City Building: Iteration 1

Object transformation

> There are two ways to write error-free programs; only the third one works.
>
> *Alan Perlis*

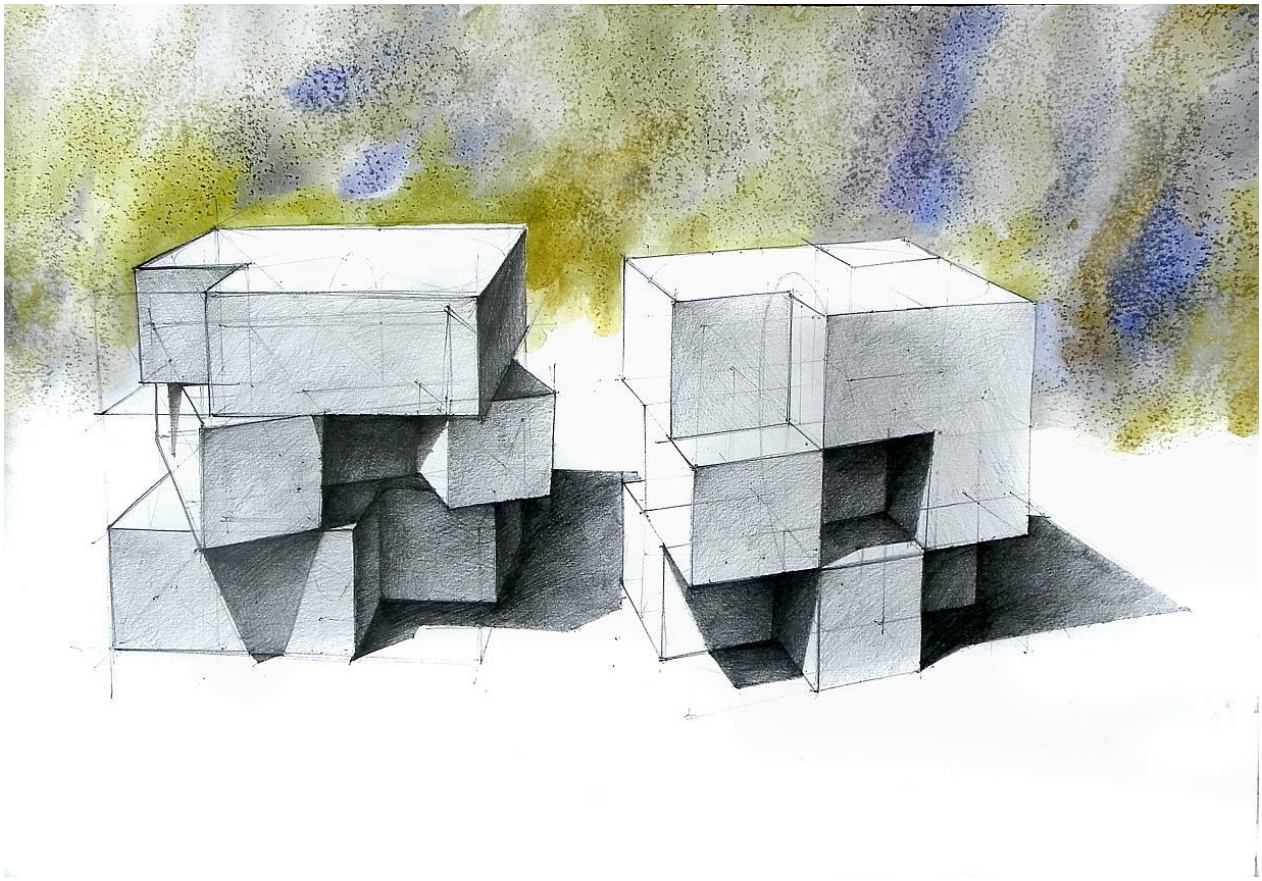

Figure 1: Every program has two purposes: The one for which it was written and another for which it wasn't.

The City Council has instilled a set of rules that would govern the conceptual and structural integrity of the city. This growing book of regulations is meant to control the greediness of our fellow city builders. Therefore, next time when you're trying to build something, our newly developed system will test the artifact before it is constructed. This system applies the aforementioned set of rules (like a spam filter) on the product and if it fails, it will tell you exactly which regulations you've broken. Moreover, this magic system is capable of outputting suggestions on how to modify the object so that it conforms with the law. As a bonus, it also notifies you whenever the City Council changes anything in their rule book.

The fun stuff

Remember that logging subsystem that you've previously built? Well, we've supercharged it. Every single recorded action that has a side-effect can be replayed. This means that you can go back and forth in history to monitor the progression of the city. Obviously, you can also form collections of actions, which really come in handy when you want to replicate some proven design decisions in some other parts of the city.

Oh, by the way, we forgot to tell you that this *magic* system doesn't exist yet. So go on and use your magic hands to make it a reality. We're giving you the gift of employment and we like our worker bees to be productive. *Buzz, buzz.*

Requirements and food for thought

1. Use the MVC(or MVVM if you're using WPF)[1] architectural pattern to re-implement the system. Thankfully, the lessons you've learned from the first assignment are of great value and you would be wise to embed them into this new system from the very start. You're now free to use any ORM you like, coupled with any type of database engine that you like.

2. Detailed documentation should be generated from the code. Some people firmly believe otherwise. We call those people *wrong*. You should write additional documentation to highlight important concepts. Think of these as comprising a first step for the reader before he or she goes into the code-based details. You can insert sketch-like diagrams to highlight the most important parts of the system. Producing detailed diagrams of the entire system is not required unless doing so is a requirement in itself. However, a newly appointed engineer may need to understand the purpose of each major component. Thus, each package should contain a class diagram. The class diagram should be supported by a handful of interaction diagrams that show the most important interactions in the system. Again, selectivity is important here.

3. One of the most important things to document is the design alternatives you didn't take and why you didn't do them. That's often the most forgotten but most useful piece of external documentation you can provide[2].

4. Use cases are an important part of the documentation. Contrary to popular belief, these are meant to be described in text, not in a graphical format.[3]

5. Here's a random file from the Flask project. Notice the *docstrings* under each function. This is how they generate the actual detailed documentation of the project. The high-

[1] https://martinfowler.com/eaaDev/uiArchs.html
[2] UML Distilled by Martin Fowler
[3] Please refer to Alistair Cockburn's Writing Effective Use Cases for a good primer on this.

level documentation in the **docs/** directory is meant to give you a bird's-eye view of the entire system. Combined, these form a cohesive design doc that describes the entire project, as seen here. Take a look at Literate programming for a good primer on what good documentation looks like.

6. Try to design your system in terms of *what* happens, rather than *how* something happens. In other words, design each workflow as a series of composable data transformation steps[4].

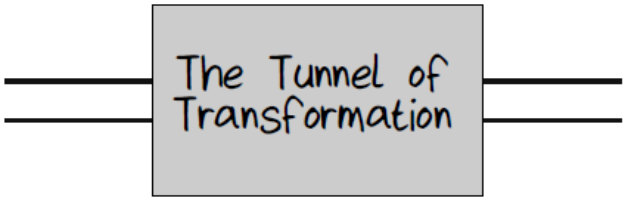

Figure 2: A transformation computation is a highly cohesive component, that does one thing, and does it well.

This allows you to think at a higher level of abstraction, which empowers you to see farther into the design space. In this project, what if you had a computation that takes an item as input and returns a transformed item as output, along with its metadata? What is the level of reuse that you could get out of such division of labour?

[4] Data-driven programming