Projects Understood as a Social Construct and as Code

William MacMillan

Last Compiled: 24 December, 2020

Introduction and Definitions

Planning and then administering the execution of a project, whether large or small, is a laborious process rife with manual labor, bookkeeping, and true uncertainty, in addition to risk. Currently, little is available in the public sphere, for love or money, that abstracts the structure of projects and scaffolds the various processes to allow for a rapid deployment or adjustment of a project. Even pre-planning is, itself, a project and suffers from the same lack of template or clarity.

Communicating the multiple dimensions of a project and the impact of adjustements is itself a manually intensive project. For every shift in timelines, for every audience that has different semantics, new artifacts of work need to be produced to communicate goals, deliverables, timelines, value statements, project structure, milestones, decision points, all of the things necessary to understand projects. Translating all of this into tasks available to workers is another manual step in the current *status quo*. For a new shift in timeline, waterfall charts of any number of semantic domains for audiences, and then any number of task tickets must be updated. Often, there is no physical linkage between where the overall designs for the project are kept (in tools such as Excel, or software good at managing projects at a high level) and the system used to distribute work such as Jira, which is quite good at a ticketed workflow.

Absent a unifying framework, no meaningful software can be built to rectify these challenges. Without a place to aggregate the various views and semantic domains of work and audiences, no meaningful optimization or stress testing is able to be completed for a single project or portfolio of projects. The absence is both in terms of software (there is no meta-programming available for project planning) as well as theory.[1] The fundamental building blocks of projects–tasks, workers, tools, goals, timelines, dependencies, products and budgets–are common and easily defined in nearly all examples of projects, regardless of the nature of any of those parts.

The way those things translate into specifics for any *particular* project, however, rarely makes the connection obvious. Project management software and thinking also does not translate well across domains of industries, project "types", products, and any number of dimensions on which these things can vary. This leads to any amount of lost energy, frustration, communication challenges, optimization challenges, slippage, cost overruns, deadweight loss of money and time, and frustration because the software of choice for managing the project must be contorted and managed externally to the tool itself for critical elements of the overall project.

The difference is largely just semantics, however. Take, for example, the concept of an artifact. Normally, artifacts are seen as elements that are not part of the core value proposition of a product. However, when selling a product to a technical audience, artifacts such as system diagrams can often be as valuable for the realization of value of a product as the product itself. A technical audience is swayed by the documentation as much, if not more so, than any amount of marketing material. In that scenario, it makes sense to consider the artifacts as much a part of the product as the tool itself. Or, alternately put, artifacts are simply products of their own right that the software is dependent upon for the realization of value.

[1] There are some development frameworks that do attempt to consider projects in this light, such as the Unified Process framework. These might be better understood as a semantic layer that attempts to link various project phases in a way that's designed to help govern the lifecycle of software in a firm.

At the end, a project can be neatly defined in an abstract way. As can all the other building block elements. The semantics hide some elemental truths. In the end, we are bags of meat burning energy to make something of tangible value.

The mapping of humans to products of work, and the aggregation of these things are what the following attempts to shape. The following will formally define the core concepts and connect them to semantic labeling which will then provide the theoretic backbone for a software layer which can interface between any number of project management tools and allow for the optimization of their utilization, as well as more meaningful ability to plan and optimize.

This discussion attempts to break down projects to their elemental components, and does so in a self-referential way with respect to management and project-management. With the underpinning elements and links between the concepts appropriately structured, mapping between different representations of projects and analyzing the work done or to be done becomes a far easier exercise.

Building block concepts

Some concepts that might seem relevant we will leave out of scope for the moment. For example, a value proposition. A value proposition is a statement about why a given product creates some utility for someone, somehow. In the case of well established products, this is a belief that copies of the previous prouducts will create utility. While interesting to consider, ultimately we are bypassing direct consideration where we might qualitatively consider the value prop. as being a unique thing different from other things. Conceptually, however, it is still something that requires human effort to create.[2]

Anything that we might want to consider of value results from human effort. That is to say, workers created it. Even in the most trivial of cases, a worker must expend effort. An alluvial diamond must still be picked up and brought to market. The building blocks of projects, then are the dimensions of consideration for production considered from this basic element: humans doing work.

At the most abstract level, a project is a ordered sequence of tasks that aggregate to a product of value. Put alternately, everything is a project, as long as it has tasks > 1, and it produces something tangible to someone. Below, I will further refine the concept of a project.

The dimensions of projects are:

- workers
- tasks
- goals
- products
- timelines[3]
- dependencies
- semantic context

Anything you might consider in addition to this list is a particular arrangment of these building blocks, or are only semantic translations from one of these. A workflow, for example, is, itself a sequence of tasks where the product of each tasks is required to proceed to the next step. Each subsequent task (or collection of them during one step of the workflow) is dependent on the prior task. In the case of a laborious workflow, it may be a project unto itself to get through that process.

[2] We will return to this idea when we consider semantics and labeling.

[3] Time is a complex element of this set of building blocks. Unlike many of the others, time should not be considered as an argument that separates members of the set of tasks or set of workers as different elements of their respective classes. However, it does become a defining characteristic to meta-level characterizations of projects, as time ordering of dependencies becomes a critical restriction.

Some Definitions

A product is an observable and measurable outcome from work. If there is not an observable and measurable (by at least two people, not including the worker), then no work was done. Even in the case of something that appears intangible (abstract performance art done for an audience of one), it will still be the case that the audience will have had a reaction that could be verified by a third party.

Goals must be enumerable, even in the most trivial or most complex cases. Showing that a product was created–whatever the quality–is itself a binary proof of existence. Other dimensions of the project can self-referent to serve as goals. For example, a goal can be the creation of a work product in less than two days from the start of a ticket, or within a time frame relative to the start of a project. Both tasks and projects must have goals, definitionally. Lacking a goal, a task or project has no condition from which it exits the state of work and becomes complete. Tasks need not have singelton goals. A task can have multiple goals or multiple products, as with projects.

A task is the basic functional unit of work in a project. While a task can be multidimensional in goals and production, it should be the case that any workers engaged in tasks in any moment in time cannot be simultaneously engaged in another. The documentation of tasks is, itself a task, which we will revisit later.[4]

A goal is a a measurable set of conditions which are used to judge the product of work. Goals can consist of externally provided conditions, or conditions derived from a combination of the other dimensions of projects. A product of work not created according to the timeline stops being of value at a certain point, for example.

Semantic context should be considered as the set mapping between different arrangements of the dimensions which allow for symmetric representations in different contexts. In the aforementioned workflow example, a workflow is a project. In some cases, it might be useful to consider when a structure that is similar is either a project, workflow, or both. Or when language would indicate they are similar.

Dependencies should be understood as antecedent elements necessary for the task at hand. This can be inclusive of raw materials, approval to work, a piece of code written by oneself or another, or another explicit element.[5] Raw materials, one should note, might be considered an alternate form of a budget, though this would be only semantically true. Budget should be considered a shared facet of a project across all tasks; raw materials are a specific input into a task.

What's a project, then?

A project is a collection of tasks done by a worker or workers where the aggregate of their work generates a product and there is, at a minimum, one goal uniquely ascribed to the product. Projects terminate in a task that declares the production process complete. In trivial cases this will be when something is shipped out of the factory. In most cases require unique planning, this will depend on a number of senior personnel declaring "good enough" which would be a form of external dependency to the latter tasks undertaken in the project.

A task is anything done by a worker that results in a tangible unit of output, which we'll generically refer to as a product. More accurately, we might think that the integration of the task output across all tasks as being the product or products that result from a project. The output from any given task is some kind of

[4] It's worth note that something that might be considered essential to project management is left unmentioned in this set: *documentation*. In some cases, the documentation may be the product of a task necessary for dependent tasks downstream, such as research. In most cases for project management, documentation is should be best considered as a parallel project running alongside the project and tasks. The observation that a task was done in some order or to some quality should be self-evident if the product meets the goals set for it. If it is not self-evident, then it is not a well defined goal. If additional documentation beyond the product is deemed necessary, it is for some alternate goal not related to the task at hand. To that end, comments on tickets on products like JIRA can conditionally be thought of as part of the product. If the output of a task is not easily understood by later users of the product of the task, then the "product" is really the joint presence of the product and the documentation. Otherwise, documentation serves no purpose and should not be utilized. In many ticketing systems a similar concept is status. In the case where a task is considered "done" by virtue of status, then the utlimate output to be judged by the goals of the task is the joint product and status condition.

[5] Alternatively, a useful frame for considering this more general form is critical path analysis. Anything critical on the path of delivering a task is a dependency, even in the case where the path may be obscured in any given view.

quantum of work.[6] This is to say a task has no meaning unless it generates an observable and measurable unit of output.

The task of defining the semantic unification of multiple workers and worker tasks falls upon special cases in the class that defines workers "managers" and "project managers." A generic worker cannot change the attributes of their task in most dimensions. Within a dependency structure driven by semantic links between workers (i.e. a team), managers can alter tasks attributes such as goals, dates, and so on.

Budgets are another special case of these fundamental elements. In a project inside an enterprise, it is assumed that a worker is to be paid for their work. It is the task of a manager to ensure this dependency for a task is in place appropriately. For a certain context (a big project ran inside a corporation), budgets form an important resource constraint. As an explicit attribute of a project, however, they are difficult to consider as an explicit attribute on their own. In isolation, budget constraints seem to have an important relationship with projects; they can define the scale and scope of projects. However, those limitations on scale and scope ultimately are the work product of the tasks from the special class of worker, the manager. The authorization to perform a task is, simply, a dependency on the task itself.

But why so formal and pedantic?

Every project management software handles the details of messaging to stakeholders, timing tasks and phases, producing artifacts, etc., in subtly different ways. Users utilize the different features in ways that can seem incompatible. In one place, a simple status may actually indicate the current task a worker is on, if it is well understood as a series of tasks in a workflow. In another, the same workflow may be represented as a project, with a series of tasks to represent each dependent task. A third is to model as you might in JIRA, with an overarching "ticket" where sub-tasks might actually represent the tasks, even if the general understand is of the ticket representing a task with a single deliverable. Fundamentally, however, these are all the same thing, a workflow. The difference is purely in semantics.

The formalization is necessary since, otherwise, you are simply attempting to play semantic games trying to guess who might have meant what in any given situation. Additionally, things that might escape notice in terms of their impact on a project become transparent. Consider approvals. Approvals require no "work" in a sense, but they are a real task for a manager and consume time. As the complexity of a product that must be approve rises, so does the time spent on the approval–or the approval becomes an inadequate measure of the product.[7] Likewise with documentation. In software engineering, most any given task is really two tasks: produce the code, and produce the documentation. Even in the case where you have written self-documenting code, you are still going through two parallel tasks.

The semantic blind spots of project management and the tools tools hide a critical element of understanding mappings between software and useful analysis of projects: the recursive impact of project planning, itself. Who decides status at any given moment? Who sets milestones? Who adjust timelines? It is almost tautologically true that project management defines a project as being something that is distinctly different from a simple workflow a worker might go through independent of any collective effort. Absent this recursive element, many of the parts necessary to map between platforms would fail, and planning of the total effort required is ultimately inaccurate and suffers from uncertainty.

[6] I realize how quantum-mechanics-like this sounds. This is somewhat intentional. This quote about the nature of measuring "work" in QM is notable for the parallel between the two contexts: "...strategy is to entangle the system of interest with a second auxiliary system that keeps a record of the work. The two systems evolve in time together, and the auxiliary system is then measured."

[7] An aside to any future humans starting out your career. If you are a developer, you should aim to make your code "work" in the sense it does what's asked, but be completely incomprehensible if you can get away with it. When it is burdensome to approve code changes from a dev, but you like the person, as a manager you'll rubber stamp it if you know it works.

Roadmap

This document is not meant as sales collatoral. I've pounded out some of the basic ideas and formalism necessary to adequetly understand and then code the elements necessary to bridge between the various solutions for task and project management in order to accomplish the goals of eleminating manual labor from project planning, create interoperability between tools, and allow for meaningful and far more accurate estimates of project outcomes.

The planned steps for forward progress on the product itself include:

1. Build data model of projects and project semantics
2. Build integrations with project management tools
3. Aggregate views and create analysis routines for forecasting
4. Build integrations with HR tools
5. Combine HR and project views for a total view of budgeting and costs and forecasting for projects and work.
6. ???
7. Profit

Structuring the Data Model

In this section we'll continue to be formal, but start to construct class functions using Python.[8]

For anything under consideration here, a worker is a human being. Humans are identified by a non-unique name, and a unique and arbitrary alpha-numeric identifier. Along with their name, people have a set of attributes, a profile, that is independent of their identity, but relevant for projects. This profile defines how a person might be able to complete a task. A software engineer, for example, may not be able to complete the work of a carpenter. Alternately, perhaps you need a plumber, but Bob the plumber charges $1,000,000 an hour, and this is outside the amount you are willing to spend. Identity is atomic, but worker profiles can be defined to an arbitrary level of precision on an arbitrary number of dimensions.[9] Additionally, people are not always available, which must be taken into consideration. Human beings require time away from tasks, as well as other physical considerations which make them unavailable for tasks. Finally, humans are semantically linked to abstract items which define the context in which they work. People may find they work on a team, which is intended to achieve particular output goals in a highly dependent way, for one example.

As a python class, the basic definition of a worker would be implemented as the following:

```
import datetime as dt
from uuid import uuid1
from timeboard.calendars.US import Weekly8x5

class Worker:

  def __init__(self, **kwargs):
    self.name = kwargs.get('name')
    self.profile = kwargs.get('profile')
    self.rate = kwargs.get('rate')
    self.availability = kwargs.get('availability')
    self.sem_context = kwargs.get('sem_context')
    if kwargs.get('worker_id'):
      self.worker_id = kwargs.get('worker_id')
    else:
```

[8] In this case, I'm using classes as they might be used in set theory, where a class is a function that defines groupings of sets.
[9] As a convenience, however, we will define the going rate of a worker as a basal characteristic of a specific worker.

```
      self.worker_id = uuid1()
    if kwargs.get('manager'):
      self.manager = True
      def alter_task(self, task_ids):
        return None

    if kwargs.get('pm'):
      self.pm = True

  def __repr__(self):
    return "<Worker name:%s, worker_id:%s>" % (self.name, self.worker_id)

  def __str__(self):
    return "<Worker name:%s, worker_id:%s>" % (self.name, self.worker_id)

# ----------

a_worker = Worker(name = 'Julio', availability = Weekly8x5())
print(a_worker)
```

```
## <Worker name:Julio, worker_id:06c881a2-4634-11eb-892b-06bd9fa42442>
```