YAWN — A TEX-enabled workflow for project estimation

Pavneet Arora

Abstract

A framework for using TEX and its variants — the term TEX is used generically in this article — in a project estimation workflow is discussed. While the emphasis here is less on the final output and more on issues related to the upstream processing, the framework itself does not place limits on how TEX might be used to create more beautiful output.

1 Introduction

Part of TEX's enduring appeal is its ability to be molded into a workflow. As the available tools have evolved — both in their expression (e.g., languages such as Ruby, Python, Perl, and Lua), and in their representation (e.g., XML) — the decoupling of the typesetting engine from the rest of the toolset has enabled it to adapt and stay current.

Often, these workflows relate to publishing and the emphasis is on the form of the final output given a marked up input. TEX's programming capabilities further foster integrating workflow solutions tightly to the document.

The problems that I deal with in my work, however, typically have less to do with the final output and more to do with upstream processing. Chief among these is the issue of project estimation.

The challenge with project estimation is that its evaluation is not a straightforward derivation. It requires exploring different solutions, adjusting costing parameters iteratively, and, yes, even judgement based on past experience to come up with a reasoned, if not always reasonable, attempt at an estimate. In essence, we seek the capability to run a set of guided "what-if" scenarios.

In the end, though, one does need to represent this estimate in a form that is meaningful to both the supplier whose very viability relies on it, and the customer who will use it to help decide to whom the project will be awarded. So the desired outcome is still a well-presented document. It is the steps leading up to it, though, that are the focus of this article.

2 A specific example to illustrate the generic problem

To help illustrate the nature of the problem, let's begin with a specific estimation problem: develop a cost estimate for a lighting control system: one that covers both hardware and labour, and encompasses design, cabling, control componentry, installation, etc. I will explain this in some detail so that the knottiness of the issues might be exposed. As specific as this example is, however, the estimation problem is a generic one and many aspects are shared across domains.

Everyone is familiar with light switches and dimmers in their own homes, so that is a good place to start. Architectural designs show these on their drawings connected to light fixtures, e.g., pot lights, pendants, surface mount lights, under-cabinet lights, etc. Imagine if to this mix we add:

- scene based keypads
- control processors
- power modules
- contact closure modules
- occupancy and vacancy sensors
- RS-232 interfaces to other equipment

Clearly, the complexity of the design grows immensely. From an estimation perspective, though, even this complexity is tractable, anchored as it is to physical aspects of the design. So far the engineering is contained.

The complexity runs quickly ahead, though, when we have to confront the haze of the gestation period during which projects are formed, and when details are decidedly lacking. During this time, it is easy to become overwhelmed with the fractured shards of the project definition. Amongst these are:

- Varying granularity of the known scope. In the case of our lighting control system, for instance, we may not know what types of light fixtures are to be used, nor a breakdown on a room-by-room basis. But we might have a rough count of the total number of light loads whose detailed decomposition will evolve during the design. To use a software development analogy, we are talking here about stubs for routines which will be fleshed out later.
- Blended designs utilizing existing and new components. We need only to include the new components in the estimate, but need to ensure fit and compatibility with existing components.
- Staged implementations requiring that the design anticipate, at the onset, the expected capacity of the overall design. The design needs to be comprehensive, but the estimate need concern itself with only the most immediate phase of the project.
- Interdependencies between components, i.e., a selected component requires a host of other components in order to function. As an example, a

wireless keypad or switch would require the presences of a wireless network to communicate with the main processor. Otherwise, its inclusion in a materials list is not meaningful.

- Tiered product solutions. Often, component manufacturers will have tiered groups of solutions with overlapping applications to a specific problem. From an estimating perspective it is good to run these alternative solutions against a common problem specification to see how the solution differs in cost and capability.

3 Typical solutions

Keep in mind, though, that the requirement to produce a budget does not wait for a full and final specification. We must produce budget estimates with varying degrees of confidence during the entire design period.

There are two typical approaches to the estimation problem, especially when they relate to a specific vendor's equipment: either use their design software to create an equipment list, or their estimation software to create broad-brushed budgets. Neither approach is satisfactory.

The design software has several shortcomings:

- It is quite laborious. Often, it can take days to create a design.
- With the absence of detailed specifications during the early stages of the project, it is difficult to capture this uncertainty in the design. The design software assumes a final or near final specification.
- The software, oriented as it is towards a design, is specific to a single solution set. So from the same company we might have multiple pieces of software, one per tiered solution, into which the design has to be entered repeatedly.
- There is no way to mark components as existing even when the design is to be blended with existing parts.
- There is no way to capture items outside of the vendor's portfolio, e.g., wire and cable costs cannot be captured inside the design software. This leads to manual tracking of material and labour costs.

The estimation software, which amounts to a simple spreadsheet, also suffers from the following:

- Most importantly, this approach ties pricing, part numbers, and quantities to the design. If the pricing changes, or if parts are added to the vendor's portfolio, one is left to migrate the design from one spreadsheet to the next — an approach that is error-prone at best.
- The template to capture the design is rudimentary and requires specific quantities of components. One is left to aggregate these outside of the estimation software.
- It does not have the logic to detect interdependencies between components.

4 A workflow that works

The estimation problem kept me up for several nights and also led me to the mirthful twin title of my talk at TUG 2012: Sleep De(p)rived Typesetting. Sleep deprivation is always a powerful motivator when seeking out a solution. The secondary title stems from the acronym that I gave to the workflow that I assembled: YAWN. This stands for its constituent components:

- YAML (YAML Ain't Markup Language)
- Algebra
- Words
- Numbers

In struggling with the software solutions offered from vendors, I yearned for greater flexibility. To my mind, if the vendors were simply going to package spreadsheets, why couldn't they do so with the (say) incredible capabilities of Lotus Improv — a superb spreadsheet product developed for NeXTSTEP — now sadly relegated to history.

The Wikipedia entry for Improv only reinforced my unease regarding existing solutions. Pito Salas, the lead developer of Improv, expressed it succinctly:

> ... it became clear that the data, views of the data, and the formulas that acted on that data were separate concepts. Yet in every case, the existing spreadsheet programs required the user to type all of these items into the same (typically single) sheet's cells.

What I was seeking was to decouple the specification from the materials list and the materials list from the budget estimate with its detailed bill of materials and labour costs.

Pito's words pointed me to the natural object-oriented framework: Model-View-Controller, or MVC. That is, if one expressed the specification in a human-readable form, and used a *controller* to analyse this specification *model* along with component and pricing *models* for alternative solutions, one could then produce a *view*, which would satisfy both the supplier and customer in providing a meaningful document expressing the estimate.

I began with XML as a representation of the model, but decided that I didn't need all of its features. Instead, I chose to use the simpler syntax of YAML, a data serialization language which allows for key-value pair hashes as well as arrays, both of which

I did need. The implementation of the controller in Ruby grew directly from the selection of YAML as the model language, since Ruby is able to read and write YAML directly and easily. I should note that there are YAML wrappers for many other languages as well.

Here is a small example of a design specification, to give a flavour of the YAML representation:

```
:specification:
    - :area:
        :name: F0
        :sections:
            - :room:
                :name: Common Areas
                :design:
                    :loads:
                        - :lightload:
                            :type: :mlv
                            :fixturewattage: 50
                            :fixtureqty: 10
                            :qty: 70
                    :controlstations:
                        - :gangbox:
                            - :keypad:
                                :qty: 10
```

Entries preceded with a dash indicate elements of an array, while keys that are surrounded with colons might contain either single or compound elements, and tokens prefixed with colons indicate constants. Because it is a simple text file, one is easily able to track the development of the design by using version control software. Additionally, since the structure of the document is free-form, one is free to interject key–value pairs as comments or secondary information that might be ignored during processing but is still valuable when reviewing the specification.

The controller first takes a specification and apply business logic to create a materials list. This intermediate form was also expressed in YAML, again allowing for visual inspection. In some cases, a request for pricing is given already as a predefined materials list with no design specification required. By having this intermediate form as a text representation, one is able to create it directly as an input to the remainder of the workflow. As a secondary step this materials list was combined with a pricing model also expressed in YAML to create a complete bill of materials. Different controllers and/or pricing models may be used against a common specification to explore their impact on the generated estimate.

Which leads me to the *view*. I believe that if we consider TEX as a toolset that can produce views on demand, we have rounded out the framework into a workable form.

In my estimation workflow, TEX's capabilities are used only lightly — in essence to convert tabular data of the bill of materials into formatted output — but as the document requirements grow, I can easily enhance the output into more pleasing forms. I also envision doing a summary document that creates an estimation `diff` that compiles and compares the costs of alternative product solutions.

5 Conclusions

TEX works very effectively in the standard object-oriented Model-View-Controller framework. By isolating specification and the control logic from final document production in the manner prescribed by the MVC framework and its associated design patterns, TEX can be of tremendous value in enabling workflows outside of the domain of publishing.

References

[1] Oren Ben-Kiki, Clark Evans, and Ingy. YAML Ain't Markup Language (YAML) version 1.2. `http://www.yaml.org/spec`, October 2009.

[2] Trygve Reenskaug. MVC. `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`.

[3] Wikipedia. Lotus Improv. `http://en.wikipedia.org/wiki/Lotus_Improv#ATG`.

◇ Pavneet Arora
`pavneet_arora (at)`
`bespokespaces dot com`
`http://blog.bansisworld.org`