What does “business logic” actually mean if not “all non-3rd party code”?

Asked 5 years, 2 months ago Active 4 years, 3 months ago Viewed 7k times

27

9

I've heard people talk about business logic a lot at work, and online, and I've read several questions on this site about it, but the term still doesn't make a lot of sense to me. For example, here are some (paraphrased) statements I often see:

- "The business logic is the part of your program that encodes the actual business rules." Most of the definitions I've read are circular ones like this.
- "Business logic is everything unique to your particular application." I don't see how this is different from "your particular application is nothing but business logic", unless we accidentally reinvented a bunch of wheels we could have used existing 3rd party software for. Hence the question title.
- "There should be a Business Logic Layer above your Data Access Layer and below your GUI Layer." In the code I write, the database accessors have to know what data they're supposed to be accessing, and the UI code has to know a lot about what it's displaying in order to display it correctly, and there's nothing to really do in between those two places other than passing blobs of data between client and server. So what's actually supposed to go into a Business Logic Layer?
- "Business logic should be separate from presentation logic." Most of the feature requests we get are to change the presentation logic for business reasons. If one of the business rules is to display US government bond prices in 32nds notation by default

Is it possible that I actually am on a team that only does business logic, and all the non-business logic is being done by other teams? Or is the whole concept of "business logic" as a separate entity only workable for certain applications or architectures?

To help make the answers concrete: Pretend you have to reimplement the Domino's Pizza app. What is the business logic, and what is the non-business logic of that app? And how would it be possible to put that pizza-ordering business logic in its own "layer" of code, without most of the pizza information bleeding into the data access and presentation layers?

Update: I've come to the conclusion that my team is probably doing 90% UI code and most--but not all--of what you'd call business logic comes from other teams or companies. Basically, our application is for *monitoring* financial data, and almost all of the features are ways for the user to customize what data they see and how they see it. There is no buying or selling going on (though we integrate a bit with other apps from our company that do that), and the actual data is supplied by loads of external sources. But we do allow users to do things like send copies of their "monitors" to other users, so the details of how we handle that probably qualify as business logic. There actually is a mobile app that currently talks to some of our backend code, and I know exactly what portion of our frontend code I would like it to share with our UI in an ideal world (basically the M in our quasi-MVC) so I'm guessing that's the BLL for us.

I'm accepting user61852's answer since it gave me a much more concrete understanding of what "business logic" does and doesn't refer to.

architecture business-logic business-rules

edited Jan 3 '15 at 13:10

asked Jan 2 '15 at 23:05

lxrec
25.8k 13 71 81

1 You sweep all the scaffolding, infrastructure, boilerplate, library code under the "wheel reinvention" rug, but that's actually a good chunk of code, and not all of it could reasonably be third party code. Maybe it's not unique to your product, but unique to your product and three competing products. Maybe you have weird requirements that rule out existing solutions. Maybe existing solutions don't cut it for you for technical reasons (say, don't meet performance goals -- this is a common reason to reinvent basic data structured in game development). – user7043 Jan 2 '15 at 23:31

For us the infrastructure, library code and scaffolding is mostly maintained by other teams or 3rd parties (though boilerplate is evenly spread everywhere), so maybe it is as simple as I'm on the team doing the UI/business logic. – lxrec Jan 3 '15 at 0:29

8 If you order more than $50 you get free cheese-encrusted breadsticks. – kdgregory Jan 3 '15 at 0:32

1 @raptortech97 He/she is saying that "if you order more than $50 you get free cheese-encrusted breadsticks" is business logic. – user253751 Jan 3 '15 at 4:48

label/textbox/widget to display whatever string the business logic (or more probably the model, but...) passed to it. – Joshua Drake Oct 23 '15 at 16:34

2 Answers

Active | Oldest | Votes

29

I'll give you some tips regarding CRUD applications, since I don't have much experience in games or graphically intensive apps:

- **Business logic usually involves rules the owner of the business has learned** or decided over years of operation, like for example: *"reject any new credit if the client hasn't yet finished paying the last one"*, or *"we don't sell breakfast past 11 am"*, or *"mondays and tuesdays, customers can buy two pizzas for the price of one"*.
- Of course the presentation layer must show a message indicating the availability of a discount, or the reason of a credit being rejected, but such layer is not deciding those things, it's only communicating to the user something that happened under the hood.
- **Business logic usually involves a workflow**, for example: *"this item must be aproved within 3 working days by some manager or put into a 'request for information' stage, if customer hasn't submitted the required documents, then the item is rejected"*.
- Presentation layer usually doesn't deal with that kind of workflow, it only reflects the states of the workflow.
- Also, data access layer is usually straightforward, because decisions have already being made by the business logic. This layer is affected when you decide to migrate your data from MS SQL Server to Oracle, for example
- It's true sometimes GUI does some validation to avoid calls to the server, but that is something that should be done judiciously or you could have a lot of business logic in that layer.
- Much of your confusion may have arisen from the fact that in your application there's no separation of concerns and effectively you have too much business logic in the presentation layer. **The fact that you (wrongly) have business logic in your application layer or data access layer doesn't change the fact that it's business logic whatsoever.**
- **Things like displaying distances in the metric system instead of miles/yards/feet is not presentation logic, it's business logic**. The business layer has to return data in the required units and tell the presentation layer what units it's handling for it to show the appropiate labels, but it's definitely a business logic concern.
- **Business logic shouldn't be affected by** the fact that you are using Oracle now instead of Postgres, or by the fact the company changed it's logo or style sheet.
- **Business rules exist whether or not you automate it** by writing an app. They can be enforced even when the business uses a low tech solution like pen and paper.
- **If you have a mobile version of your desktop app, or a web version**, each one of those versions have a *different presentation*

edited Dec 22 '15 at 19:15

answered Jan 3 '15 at 1:08

Tulains Córdova

36.2k 12 85 143

5

It sounds like most of your work may be in the UI layer. Changing the display format for business reasons, does not imply any business logic. The change is a change to the view logic.

Being able to change the format implies some business logic possibly involving persistence of the preference.

Persisting the format to a cookie, could also be implemented in the view layer.

However, this could be implemented in a MVC manner. (Some models implement the view as its own MVC capable of dealing with preferences.)

- The user's preference could be stored by the model (database/cookie).
- The controller would react to format requests by changing the user's preference in the model.
- The view would adjust to the user's preference. The preference can be requested from the model, or provided by the controller.

Making an educated guess about your application.

- There is a model containing available bonds, and pricing data for them.
- There is a view allowing the user to see various things they can do: search for bonds, display prices, comparing yields, taking orders, and which displays the results returned by the business model in response to the request.
- The business logic handles things like:
  - Performing a search and providing for the view to display.
  - Finding prices for a bond and providing for the view to display.
  - Comparing yields and providing data for the view to display.
  - Processing orders and storing them in the model.

If this is done right, it should be possible to provide multiple view components without changing the model or business logic. For example, if the current design is a web site, a new view server for an iPhone or Android application would use the existing model and business logic. These may introduce new business functionality to provide push notifications when an order is fulfilled, which may require changes to multiple layers.

- The data layer represented by the model tends to be relatively stable.
- The business layer is where business decisions are applied: Will/can the request be fulfilled? Has all the required data been obtained? Is the user authorized? Are there any red flags on the transaction?
- The view layer tends to be less stable, and there may be more than one. This is where the look and feel of the application resides. It is possible to completely re-skin an application, without changing the other layers.

answered Jan 3 '15 at 0:23

BillThor
5,879 14 17