Rational Scheduling

Computers excel at collating large amounts of data and processing it algorithmically. This competence is seductive and computers are sometimes employed to do things they are not really good at, though it isn't always apparent to us because neither are we. Projecting realistic schedules based on speculative estimates is one such case.

All project scheduling software has this much in common: they all estimate the amount of time and effort required to complete a task by introducing two estimates:

- *priorEstimate* – an estimate of the time and/or effort that will be required to complete the task. This estimate is made in the planning stages of a project, before work starts. It is based on past experience, a model, or, in the case of computer-based scheduling, the expedient necessity of producing something that *has the appearance of* being a rational relationship between budget, resource allocation, and work breakdown schedule.

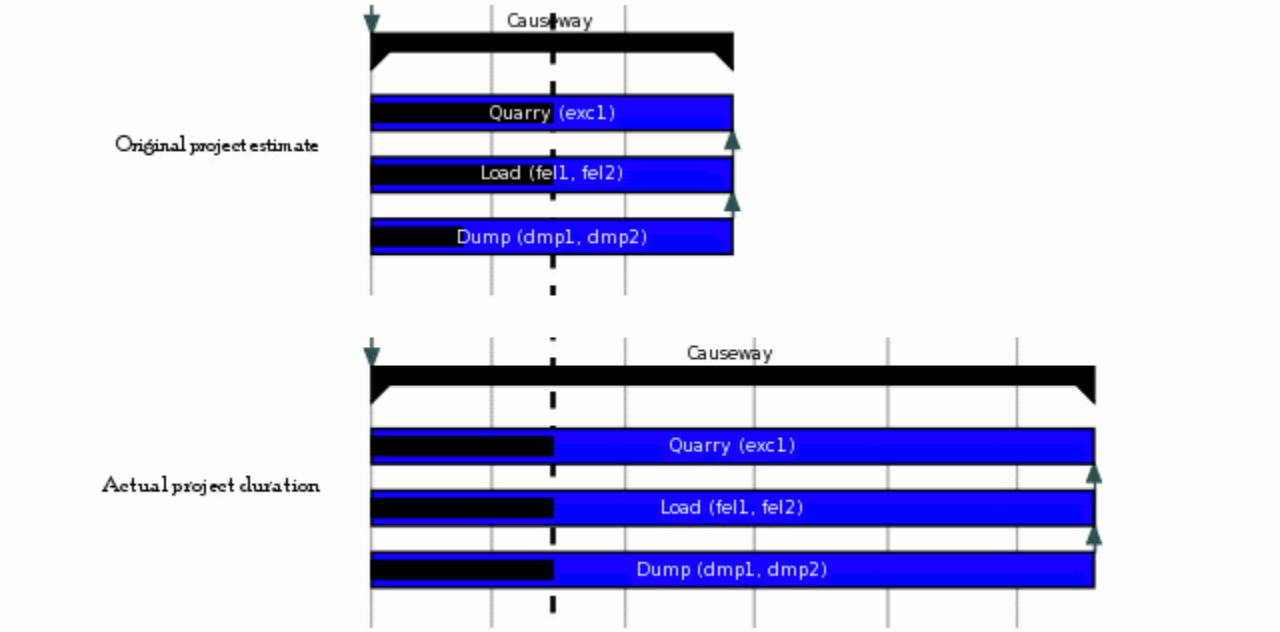
- *percentageCompleteEstimate* – once engaged in the task, a sporadic estimate of how much work remains yet to be done. The tension between *a priori*[1] and empirical estimates is a direct function of the difference between them, and it is not always clear which one is better.

While there is merit in estimating how long a task will take, it is actually one of those competencies that fall outside both human and computer prescience. There may be people with sufficient experience to make accurate estimates, and if there is sufficient data a computer may achieve a fairly reliable statistical prediction, but if there is both insufficient data and insufficient experience then any estimate is more speculation than forecast.

An Extreme Case

It is instructive to consider extremes for the lessons they teach us. Most well-planned projects are highly granular, but a long, unvarying, predictable task can be considered a microscopic view of almost any task in a sufficiently granular project. The following snapshot shows the Gantt chart of a causeway construction for a dam – a fairly predictable task which essentially resolves to little more than moving earth from a quarry to a dump. The snapshot was taken at what was believed to be the half-complete stage, at which time the causeway construction appeared to be running according to schedule; a schedule based on a simple measure - the amount of earth to be moved.

1 **a priorị: adj. & adv**. based on theoretical deduction rather than empirical observation.



In hindsight, though fifty percent of the earth had been moved the project was actually only twenty-five percent complete in terms of the time the task would require. Earth-fill had to be moved along a narrow causeway that was being continuously extended, and dumpers could not pass each other on the narrow causeway. The further it extended the longer it took to clear the causeway for the next load.

It became apparent that the project was starting to run behind schedule and experts were called in. The core problem was quickly identified and two proposals were tabled:

- replace three forty-ton dumpers with two hundred-ton dumpers,
- widen the causeway near the middle so dumpers could pass each other midway.

It was decided to replace the dumpers, but logistic delays in securing and transporting them to site delayed execution of the alternative - widening the causeway. Time passed and it seemed neither plan would offer worthwhile returns in the remaining time, for it appeared the project was finally nearing completion. Again, that assumption was wide of the mark; in reality the task was only fifty percent complete, for the dumpers were now maneuvering in extremely constricted spaces, hemmed in by overhanging rock, which made turning a slow and laborious process.

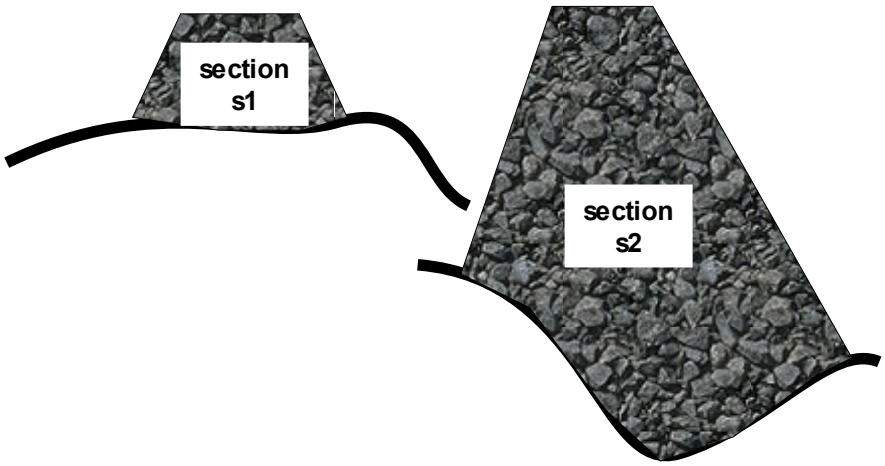
Fourteen weeks into what was originally to be a twelve week task it was clear that it was far from complete. Another dumper and and another front-end loader were allocated, which only slowed operations still further. Whereas a load-transit-dump-transit sequence took fifteen minutes at the

beginning of the task, it was now taking nearly forty minutes. While in terms of fill yet to be dumped the task appeared ninety-five percent complete, in reality that number was closer to eighty-percent. The task was on the critical path so they decided to immediately replace the 40-ton dumpers with 100-ton dumpers. The 100-ton dumpers were much bigger and harder to maneuver, so it probably didn't expedite matters much as it turned out.

In short, a task that was projected to take three months to complete took nearly six months. Nothing really went wrong, and the reasons for the delays were understood early in the process, but plans to address the delays waited on events and decisions until they became irrelevant. This is not unusual in a project - in fact it is more the rule than the exception. There is usually a better quicker way of doing things, but that way is discounted for a host of reasons: expense, other tasks need the resources, a task has already been planned and approved and now it would take more effort to change things than it's worth... These all account for tasks taking rather longer than anticipated for reasons that seem valid in the immediate context, but invalid in retrospect.

What could have been done better?

Using volume-dumped as a measure of progress was not the correct measure to use. A better measure would have been a time-normalized relationship between volume dumped and causeway extension.



$$TimeNormalizedMeasure = \frac{SectionalArea \times CausewayExtension \times TimePeriod}{VolumeDumped}$$

As an aside, a fundamental element of measurement lies in choosing the right thing to observe. Earthmoving is a relatively simple thing to measure, but what measure allows one to observe progress in developing a new microprocessor chip? Choosing or synthesizing the right measure is both a science and an art.

The project engineer made at least two mistakes:

- **underestimating the efficiency of the process in the early stages** - almost forty percent of the fill was moved in thirty percent of the time, and this gave a false sense of security in the early stages. The underlying problem was the the way measurements were done – non-normalized tonnage moved was seen as the primary measure; no other measures were employed to validate the primary measure.

- **modeling a nonlinear process with a linear model** – the model made some concessions to the relief of the underlying terrain, but it assumed that earth would be moved at the same rate throughout the task. Linear models are an artifact of the way we think, and we tend to accept them far too unquestioningly.

Model-less Project engineering methodologies

To control a system you must of necessity have a model of it. Without one there can be no causal relationship between what you do to a system and what you expect from it in response. If the system has no lag or lead terms in it then it can serve as its own model in a feedback control, but this is a special case.

While a model cannot be eliminated, its role can be reduced. Two techniques that reduce the need for an accurate model are *Iterative Measurement* and *Auxiliary Structure*.

Iterative Measurement

William of Ockham, a Franciscan friar, first formalized a powerful heuristic maxim. Known as Occam's Razor or lex parsimoniae (law of succinctness) it states that if offered two alternative explanations of an observable one should adopt the one least extraneous to the observable. *Given a choice of explanations, adopt the least complicated one*.

To use Occam's Razor formally we need a measure of complication. The Maximum Entropy[1] method uses information entropy to measure complication and achieves astonishing results with very little information, but it entails a great deal of mathematical formalism and can't be applied to all systems.

The alternative, our innate sense of a thing's complexity, is not always reliable, which means we must regularly test our perceptions against reality. Software engineers routinely employ *unit testing* and *refactoring* to test perceptions against reality.

1 Maximum Entropy is equivalent to finding that solution that assumes least beyond what is known about the system.

- **Unit tests** are a battery of tests performed every time code is changed. Unit tests can be used as the system specification: the tests are designed and coded before the system is designed. When the system passes all its unit tests then the system is complete.

- **Refactoring** is a process of periodic code rewrites that improves the understandability of code by changing its structure and design.

Whereas refactoring software is a relatively simple matter of changing some lines of code in an editor, refactoring non-software systems is seldom a practicable option. While we can refactor during the design process it is usually not feasible during the construction phase, and it is in this phase that avenues of simplicity usually reveal themselves.

Measurement on the other hand is something easily refactored – provided sufficient diversity and scope of historic measurement data is available. For example, a *VolumeMoved* versus *Time* measurement can only be refactored into a *VolumeMoved* vs *ResourceHoursWorked* measurement if resource working hours have been measured.

Why would one measurement system be better than another? Given two system factorings, then:

- one is either aesthetically and analytically better than the other, or

- it can be better validated.

Choosing measurements and targets for them to achieve is analogous to unit testing and refactoring. Implicit in the process is the need to perform it iteratively.

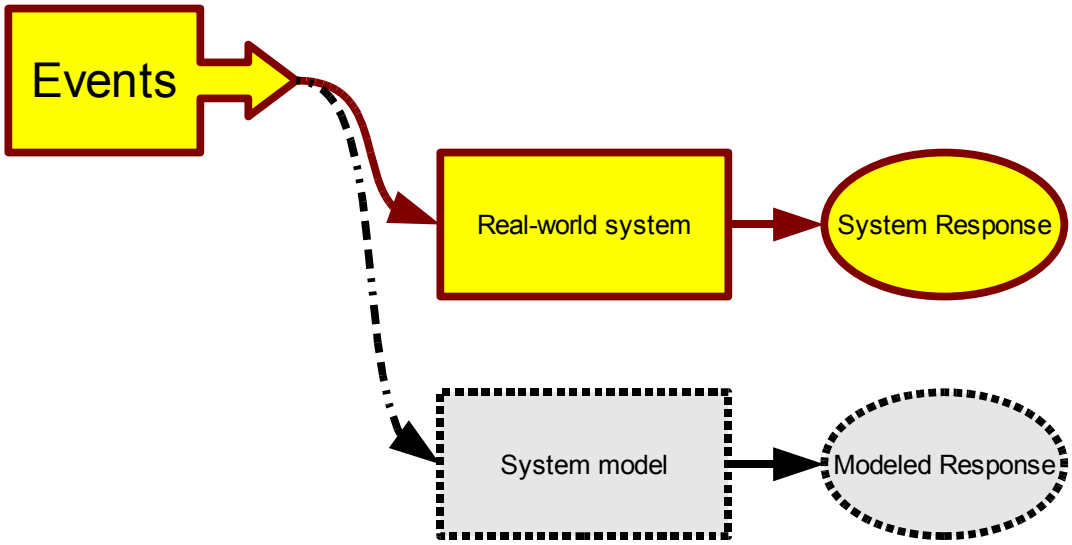
As an aside, much of this iterative theory is based on a range of software engineering methodologies collectively known as *Agile Methods*. If there can be said to be a continuum between *predictive* and *adaptive* project planning methodologies, then Agile Methods belong to the adaptive end of that spectrum. Whereas predictive methods try to anticipate the course of a project, adaptive methods react rather to immediate reality and context. The following points are lifted verbatim from *The New Methodology*, by Martin Fowler, which explains agile methods rather well:

- *Agile methods are adaptive rather than predictive.* Engineering methods tend to try to plan out a large part of the software process in great detail for a long span of time, this works well until things change. So their nature is to resist change. The agile methods, however, welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves.

- *Agile methods are people-oriented rather than process-oriented.* The goal of engineering methods is to define a process that will work well whoever happens to be using it. Agile methods assert that no process will ever make up the skill of the development team, so the role of a process is to support the development team in their work.

Perhaps the most defining characteristic of Agile methods is that they employ very short, rigid time frames. Whereas most tasks' primary aim is to conform with specification, an agile task aims to achieve as much as possible in a fixed period.

Control and Auxiliary Structure

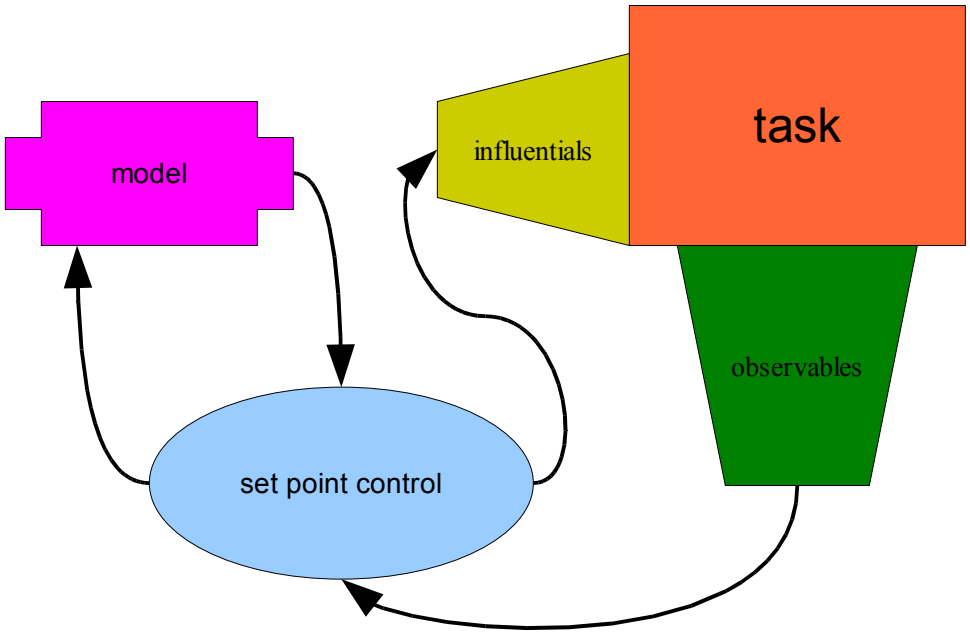


In any controlled system the difference between the *System Response* and the *Modeled Response* is adjusted to achieve the desired response. If you have time and expertise to throw at a problem then you can model it more fully and achieve better control. This is probably worth doing with some tasks, but it is not always obvious which tasks these are, and even good models have their blind spots.

However, it is not always necessary to base a model on an understanding of the exact physics of the problem. Sometimes it is sufficient to assume merely that some influential element of the task might impose a change-of-rate on another. An empirical First-Order Model (FOM) is often sufficient[1], but there are many situations where a first-order model is not only wrong, but misleadingly so. Once again we are reminded that computers make extremely poor judges of real-world worth and truth.

1 A model with only first-order differential terms in it. Higher order models imply higher orders of differentiation (or integration). In the case of the earth-fill task the FOM would have been ideal, for the causeway transit time imposed a delay term.

Auxiliary structure allows us to observe the task directly rather than via its conformance with a model's predictions. Control based on measured performance is clearly more reliable than control based on estimates. As regularly-spaced data becomes available it can be used to develop representative models – not something that can be done without data[1].



Auxiliary Structure

Auxiliary Structure lends a task sufficient structure that it has a degree of self-awareness. Whereas scheduling attempts to predict the amount of time a task will require, auxiliary structure monitors and controls the amount of time a task actually demands. In effect, the task has acquired an introspective capability. By observing its response to perturbations, such as reducing or increasing the hours worked by a resource, it is possible to probe inner mechanisms, develop models, and validate them. Introspective elements are as follows:

- Introspection
  - Preliminary Schedule
    - Milestones
    - Correlation milestones
  - Set-Point Control
    - Observation

1 As an aside, it is very important to perform measurements over a period while conditions remain static, or when changes can be compensated for. Many project engineers fall victim to measures that change their fundamental nature as the projects progresses.

    - Perturbation Analysis
    - Response to Control
    - Modeling
    - Model Validation
- Resource Efficiency
    - Measurement Correlation Analysis
- Financial correlation
    - Budgetary correlation
- Evolutionary schedule
- ...

With introspection it becomes possible for a task to generate its own milestones, evaluate its own performance, and calculate its own schedule. Milestones that have meaning within the context of the task are vastly superior to milestones generated outside context. For example, a software development project tends to make its milestones conform to its deliverables, but true milestones only become apparent when sufficient expertise and context has been developed. For this reason the first milestone of any task should be a preliminary schedule for that task. Any schedule prior to that milestone might have some value but very little confidence.

The schedule should be based on a formal model – even if that is merely an empirical one based on a similar task. As the task progresses every effort should be made to validate and improve this model; the task itself offers scope for this. For example, if a output is considered largely independent of a resource, then diminishing or removing that resource entirely should have no effect. The model evolves to become more reliable and representative as it is tested and improved.

Modeling the task encourages discovery of observables and exploring the internal state functions of these variables. From this grows awareness of the relationship between observables and inputs – of how to tweak the task.

An auxiliary measure is the financials. If there is a big difference between the measured cost of a human resource and the time usage or productivity of that resource, then there is a hidden truth somewhere amidst the contradictory measures.

The task is not independent of other tasks; there must be some correlation between tasks, even if it is one task surrendering a resource to another. Higher dimensions of correlation are associated with *interfaces* – a subject beyond the scope of this paper.