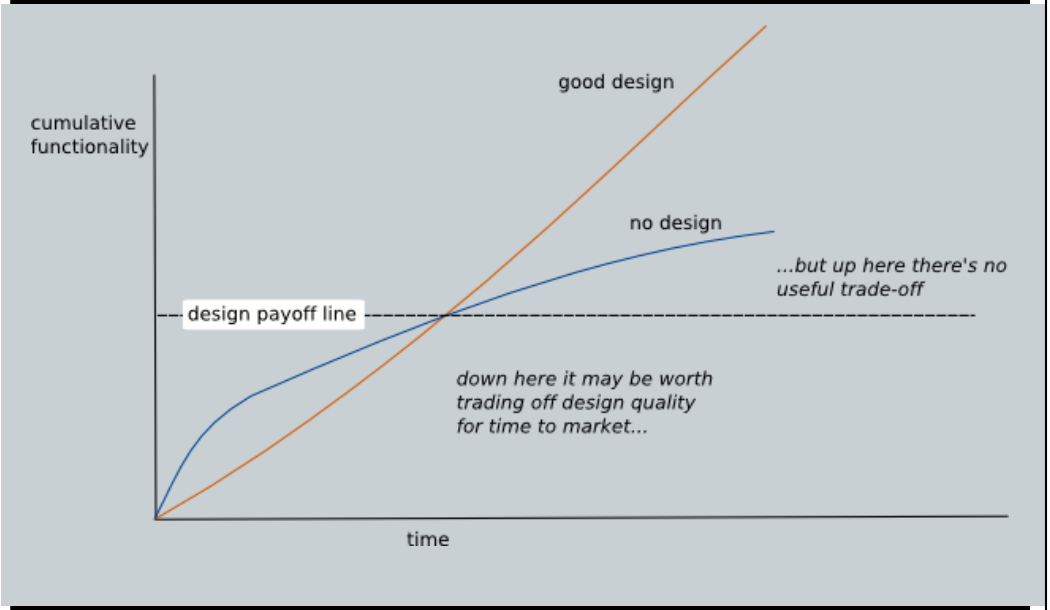
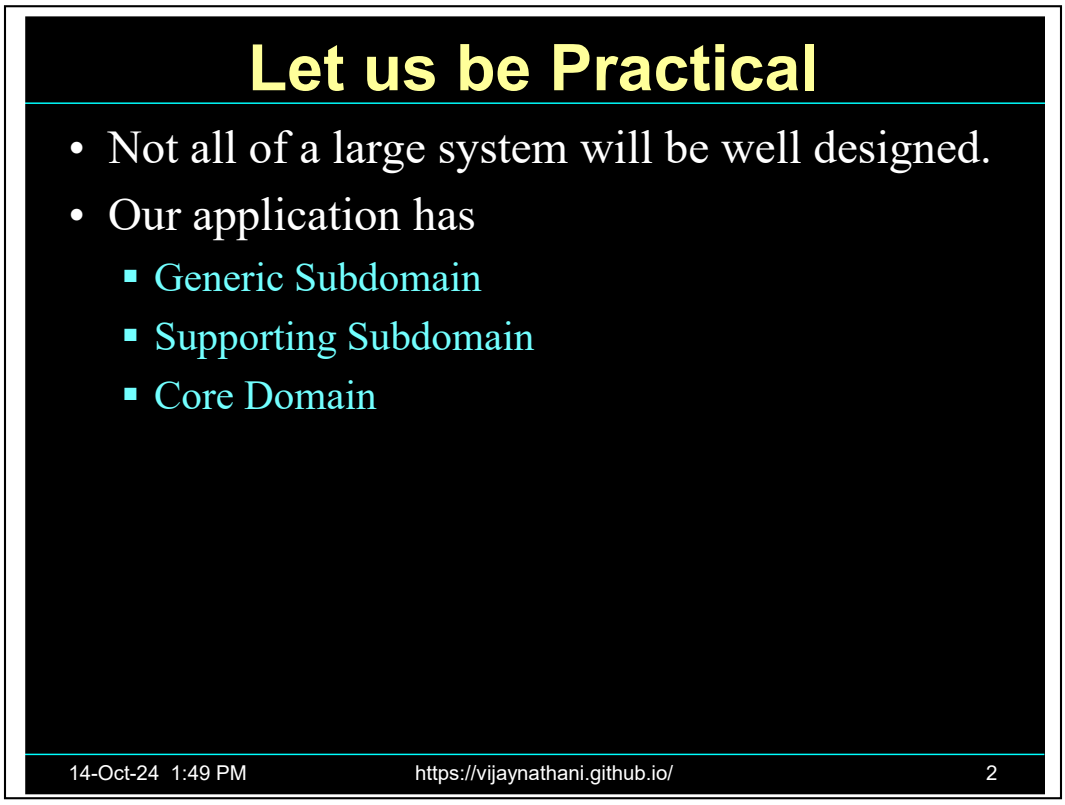
Is Good Design worth it?





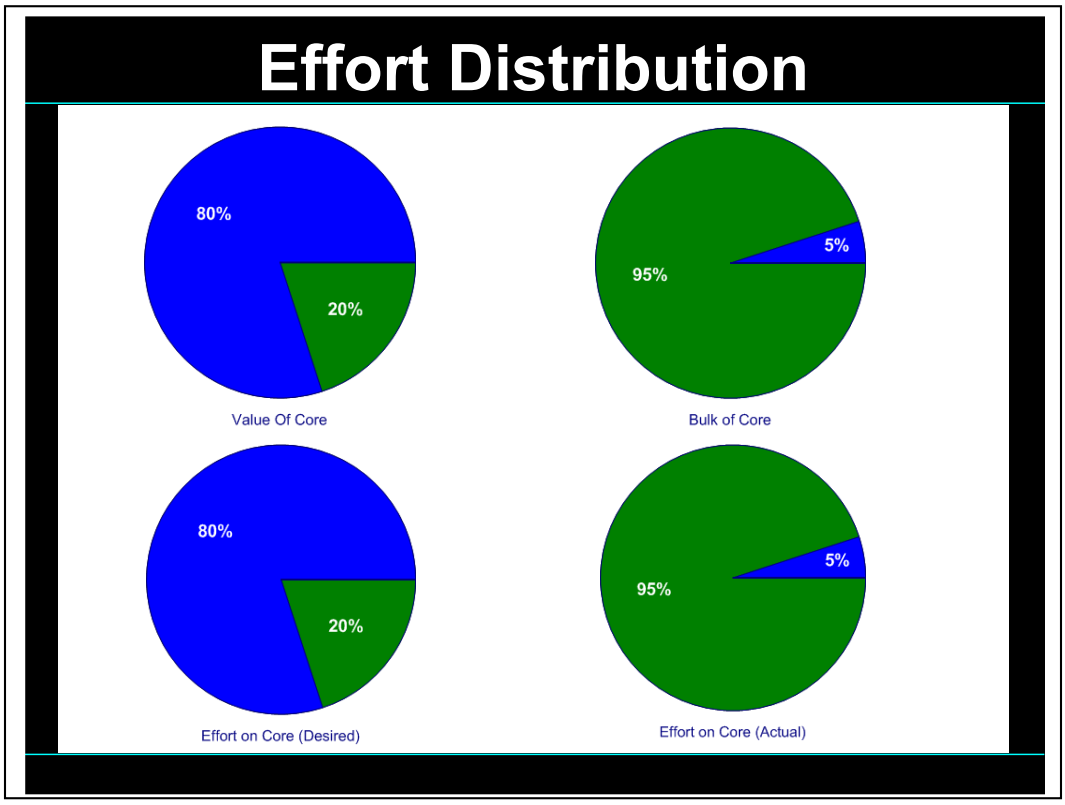
Core domain:

The core domain is the one with business value, the one that makes your business unique. Apply your best talent to the core domain. Spend a lot of time on it and make it as good as you can.

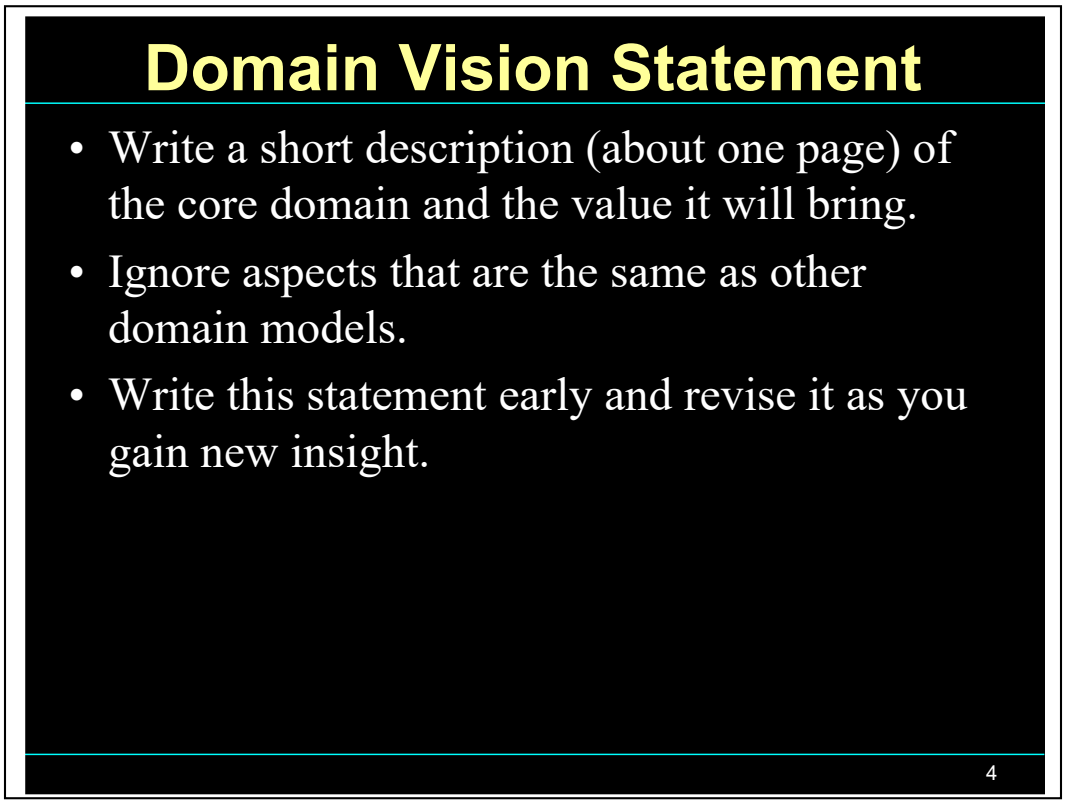
Generic Subdomain: Identify cohesive subdomains that are not the motivation for your project. Factor out generic models of these subdomains and place them in separate modules. These domains are not as important as the core domains. Don't assign your best developers to them. Consider off-the-shelf solutions or a published model. E.g. calendar, Invoicing, Accounting

Supporting Subdomain: Related to our domain but not worth spending millions of dollars every year. E.g. Advts on bank statements.

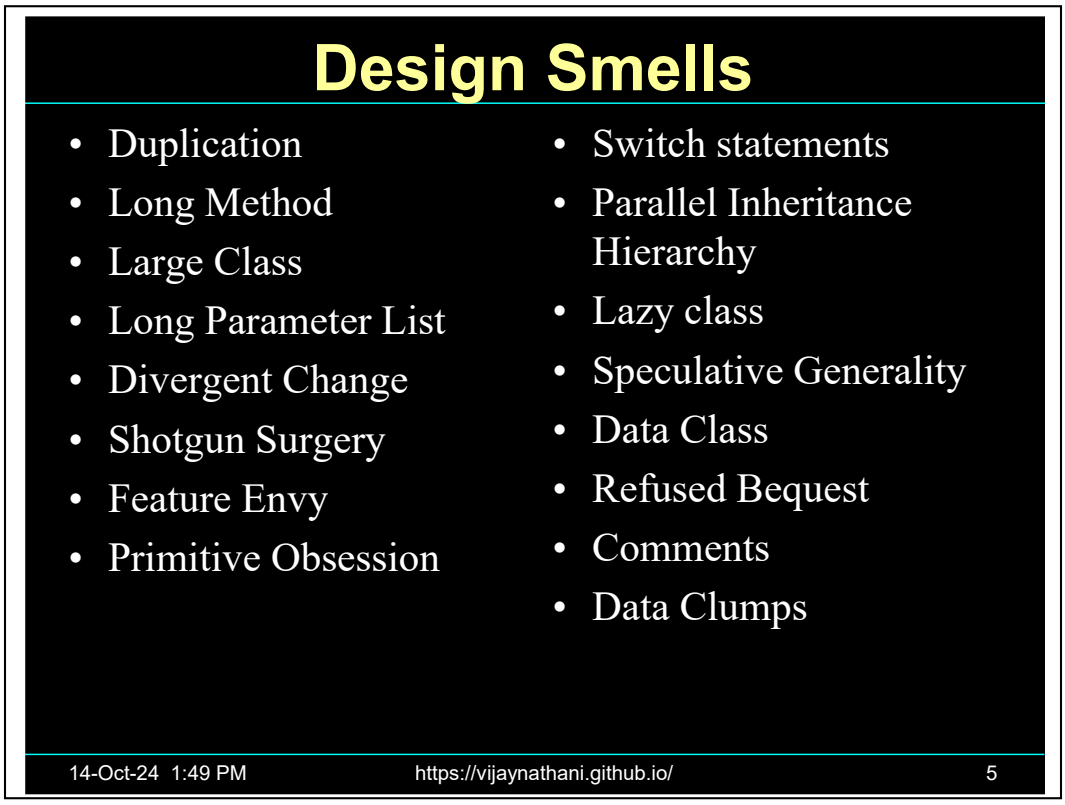
User ratings on Amazon and Ebay. On Amazon, this feature is supporting domain because customers usually buy the book even if others have given it low rating and they really want it. For ebay it is core domain because people will hesitate to make a deal with a person with low rating.



From Domain Driven Design.



Highlighted Core: Write a very brief document (3-7 pages) that describes the core domain and the primary interactions between core elements.



Divergent change occurs when one class is commonly changed in different ways for different reasons. If you look at a class and say, "Well, I will have to change these three methods every time I get a new database; I have to change these four methods every time there is a new financial instrument," you likely have a situation in which two objects are better than one. That way each object is changed only as a result of one kind of change.

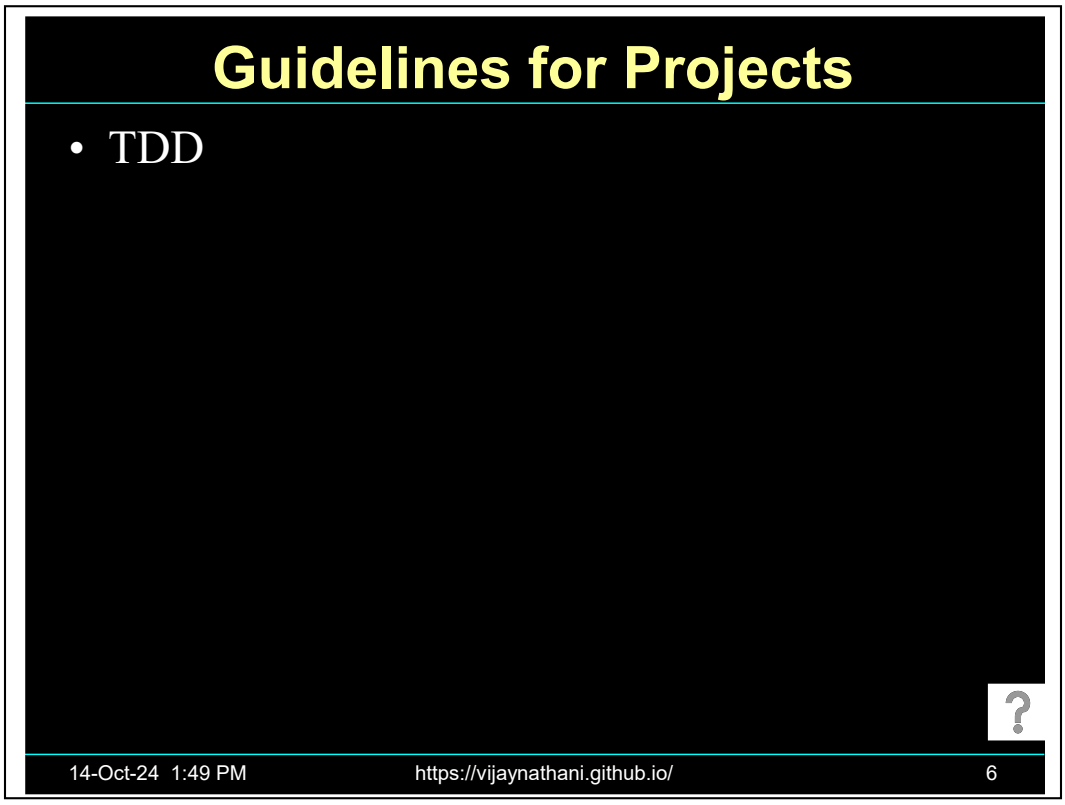
Shotgun surgery is similar to divergent change but is the opposite. You whiff this when every time you make a kind of change, you have to make a lot of little changes to a lot of different classes. When the changes are all over the place, they are hard to find, and it's easy to miss an important change.

Feature Envy: The whole point of objects is that they are a technique to package data with the processes used on that data. A classic smell is a method that seems more interested in a class other than the one it actually is in.

Data Clumps:Data items tend to be like children; they enjoy hanging around in groups together. Often you'll see the same three or four data items together in lots of places: fields in a couple of classes, parameters in many method signatures. Bunches of data that hang around together really ought to be made into their own object.

Data classes are like children. They are okay as a starting point, but to participate as a grownup object, they need to take some responsibility.

Refused Bequest: Subclasses get to inherit the methods and data of their parents. But what if they don't want or need what they are given? They are given all these great gifts and pick just a few to play with.



QCardChips

QCoffee

Qinvestor

Q95 – Hangman (only in Java, C++ and PHP)

Q92 – Mastermind(only in Java, C++ and PHP). Solution for some languages present.

Summary

- Keep it DRY, shy and tell the other guy.
- Prefer composition over inheritance
- Self-documenting code
- Not all parts of a large system will be well-designed.

No Golden Bullet.