So You Want Brooks in Your Classroom?

Daniel Port  
University of Hawaii at Manoa  
Information Technology Management  
dport@hawaii.edu

David Klappholz  
Stevens Institute of Technology  
Dept. of Computer Science  
aklappho@stevens.edu

ABSTRACT
Fred Brooks’ seminal book, “The Mythical Man-Month” (MMM) is a firmly established classic in software engineering. Many of us feel compelled to use this work to help our students appreciate and put into practice the fundamental software engineering concepts contained between its covers. This often amounts to using “passive” lesson plans such as required readings followed by lectures and exams; these rarely fully satisfy our learning objectives. Rather, students often have mixed reactions to MMM with the result that it has little impact on their attitudes and practices, both in and out of the classroom. This paper outlines a more active approach to incorporating MMM into the classroom, one that we have refined over 6 years, at multiple universities and in both graduate and undergraduate courses. It includes learning objectives, a lesson plan, sample materials, an implementation discussion, and an evaluation of the approach’s impact.

Categories and Subject Descriptors
D.2.0 [Software Engineering]: General


Keywords
Fred Brooks, software development process, mythical man month

1. INTRODUCTION
Few software engineering educators would disagree that Fred Brooks’ book, “The Mythical Man Month” (MMM) [1] is a classic. It lays out with great clarity and profound insight many of the field’s fundamental challenges and principles, both past and present. This landmark book has inspired, illuminated, and educated generations of software engineering professionals and academics alike. As educators, we are compelled to include this book within our software engineering curricula. Indeed, many of us have MMM as required reading and include substantial lectures on its contents in our courses. Some of us have tried to use it as a course textbook or as a supplementary textbook but have met with limited success. Over the years we have observed the following:

<table>
<thead>
<tr>
<th>Action</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require students to purchase MMM “because it is a must have” and “you will appreciate it later”</td>
<td>Large numbers of used copies available on the net and in school bookstores</td>
</tr>
<tr>
<td>Teach MMM using a passive lesson plan (e.g. reading-lecture-exam)</td>
<td>Poor retention and assimilation of material; little application to class or outside projects; little impact on attitude towards SWE process</td>
</tr>
<tr>
<td>Require reading of MMM</td>
<td>Mixed student reactions; confusion as to relevance</td>
</tr>
</tbody>
</table>

The mixed student reactions are particularly confusing. Some common ones are:
“I enjoyed reading MMM, but find the material irrelevant to the course. A nice history lesson.”
“It is written by some old guy about a system long since gone”
“Things are different now. We know how to build things better now. There are more sophisticated tools…”
“Nobody builds operating systems like that anymore. Everything is web based anyway.”

Such comments suggest that students’ attitudes toward non-technical factors in software development do not change as a result of reading MMM. Matters are very different when a software engineering professional reads it. A cursory check of Amazon reviews of MMM reveals that almost every one has 5 out of 5 stars, indicating that MMM is as relevant today as it was when first published in 1967 and that it continues to have a significant impact on the professionals who read it.

Clearly, we would like our students to:
- Appreciate the significance and value of the observations in MMM
- Keep the book for future reference and inspiration
- Be active and engaged in applying the principles described in MMM
- Develop attitudes more aligned to those of software engineering professionals
In our experience simply reading, being lectured on, and taking an exam on MMM is ineffective. MMM requires “active” not passive learning. We must render MMM topics alive and relevant to current challenges and methods (e.g. “gutless” estimation leads to the need for effort estimation models such as COCOMO; the notion of a ‘surgical team’ applies to pair programming, XP and other agile methods). In short, we would like students to experience the challenges of MMM without having to participate in a large, risky project.

In this paper we take you on a tour of an “active learning” approach that overcomes many of the challenges indicated above. Our successful approach has been refined over that past 6 years within both graduate and undergraduate software engineering courses at multiple universities. We enthusiastically share it with the software engineering education community and hope that it will help students appreciate MMM and enhance others’ software engineering courses.

We begin by reviewing our learning objectives. We follow this with an example lesson plan that we have used to achieve these objectives. We conclude with a brief discussion on how we have evaluated the impact of our approach and why we believe it is a valuable learning tool.

2. Learning Objectives

Teaching software engineering is hard. Teaching it to students who have little or no software development experience is near impossible. It seems that the “school of hard knocks” is the only way in which people finally appreciate fundamental software engineering principles and practices. As Brooks puts it, "It is a very humbling experience to make a multi-million-dollar mistake, but it is also very memorable." Regrettably this type of learning generally occurs, if at all, long after our students have graduated; all too frequently it never happens. Our overall objective is to educate students to appreciate and practice the fundamentals. With respect to MMM, specific objectives include the development of an appreciation for:

- the use of effort estimation in software development
- the concept of project risk and the impact of unrealistic assumptions on project risk
- the reason that people and months are not interchangeable
- the impact of non-technical issues such as communications overhead, training, integration of parts, and staffing profiles on the effort required to complete a project

Realizing these objectives by themselves would be a major accomplishment. However, in pursing them using MMM, we have discovered that they tend to lead to the achievement of more general software engineering learning objectives such as:

- an understanding of the “no magic formula,” the “rarely any certain right or wrong” and the “plug into a formula for answer” mentality
- an appreciation for the utility and the limitations of models (e.g. assumptions) in software engineering decision making
- an appreciation of the significance and current relevance of the long established principles illuminated in MMM

Objectives are only as good as our ability to measure their achievement. For well-known reasons, grades are not a good measure of achievement of the kinds of objectives listed above. Rather, we have found two meaningful approaches – measuring changes in attitude towards software engineering practice (via the ATSE survey instrument [2,3]), and reflective student self-evaluation and course evaluation [4]. We will elaborate on these in the conclusion of this paper.

3. The MMM Lesson Plan

To realize the objectives indicated above, we utilize a lesson plan that consists of:
- reading MMM
- exercises based on the reading
- discussion of the reading and exercises
- reflection on reading, exercises, and discussion
- application of principles and practices from the reading and exercises

3.1 Readings

Students are asked to read all of Brooks’ “The Mythical Man Month” text in preparation for a homework assignment and class discussion.

3.2 MMM Homework Exercise

This is the key ingredient to the active learning of MMM principles and to rendering them alive, current, and relevant. Even if a student has enjoyed reading MMM and has found it worthwhile, the typical desire is to get “something concrete” to work with. The exercises are deceptively simple looking and appeal to this tendency toward “concreteness” of computer science and engineering students.

After seeing the following homework, students frequently are excited by the “tangible” mathematical model. We often hear, “Finally, something they can get an answer to!” and similar sentiments.

Problem 1:

[5 points]

Consider the following mathematical model which is one way of formalizing some of the concepts in MMM. This formula is not intended to be a predictive model. Rather, it is useful for illustrating and analyzing the concepts expressed in Brooks' book.

\[ T = \frac{E}{n^p} + bn + c\frac{n(n-1)}{2} + pS \]

- \( T \) = calendar effort (in months)
- \( E \) = total effort (in man/months)
- \( n \) = number of people
- \( p \) = degree of partitionability (where 0 = not partitionable, and 1 = totally partitionable)
- \( b \) = base training time (months per person)
- \( c \) = average intercommunication overhead between two people (in months/person*person)
Problem 2:
Show by setting the parameters of $E, p, c, b, S$ appropriately that the mathematical model in problem 1 matches that which is described by the indicated figures' contained in the MMM [1]. Either plot a graph that matches the figures indicated or explain qualitative characteristics in such a way that it is clear that the model has the same features. Provide a brief explanation of what the values for the parameters represent. You may find using Mathematica useful in completing the remainder of the questions. Also, when you are having trouble, try creating an explicit example by choosing particular values for the parameters. Avoid writing any programs for this assignment.

a) Fig. 2.1 (hint: perfectly partitionable task, no training, no intercommunications overhead, no subtask communication overhead) [2 points]

b) Fig. 2.2 (hint: un-partitionable task, no training, no intercommunications overhead, no subtask communication overhead) [2 points]

c) Fig. 2.3 (hint: perfectly partitionable task, no training, no intercommunications overhead, subtask communication overhead) [2 points]

d) Fig. 2.4 (hint: semi-partitionable task, training, intercommunications overhead, subtask communication overhead) [2 points]

e) Show that the model implies Brooks Law: “adding people to a late project makes it later.” [7 points]

Hint for (e): Consider a project that started with $n$ people which is late, but has completed some fraction $0<d<1$ of the total required effort $E$. Compare the estimate, $T_1$, of what it would take to complete the remaining effort $(1-d)E$ using the Brooks formula just with the original $n$ people versus the estimate, $T_2$, from adding $m$ more people. You may assume that for $T_1$, people no longer have to be trained, but for $T_2$, the $m$ new people will have to be trained. We may interpret Brooks’ law as “adding late people to a late project may make it later” so that what needs to be shown is that there is at least one case (given reasonable assumptions) where $T_2 > T_1$. So either find such a case using actual numerical values, or better, show this by doing some algebra that for $m$ and $n$ sufficiently large $T_2 > T_1$.

Problem 3:
For this problem we assume the characteristics of our class projects. Let us say that we have about 4 man-months to develop the product, so $E=4$ man-months. All the projects must be complete by the end of the semester, so we have a hard deadline.

$S$ = base sub-task communication overhead (in months)

Explain and derive (briefly!) where each term comes from. Provide explicit references (such as page numbers and quotations) from MMM.

Extra Credit: Effects of Object/Component Oriented Development

The component/object-oriented approach to product development is used to create a more natural partitioning of a system. As such, it incurs less intercommunication overhead (due to encapsulation) and provides an easier assignment of development subtasks. The tradeoff is that there will be more parts and the overhead on putting these parts together in a faithful way.

a) What parameters in the model does this affect and how are they affected?

b) If the task at hand is totally un-partitionable, how does the component/object-oriented approach affect it?

c) If the task is totally partitionable, the component/object-oriented approach will make the task “hyper- partitionable”. How does this affect the task effort?

d) Is there a “point of diminishing returns” to the component/object approach? That is, as the partitionability of a task increases, is there a point at which partitioning the task further increases the time needed to complete it (given a fixed number of people working on it)?

The connection of these exercises with MMM is quite clear. What is fascinating is how well they get students to think deeply about

1 The reader of this article may wonder why the figures are referenced in the MMM and not simply included here. This is a nice tactic to force students to at least glance at the MMM text!
the MMM material and the variety of answers we get to seemingly straightforward questions. Take as an example problem #1. While the instructions clearly ask the student to explain and derive where each term comes from, many students will choose to ignore the schedule effort term T as they believe it is self-evident. Other students will explain the term in ways similar to the following:

\[ T \text{ (effort in months)} \]  
\[ \text{The point of Chapter 2 is to illustrate the differences between manpower and time. “Cost does indeed vary as the product of the number of men and number of months. Progress does not.” (Brook’s MMM page 16)} \]

MMM does not always explicitly indicate a formula for each statement made. This leads to an amazing variety of answers, despite a common basis. Problem #1 again provides a nice illustration of this. Here are two variations on descriptions of the \( E/n^p \) term. Both found the correct basis for the term:

"Men and months are interchangeable only when a task can be partitioned among many workers with no communication among them." (MMM p. 16)

However one elaborated with:

\[ E/n^p \text{: This term is described in the statement: [quote here] In other words, time = (total effort) \div (number of people). The p in this term allows us to express the partitionability (such as sequential constraints). If } p = 1, \text{ then the task is nominally partitionable, whereas } p = 0 \text{ implies that the task is completely unpartitionable (MMM p. 17).} \]

and another student elaborated with:

\[ E/n^p \text{: When the task is highly partitionable (p close to 1), increasing the number of workers on the project (increasing n) decreases overall effort (T decreases). If the task is unpartitionable (p close to 0) adding more men (increasing n) has no effect on the total effort (T is constant).} \]

Grading is done on a point deduction system. For example, in problem #1 we look for common errors:

**Problem #1 grading**

-1 point for each term not explained or not explained well
e.g. “the bn term is the time for training” is inadequate versus “According to Brooks the time overhead for training a person is linear, hence if b is how long it takes to train a single person then with n people to train the total overhead due to training will be bn as in the model.”

-1 point for each term not referenced in Brooks

-1 point for poor explanation of \( n(n-1)/2 \) term (i.e. not describing how pair-wise interactions lead to the quadratic term).

### 3.3 Discussion of the reading and exercises

When the homework is due, there is always an excited discussion in class. The discussion typically begins with “that homework was impossible” and “the formula in the homework is not correct.” The students most often want to know what the “right” answer is for the problems. These provide excellent entry points into meaningful lessons on the nature of software engineering. When asked for a solution to one of the problems, we generally ask for a volunteer to provide theirs. Inevitably another student who did it differently will challenge the solution presented. Students who didn’t get a satisfactory answer will chime in to complain that the problem wasn’t clear or correct. After letting this go on for a bit, the instructor steps in and explains how all the points raised are valid and then points out the inherent ambiguity of the problems noting how on the surface they seem to be unambiguous. Consider problem #1 again. The formula is arguably correct, assuming you subscribe strictly to what Brook’s states in MMM and interpret his statements in the simplest ways. But one can easily take exception to Brooks’ claims and present many counterexamples that imply a more complex view is required. The main point is to indicate that models are based on assumptions that necessarily simplify the problem at had. There is no basis for thinking that these assumptions hold for every situation. The wise software engineer knows what assumptions are being made and how well they hold for any given situation!

### 3.4 Reflection on reading, exercises, and discussion

After the excited class discussion, students are given an opportunity to revise their answers to the homework problems. This is an essential activity to help solidify the insights gained from the discussion and from the initial experience of working the problems. It also serves to alleviate grade anxiety with the exercises as most students feel they did not perform well on the first try as a result of a lack of understanding of the nature of the problems and of the MMM material.

### 3.5 Application of principles and practices from the reading and exercises

Another key element to the MMM lesson is to have students put the principles into practice. For this we have them develop: (i) a sound roles and responsibilities breakdown for the class software development project; (ii) a project schedule that is consistent with effort estimates and accounts for the non-technical schedule overhead; and (iii) a project risk management plan.

### 4. Evaluation of impact of MMM lesson plan

We have outlined a basic lesson plan for making use of MMM beyond the traditional reading-lecture-exam format, but to what avail? Does this approach better help meet the learning objectives listed in Section 2? Is this a practical and valuable learning tool? To help address these questions we have considered three measurement approaches. First, making use of the attitudes towards software engineering (ATSE) survey [2,3], second accumulating evidence of application of MMM methods within course projects, and third, though feedback from student self-evaluations and course evaluations.
4.1 ATSE results

Beyond skills and knowledge, we believe that to eventually become successful software engineering professionals students must have an attitude towards software engineering that is compatible with that of the general software engineering community. Our previous works have included investigation into methods of measuring software engineering attitudes and their potential ramifications [2,3]. A survey instrument called ATSE was created to measure attitude, knowledge, and ability to apply principles with respect to a representative sample of software engineering professionals. As such, there are no specifically correct or incorrect answers. Instead, we look for how close a respondent matches answers with the majority of answers from our sample or professionals. The closer an answer is to one of these, the higher the score.

Here is a sample ATSE attitude question:

A27. When working on a team project, team members should start by coding their own modules and then figure out how to put the modules together.

a. strongly agree
b. agree
c. disagree
d. strongly disagree

Here is a sample ATSE knowledge question:

3. COCOMO:

a. was an early high-level business-oriented programming language
b. is the name of an early DoD programming environment
c. is a configuration management tool
d. is a cost estimation tool

Here is a sample ATSE ability to apply question:

4. Your team is developing a system for a small company that sells one-of-a-kind antiques on the web. The system will contain modules for: creating and editing descriptions of items for sale; uploading descriptions to the web site’s database; taking orders over the web; validating and processing credit card payments; maintaining the status –for sale, ordered, or shipped – for each item; creating records of ordered items for mailing by the shipping department; performing accounting activities for tax and other purposes. Progress on the various modules is as follows:

-creating and editing descriptions of items for sale: 80% complete
-uploading descriptions to the web site’s database: 90% complete
-taking orders over the web: 75% complete

This puts you about two months behind schedule: you will have to tell that to the client, but the client understands that scheduling of software projects isn’t a precise science, and will agree to a two-month extension. You are meeting with the client in two weeks. You estimate that you can accomplish one of the following by the time of the meeting: which one do you do:

a. work on all the modules so as to get them all, with any luck, to 90% or more of completion
b. get the editing, uploading, order-taking, and credit card validating functions 100% complete and working, leaving the other modules at the current completion levels
c. get the, status maintenance, shipping order and accounting functions 100% complete and working.
d. it doesn’t matter as none of the above three alternatives is any better than the others

To understand the scoring, if one answers ‘strongly agree’ for the attitude question, and the majority of our professionals answered ‘strongly disagree’ then the score for that question would be -2. On the other hand, if one has answered the same as the professionals, the score would be 2. A similar scoring is performed for the knowledge and ability to apply questions.

To measure the impact on attitude of our MMM lesson plan we preformed the following experiments:

1. Administer ATSE to students at the beginning of a software engineering course where MMM was not used.
2. Administer ATSE to students at the beginning of a software engineering course then ask students to update their answers after reading and lectures on MMM.
3. Administer ATSE to students at the beginning of a software engineering course then ask students to update their answers after completion of the MMM lesson plan described in this work.

It should be noted that students are never informed as to the purpose of the survey or “coached” as to what answers they should give. They are told it is simply a survey to help us improve the class and it will not affect their grade. We have considered the following measures:
Measure a: mean and variance of overall score
Measure b: mean and variance in change of score after updating answers
Measure c: percentage of score increases with respect to scores that were not already maximum

Experiment #1 is used to establish a baseline as there is no updating of answers. Results for experiment #2 and #3 are compared and tested for significance.

Unfortunately at this time we have not completed our data analysis. However, preliminary results indicate that:

- There is slight increase in attitude score for experiment #2 (Measure c had an average of 15%)
- There is a very significant increase in attitude score for experiment #3 (Measure c had an average of 65%)

While we are still processing the results for the other ATSE areas, anecdotally we have seem more visible differences in scores with experiment #3.

4.2 Evidence in course projects
The ATSE survey is a nice tool, but is not without its questionable aspects. It is especially difficult to map the ATSE areas to the overall learning objectives set forth in Section 2. To further understand and validate the impact of our MMM lesson plan on learning objectives, we carefully note any time a student uses a method or provides rationale based on the MMM material (including the homework problems). One surprising place this appears frequently is in the schedule rationale we ask students to provide when creating a project schedule. The students are asked to provide justification as to why their schedule is reasonable given the time and resource (e.g. number of team members, team member availability, etc.) constraints. They are advised to perform some form of effort estimation and to match this with their project schedule. They are also told that effort estimation if not entirely accurate and that estimating with more than one model is advised. Here it is very common to see use of both COCOMO and the Brook’s formula given in the MMM homework!

Every year we see that at least 95% of the students will mention or refer to specific MMM materials or methods.

4.3 Student self-evaluation and course-evaluation
Each student at the end of the course will submit an “individual critique” that asks him/her to evaluate project results, participation in the project, and reflections on the course. An example guideline for this can be found at [4]. We do not specifically mention the MMM lesson plan, but we do look for evidence that it has had an impact. It is common for us to receive specific comments referencing the homework, the MMM book, or a particular “lesson learned” similar to one discussed in the MMM homework discussion or within the MMM book. While we are still processing these results, well over 60% of the students will refer directly to some aspect of the MMM lesson plan.

5. Conclusions
While our evaluation is still ongoing, we are confident through over 6 years of practice that the MMM lesson plan described in this paper is highly effective in meeting the learning objectives stated in section 2. Furthermore, there are many additional benefits we continue to discover. In particular, we have observed that it is very difficult to motivate students at the very beginning of a software engineering course when they do not have much background knowledge and there is no project in which to experience challenges first hand. Starting with a lot of general overviews and theory more often than not loses a student’s understanding, interest, patience (and sometimes consciousness!). In contrast the MMM lesson plan provides a “tangible” hands-on entry point into the world of software engineering modeling and decision making without the need for extensive background or project development time. We have observed significant changes in attitude and ability to apply software engineering principles after completing the MMM lesson plan. In particular, students are more open minded, and appreciative of well-established, but perhaps dryly presented software engineering principles and practices. The hard fought lessons learned by those such as Brook’s now seem more current and important to heed.

We are excited to share what materials we have developed and results we have obtained with other software engineering educators. If you wish to try our MMM lesson plan, please contact one of the authors and we are happy to supply whatever materials we have developed. We are also very interested in related experiences and other innovative approaches to leveraging the MMM in the classroom.

6. REFERENCES