate
  - NHibernate
  - Entity Framework
  - Etc...

```
String sql = "SELECT id, first_name, last_name,
phone, birth_date, sex FROM persons WHERE id =
10";
Result res = db.execSql(sql);
String name = res[0]["FIRST_NAME"];
```

vs

```
Person p = repository.GetPerson(10);
String name = p.getFirstName();
```

Content Management Systems

- Manages creation and modification of digital content
- Allows templatization of your user interface
- Allows non-developers to build the hierarchy and content of your webapp, leaving developers to do the work of adding new page types, or complicated business logic that drives content manipulation
- Examples:
  - Umbraco
  - Sitecore
  - Joomla
  - Drupal
  - Wordpress