

The following excerpt is from R. Madachy, *Software Process Dynamics*, IEEE Computer Society Press, 2005

Brooks's Law Example

In the early software engineering classic *The Mythical Man-Month* (it was also updated in 1995), Fred Brooks stated:

Adding manpower to a late software project makes it later [Brooks 75].

His explanation for the law was the additional linear overhead needed for training new people and the nonlinear communication overhead (a function of the square of the number of people). These effects have been widely accepted and observed by others. The simple model in Figure 0.1 describes the situation, and will be used to test the law. Model equations and commentary are in Figure 0.2¹. This model is described below assuming no background in systems dynamics. More detail on model notations and equations are discussed in the Chapter 2.

The model is conceived around the following basic assumptions:

- new personnel require training by experienced personnel to come up to speed
- more people on a project entail more communication overhead
- experienced personnel are more productive than new personnel, on average.

It is built on two connected flow chains representing software development and personnel. The software development chain assumes a level of *requirements* that need to be implemented (shown as the upper left box in Figure 0.1). The *requirements* are transformed into *developed software* at the *software development rate* (rates are shown as the circular pipe valve symbols). Sometimes this rate is called “project velocity”, especially by agile methods people. The level of *developed software* represents progress made on implementing the requirements. Project completion is when *developed software*

¹ Some mathematically inclined readers may appreciate differential equation formulas to represent the levels. This representation is not necessary to understand. The following integrals accumulate the inflow and outflow rates over time to calculate a level at any time t using the standard notation:

$$\text{Level} = \text{Level}_{t=0} + \int_0^t (\text{inflow} - \text{outflow})dt.$$

$$\text{requirements} = \text{requirements}_{t=0} - \int_0^t (\text{software development rate})dt$$

$$\text{developed software} = \text{developed software}_{t=0} + \int_0^t (\text{software development rate})dt$$

$$\text{new personnel} = \text{new personnel}_{t=0} - \int_0^t (\text{assimilation rate})dt$$

$$\text{experienced personnel} = \text{experienced personnel}_{t=0} + \int_0^t (\text{assimilation rate})dt$$

equals the initial *requirements*. Software size is measured in function points, and the development rate is in function points/day. A function point is a measure of software size that accounts for external inputs and outputs, user interactions, external interfaces and files used by the system.

The *software development rate* is determined by the levels of personnel in the system: *new project personnel* who come onto the project at the *personnel allocation rate*, and *experienced personnel* who have been assimilated (trained) into the project at the *assimilation rate*. Further variables in the system are shown as circles in the diagram. Documentation for each model element is shown underneath its equation in Figure 0.2. Note that the time-based level equations are automatically generated by visually laying out the levels and rates with a simulation tool.

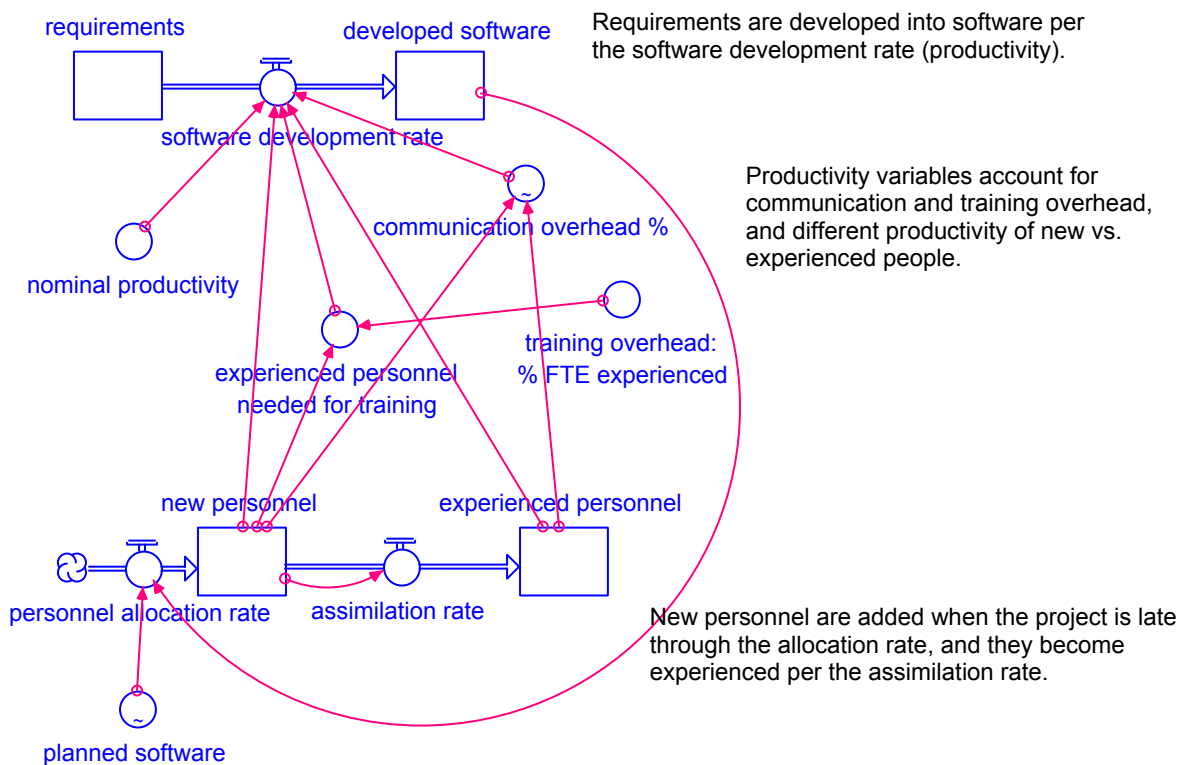


Figure 0.1: Simple Brooks's Law Model



- $\text{developed_software}(t) = \text{developed_software}(t - dt) + (\text{software_development_rate}) * dt$
INIT $\text{developed_software} = 0$
- DOCUMENT: This level represents software function points that have been implemented.
INFLOWS:
 $\text{software_development_rate} = \text{nominal_productivity} * (1 - \text{communication_overhead_}\% / 100) * (8 * \text{new_personnel} + 1.2 * (\text{experienced_personnel} - \text{experienced_personnel_needed_for_training}))$
DOCUMENT: The development rate represents productivity adjusted for communication overhead, weighting factors for the varying mix of personnel, and the effective number of experienced personnel.
- $\text{experienced_personnel}(t) = \text{experienced_personnel}(t - dt) + (\text{assimilation_rate}) * dt$
INIT $\text{experienced_personnel} = 20$
- DOCUMENT: The number of experienced personnel.
INFLOWS:
 $\text{assimilation_rate} = \text{new_personnel} / 20$
DOCUMENT: The average assimilation time for new personnel is 20 days.
- $\text{new_personnel}(t) = \text{new_personnel}(t - dt) + (\text{personnel_allocation_rate} - \text{assimilation_rate}) * dt$
INIT $\text{new_personnel} = 0$
- DOCUMENT: The number of new project personnel.
INFLOWS:
 $\text{personnel_allocation_rate} = \text{if}((\text{developed_software} - \text{planned_software}) < - 75 \text{ and time} < 112) \text{ then } 10 \text{ else } 0$
DOCUMENT: If the gap between developed software and planned software becomes 15% of the plan (75 function points), then add people. The time condition constrains the correction to one time, for demonstration.
- OUTFLOWS:
 $\text{assimilation_rate} = \text{new_personnel} / 20$
DOCUMENT: The average assimilation time for new personnel is 20 days.
- $\text{requirements}(t) = \text{requirements}(t - dt) + (- \text{software_development_rate}) * dt$
INIT $\text{requirements} = 500$
- DOCUMENT: The project size is 500 function points. This level represents the number left to be implemented.
OUTFLOWS:
 $\text{software_development_rate} = \text{nominal_productivity} * (1 - \text{communication_overhead_}\% / 100) * (8 * \text{new_personnel} + 1.2 * (\text{experienced_personnel} - \text{experienced_personnel_needed_for_training}))$
DOCUMENT: The development rate represents productivity adjusted for communication overhead, weighting factors for the varying mix of personnel, and the effective number of experienced personnel.
- $\text{experienced_personnel_needed_for_training} = \text{new_personnel} * \text{training_overhead_}\% _ \text{FTE_experienced} / 100$
DOCUMENT: Training overhead is the effort expended by experienced personnel to bring new people up to speed. It is the number of new personnel * the percent of an experienced person's time dedicated to training.
- $\text{nominal_productivity} = .1$
DOCUMENT: The nominal (unadjusted) productivity is .1 function points/person-day.
- $\text{total_personnel} = \text{experienced_personnel} + \text{new_personnel}$
- $\text{training_overhead_}\% _ \text{FTE_experienced} = 25$
DOCUMENT: Percent of full-time equivalent experienced person's time dedicated to training new hires.
- $\text{communication_overhead_}\% = \text{GRAPH}((\text{experienced_personnel} + \text{new_personnel}))$

 (0.00, 0.00), (5.00, 1.50), (10.0, 6.00), (15.0, 13.5), (20.0, 24.0), (25.0, 37.5), (30.0, 54.0)
DOCUMENT: Percent of time spent communicating with other team members as a function of team size. This graph represents the n^2 law in this size region, and was used in the Abdel-Hamid model.
- $\text{planned_software} = \text{GRAPH}(\text{time})$

 (0.00, 0.00), (20.0, 50.0), (40.0, 100), (60.0, 150), (80.0, 200), (100, 250), (120, 300), (140, 350), (160, 400), (180, 450), (200, 500)
DOCUMENT: Plan assumes a constant productivity of 2.5 function points per day.

Figure 0.2: Simple Brooks's Law Model Equations

Brooks's Law Model Behavior

In order to start out simple and understand the primary influences in the model, it is a high-level depiction of a level-of-effort project. It allows tracking the *software development rate* over time and assessing the final completion time to develop the requirements under different hiring conditions. When the gap between developed software and the planned software becomes too great, people are added through the *personnel allocation rate*. All parameters are set to reasonable approximations as described below and in the equation commentary. The overall model represents a nominal case, and would require calibration to specific project environments.

As time progresses, the number of *requirements* decreases since it represents requirements still left to implement. These requirements are processed over time at the *software development rate* and become *developed software*, so *requirements* decline as *developed software* rises. The *software development rate* is constrained by several factors: the *nominal productivity* of a person, the *communication overhead %*, and the effective number of personnel.

The effective number of personnel equals the *new project personnel* plus the *experienced personnel* minus the amount of *experienced personnel needed for training* the new people. The *communication overhead %* is expressed as a nonlinear function of the total number of personnel that need to communicate ($.06 * n^2$). This overhead formulation described in [Abdel-Hamid-Madnick 91] is further elaborated on in Chapter 5. The *experienced personnel needed for training* is the training overhead percentage as a fraction of a full-time equivalent experienced personnel. The default of .25 indicates one quarter of an experienced person's time is needed to train a new person until he/she is fully assimilated.

The bottom structure for the personnel chain models the assimilation of *new project personnel* at an average rate of 20 days. In essence, a new person is trained by one fourth of an experienced person for an average of 20 days until they become experienced in the project.

The *nominal productivity* is set to .1 function points/person-day, with the productivities of new and experienced personnel set to $.8 * \text{nominal productivity}$ and $1.2 * \text{nominal productivity}$ respectively as a first-order approximation.

Static conditions are present in the model when no additional people are added. These conditions are necessary to validate a model before adding perturbations (this validation is shown as an example in Chapter 2). The default behavior of the model shows a final completion time of 274 days to develop 500 function points with a constant staff of 20 experienced personnel.

The first experiment will inject an instantaneous pulse of 5 new people when a schedule variance of 15% is reached at about day 110. On the next run 10 people will be added instead. Figure 0.3 is a sensitivity plot of the corresponding software development rates for the default condition and the two perturbation runs.

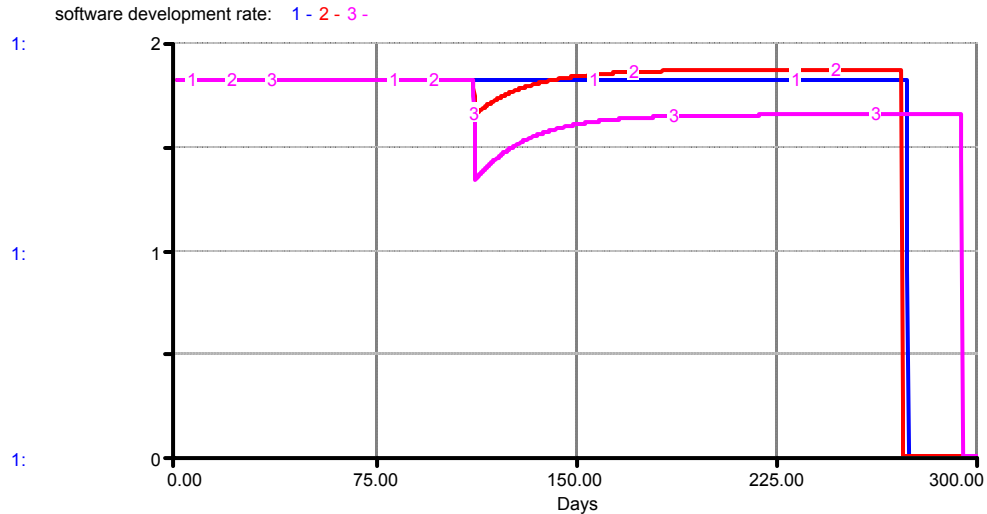


Figure 0.3: Sensitivity of Software Development Rate to Varying Personnel Allocation Pulses

(1: no extra hiring, 2: add 5 people on 110th day, 3: add 10 people on 110th day)

Figure 0.3 now shows some interesting effects. With an extra staff of five people (curve #2), the development rate nosedives, then recovers after a few weeks to slightly overtake the default rate and actually finishes sooner at 271 days. However, when 10 people are added the overall project suffers (curve #3). The initial productivity loss takes longer to stabilize, the final productivity is lower with the larger staff and the schedule time elongates to 296 days. The plunge and smooth recovery seen on both are the training effect. The extra staff productivity gains in the first case are greater than the communication losses, but going from 20 to 30 people in the second case entails a larger communication overhead compared to the potential productivity gain of having more people.

This model of Brooks's Law demonstrates that the law holds only under certain conditions (Brooks did not mean to imply that the law held in all situations). There is a threshold of new people that can be added until the schedule suffers, showing the tradeoff between adding staff and the time in the lifecycle. In some cases there is an eventual net gain in productivity and the project finishes sooner. The increased productivity must persist long enough to outweigh the initial productivity drains.

Note that only schedule time was analyzed here, and that effort increased in both cases of adding people. The effort would also be taken into account for a true project decision (though schedule may often be the primary performance index regardless of the cost).

There is a tradeoff between the number of added people and the time in the lifecycle. The model can be used to experiment with different scenarios to quantify the operating region envelope of Brooks's Law. A specific addition of people may be tolerable if injected early enough, but not later. The project time determines how many can be effectively added.

The model uses simplified assumptions and boundaries, and can be refined in several ways. The parameters for communication overhead, training overhead,

assimilation rate and other formulations of personnel allocation are also important for a thorough sensitivity analysis. A myriad of different combinations can and should still be tested. This model is analyzed in more detail in Chapter 2 to illustrate the principles of sensitivity analysis, and Chapter 6 shows enhancements to it.

This exercise has demonstrated that a small-scale and simple model can help shed light on process phenomena. Even though the model contains fairly simple formulations, the dynamic time trends would be very time consuming for a person to manually calculate and all but impossible to mentally figure.

Based on the insight provided, we may now clarify Brooks's Law. *Adding manpower to a late software project makes it later if too much is added too late.* That is when the additional overhead is greater than productivity gains due to the extra staff, as subject to local conditions.

This model is a microcosm of the system dynamics experience. Simplifying assumptions are made that coincide with the simple purpose of this model. The reader may need some faith at first since rationale for the full model formulation was not yet given. Many aspects of the model can be challenged further such as the following:

- determining the adequacy of the model before experimenting with it
- boundary issues: where do requirements come from, and why is there no attrition of people
- accounting for new requirements coming in after the project has begun
- varying the starting levels of new and experienced personnel
- alternate representations to model learning by new hires
- effects of different hiring scenarios (e.g. step or ramp inputs rather than a single pulse)
- different relative productivities for experienced and new personnel
- different communication overhead relationship
- the policy of determining schedule variance and associated actions.

All of these issues will be addressed in subsequent sections that elaborate this Brooks's Law model and others.