City Building Education

The one to throw away

Systems program building is an entropy-decreasing process, hence inherently metastable. Program maintenance is an entropy-increasing process, and even its most skillful execution only delays the subsidence of the system into unfixable obsolescence.

Fred Brooks in "The Mythical Man-Month"

Figure 1: The *City Building* project series is inspired by Doreen Nelson's groundbreaking work of the same name.

Welcome to the future, *Star Trek* is NOW! A simple *create* function has immediate effects on the physical domain.[1] This allows for a *1-to-1* relationship between the digital world and the real world. Programming hasn't changed a bit since its inception so that's a shame, but this feeling of sadness is overwhelmed by the joys of job security. Speaking of your job, your role is to manage the information model of the city.

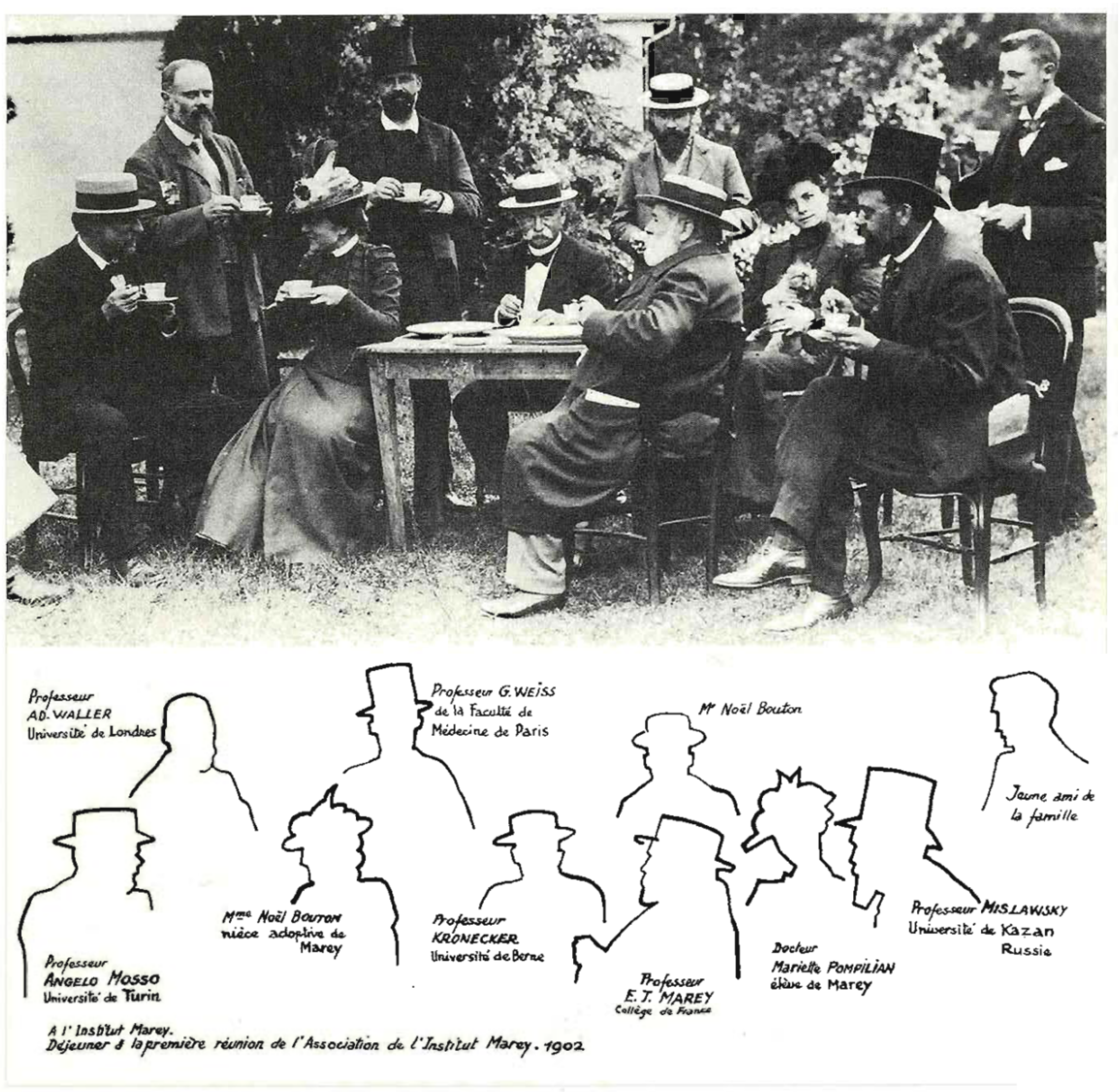

Professeur AD. WALLER Université de Londres
Professeur G. WEISS de la Faculté de Médecine de Paris
Mr Noël Bouton
Jeune ami de la famille
Mme Noël Bouton nièce adoptive de Marey
Professeur KRONECKER Université de Berne
Professeur MISLAWSKY Université de Kazan Russie
Professeur ANGELO MOSSO Université de Turin
Professeur E.T. MAREY Collège de France
Docteur Marielle POMPILIAN élève de Marey
A l'Institut Marey.
Déjeuner à la première réunion de l'Association de l'Institut Marey. 1902

Figure 2: This picture and mapping of a 1902 reunion of colleagues of E.J. Marey, the great scientific photographer, provides names and affiliations linked to hat-head outlines that run parallel with the image. The diagram allows label-captions of greater detail than labels placed on the picture itself. (*From "Beautiful Evidence" by Edward Tufte, page 42.*)

The natural surroundings of the city play an important role in its development. You'll need

[1]For the sake of this project series, every *CRUD* operation on the city is zero-cost.

a layered representation of the world, such as:

- Land cover
- Hydro layer (e.g: rivers, sea, ocean)
- Places layer
- Amenity layer
- etc.

We won't concern ourselves with the road network in this project series (unless you want to) but it would constitute yet another layer in the above-mentioned list. Each place has a type[2] and a geometry which directly impact the pollution level of the city. Moreover, geographic proximity plays a key role in the efficiency of a place. The *id* of a place is associated with a multitude of properties, such as its address, public photos, capacity, availability, contact and an icon, should you choose to polish it further. Each place modifies the rating of its encapsulating district, which is yet another score that you should keep track of. A hospital or a park may increase the prestige of the district, while a factory may have a negative impact on this metric. Obviously, you cannot build two or more places in a single physical location. As a direct result of that, you have to implement a logging system that records every single action that has been performed on the city.

This project series is centered around envisioning information,[3] which means that you'll need a visualizer to quickly debug the problems in the system. This tool will also reduce the feedback time of the project, hence increasing your development speed. At a minimum, you'll need a layered pixel map to get a rudimentary understanding of your program's behaviour. On top of that, you'll want some metrics for each district, which means that you'll have to develop a system that computes the scores for each zone, and a compound score for the city itself. Don't forget that these reports are directly impacted by the *create*, *update* and *delete* actions.

You crack you knuckles, prepare a tasty dose of dehydrated water and get to work.

Requirements and food for thought

1. Extract all the requirements and sort them by the MoSCoW method
2. Create a Kanban board and place all the items there (you can use Trello for that, or GitHub Issues). This board is a living document that indicates your progress, which is going to haunt you for a couple of months. The next few iterations will add new

[2]You can get some inspiration from here. The list is extensive, but you'll add more supported types as the project develops.

[3]Edward Tufte's quadrilogy is a mind-altering experience that will greatly change the way you think about the world.

features and you may have to reshuffle the requirements. Implement them in your favourite language.

3. Now that you've matured a bit as a software engineer, you know that database driven design isn't the way to go, as it is highly rigid. You like flexibility, so persistence ignorance is the only kind of positive ignorance you accept. You study the domain of the problem carefully, get the pen and paper, and start modeling the domain with highly cohesive bounded contexts. Bad decisions taken at this stage are going to be very costly later on. Think very carefully about the data structure formats associated with the aforementioned entities.

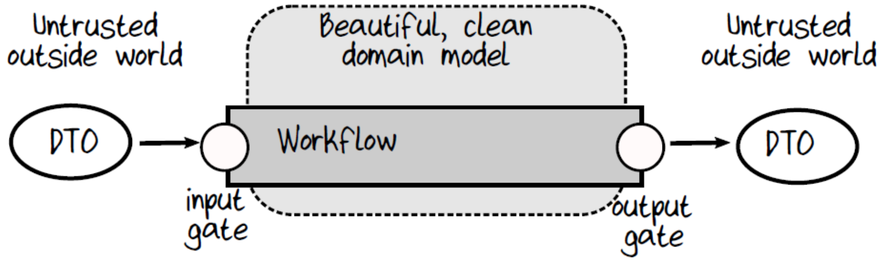


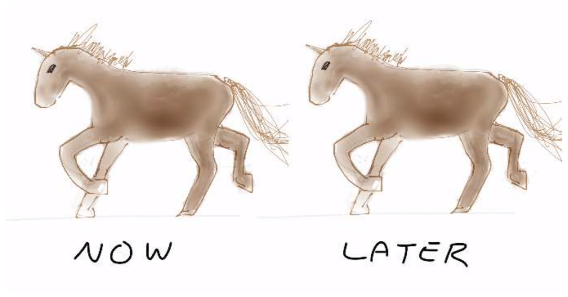
Figure 3: The perimeter of a bounded context acts as a "trust boundary." Anything inside the bounded context will be trusted and valid, while anything outside the bounded context will be untrusted and might be invalid. At the input gate, we will always validate the input to make sure that it conforms to the constraints of the domain model. The job of the output gate is different. Its job is to ensure that private information doesn't leak out of the bounded context, both to avoid accidental coupling between contexts, and for security reasons.

4. You are from the SQL school of thought, so you hate ORMs. Therefore, for this assignment, you get to choose two patterns (one from each):[4]

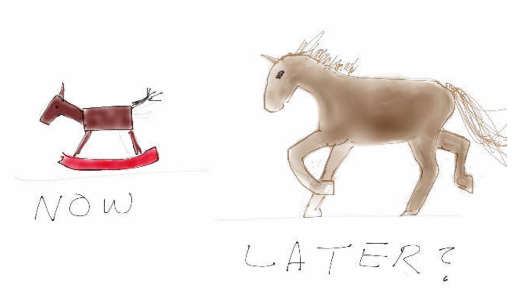
   (a) One Domain Layer Pattern
   - Transaction Script
   - Domain Model
   - Table Module
   - Active Record

   (b) One Data Source Pattern
   - Row Data Gateway
   - Table Data Gateway
   - Data Mapper

[4]You can find them all in Martin Fowler's book: Patterns of Enterprise Application Architectures



(a) Value is what we want.



(b) Value starts when we ship the software.

Figure 4: What if we shipped some valuable part sooner than the rest?

No point in over-complicating it, so you go with the tried-and-true Layers pattern. One unwritten rule of software development is that illegal states should be unrepresentable (easier said than done, however, some languages make it easier). You have proven yourself to be a good developer, so everybody is expecting to witness a quality product (that is thoroughly tested from the development and business sides). This is the prototype phase, so no need to focus on polishing the product at this point. Spin up an RDBMS, ignore the pretty aspects of the UI and implement a working system first. Beauty comes later.

5. A software architecture consists of four levels, if we are to follow Simon Brown's "C4" approach:[5]

   - The *system context* is the top level representing the entire system.
   - The system context comprises a number of *containers* which are deployable units such as a website, a web service, a database etc.
   - Each container in turn comprises a number of *components*, which are the major structural building blocks in code.
   - Finally, each component comprises a number of *classes* (or in functional architecture, *modules*) that contain a set of low-level methods or functions. One of the goals of a good architecture is to define the various boundaries between containers, components and modules, such that when new requirements arise, as they will, the "cost of change" is minimized.[6]

[5] http://static.codingthearchitecture.com/c4.pdf
[6] Passage shamelessly stolen from Scott Wlaschin's excellent book: Domain Modeling made Functional