

# A4: Project Management

From  
System Dynamics Group  
Sloan School of Management  
Massachusetts Institute of Technology

15.876  
System Dynamics II  
Prof. Jim Hines

**Technical notes:** This case is based on a simple project model. The case contains an adequate analysis of the model behavior, so you will *not* need to simulate or analyze the model. On the other hand, you do need to understand the model. Accordingly, we have uploaded the model to our web site so that you can examine the structure (diagrams and equations). Of course, feel free to play with the model as your time and interest permit.

**Background.** You have just been hired to manage the development of the next release of Macrohard Word, the world's first word processor to not have a delete capability ("If its not hard, its not Macrohard").

The last several releases have all been significantly late, permitting competitors to eat into Macrohard's market share. In addition the past several releases have required much more staff – in terms of headcount and overtime – than budgeted. As a result expenses have risen while revenues have shrunk. You have decided to spend some time thinking about how to get the next release out on schedule and under budget. Your concentration is sharpened by the knowledge that your predecessor was fired for failing to achieve these goals.

You first talk to a number of programmers. The programmers feel that problems are caused when new people come onto projects late in the development cycle. It takes months for new comers to get up to speed, and in the meantime they are less productive and generate more bugs than experienced programmers. The workload on veteran programmers increases because there are more bugs to fix and because they need to train and check the work of the new folks.

One programmer put it succinctly: "Releases would be completed a lot faster if you'd just give us the people we need at the beginning of the program."

Your sense is that most of the programmers are dedicated to the development of the software, enjoy their work, and are willing to work long hours to complete the code. In fact one of your colleagues suggests this is a big part of the problem. According to this manager, "Programmers will work 24 hours straight – during which time they will

consume a 6-pack of Jolt and several boxes of donuts. They end up tired and wired producing more bad code than good.”

You listen to the different sides of the story and wonder how you are ever going to get a handle on the situation. Then you remember. A few weeks ago, you sat next to an extremely interesting MIT graduate student named Ken on a flight from Chicago to Boston. Ken said he was learning about a way of capturing lots of viewpoints into a single mathematical model. He called his discipline something like “system dominos” or “sister dynamics”. Despite your haziness on the field’s name, you distinctly remember Ken’s confidence that his approach solved tough problems.

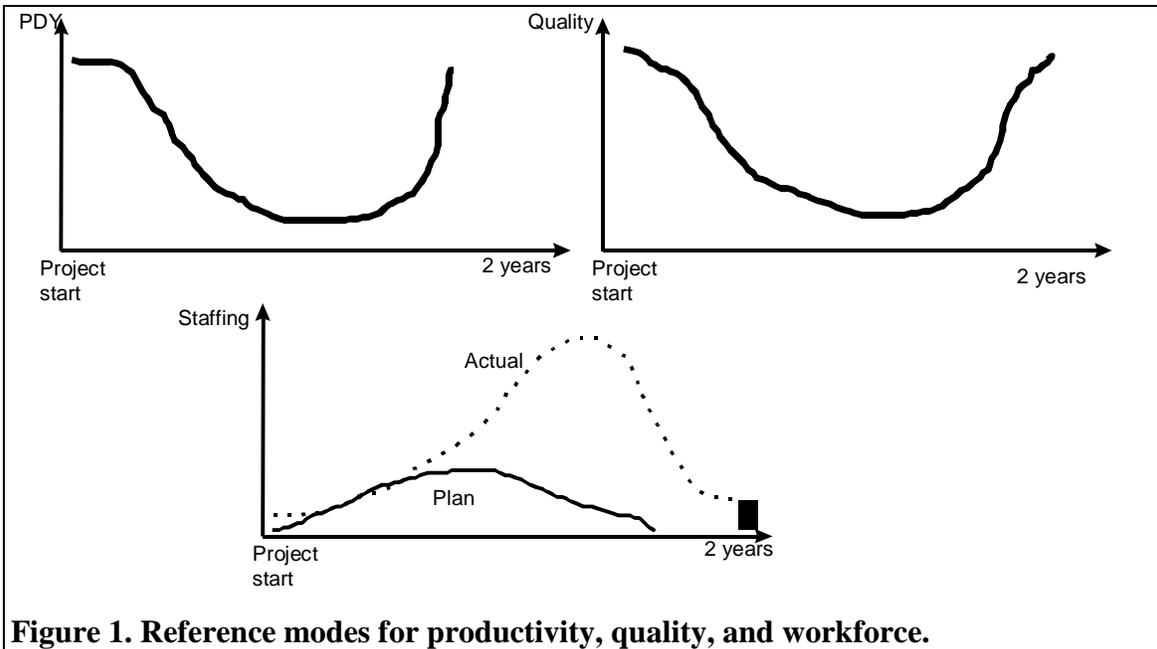
You give Ken a call to see if he would be willing to work with you on this problem. He is enthusiastic and is able to console you on several points almost immediately. First, Ken says, your problem sounds like a classic project model, a class of models with which he is rather familiar. Second, he says, it’s spring break next week so he can work with you for a week.

### **Putting together the model**

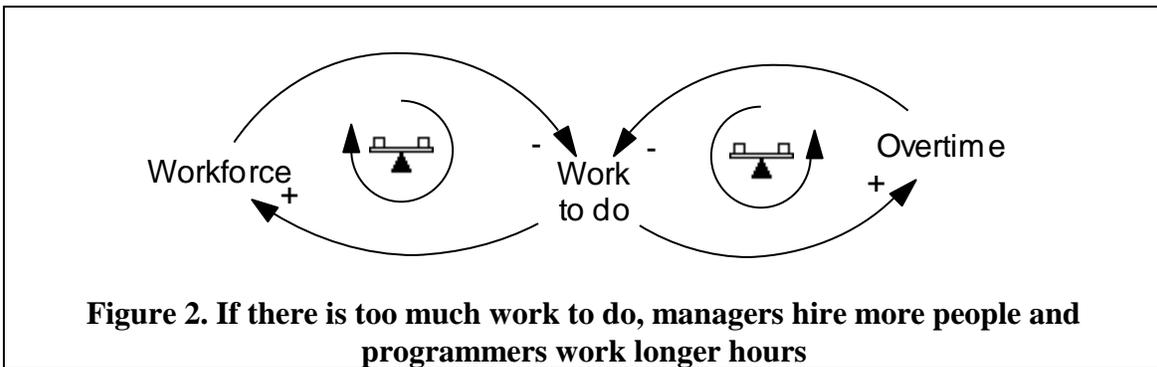
Ken comes out to your office during his spring break. He says the first thing to do is simply list variables that seem to play a role in your issue. You and Ken create the following list:

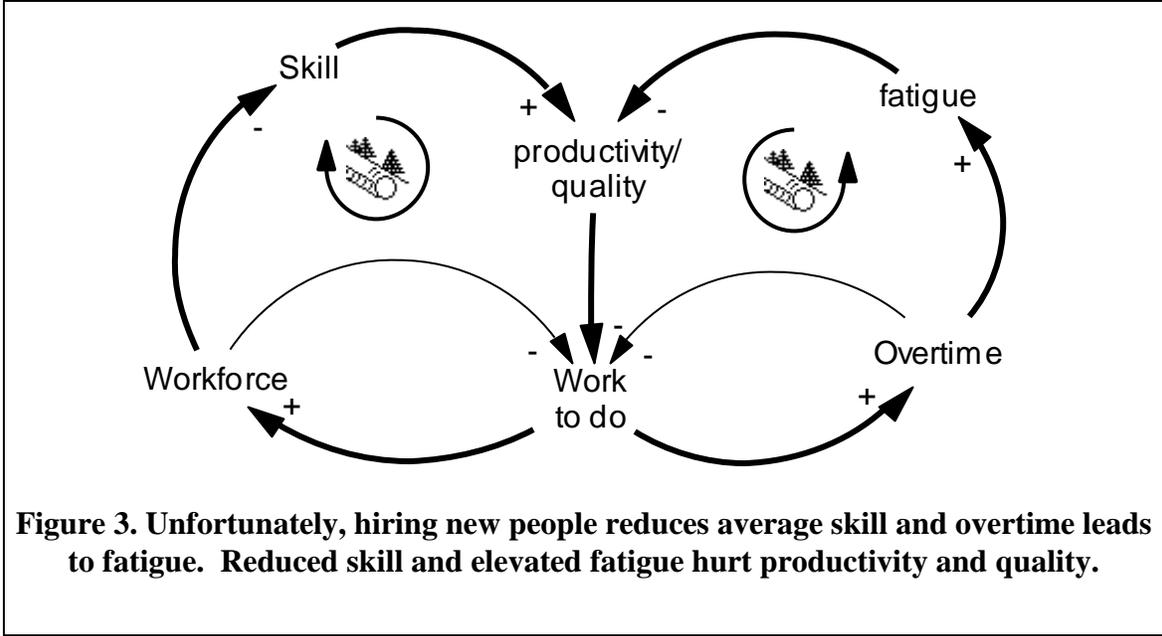
- Release date
- Workforce on the release
- New programmers
- Productivity
- Overtime
- Size of project
- Skill
- Quality
- fatigue

Ken says the next step is to choose a couple of the variables and draw “reference modes” – rough (and idealized) graphs of behavior over time. You choose productivity, quality, and workforce. After touching base again with a few programmers and managers, you draw the following reference modes.

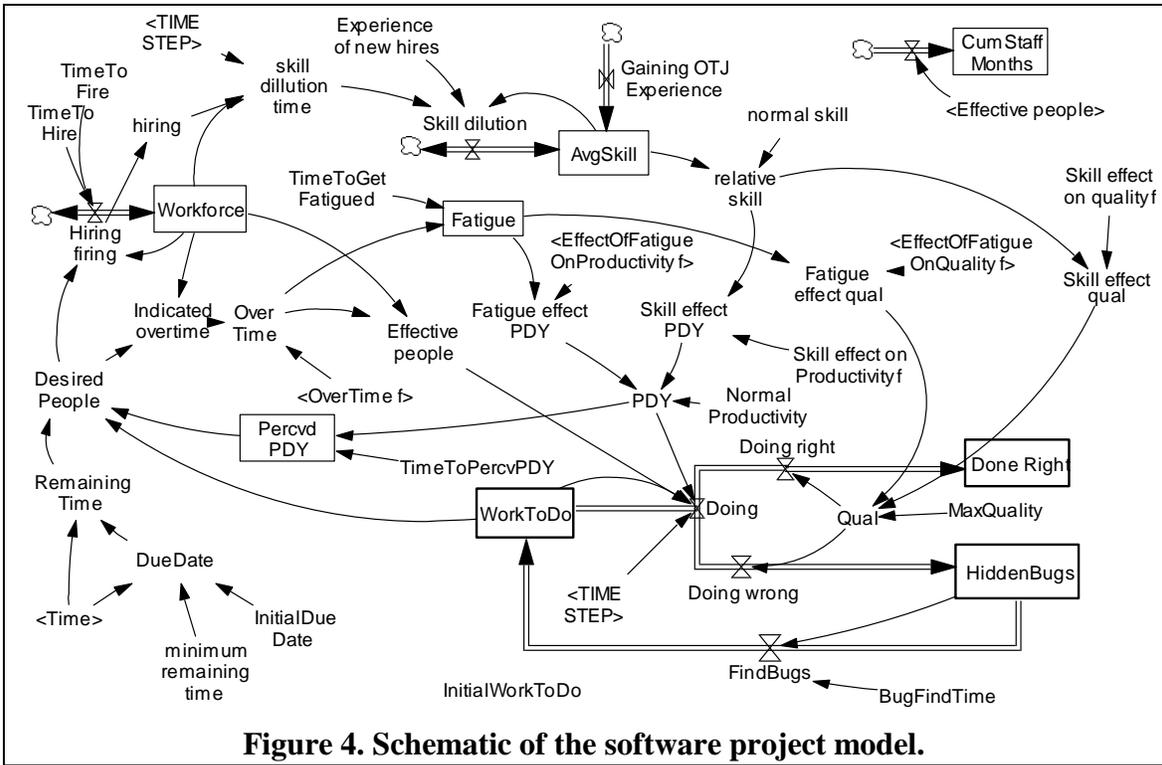


Ken says that you've done a terrific job defining the problem and that finally its time to develop some theories of what causes the problems. You bring in a couple of programmers and managers to work with you and Ken. After several hours the group completes a large spaghetti-like mess. The next day you and Ken pare the diagram down to something that is clearer to read. The diagram can be built up in two "layers":



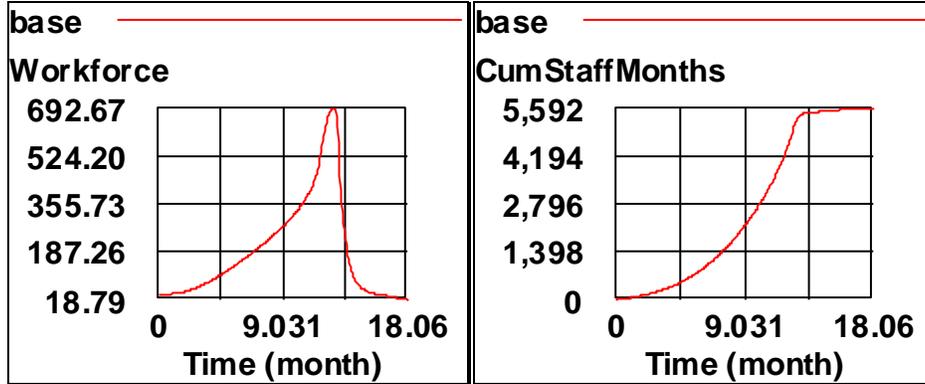


Suddenly, the whole situation seems much clearer to you. You feel energized. Ken says the next step is to create a simulation model of the theory to help you understand the theory more deeply. Drawing from his experience of project models, Ken helps you to create the following model – checking from time to time with your informants when you need a parameter or table function.

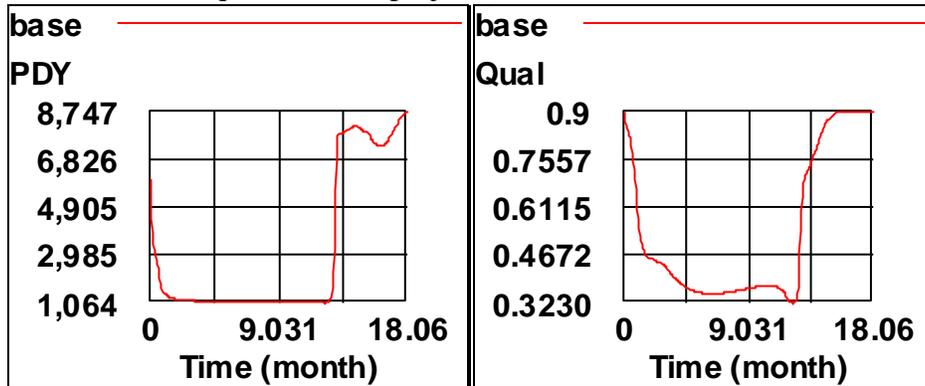


**Model Analysis.** You and Ken analyze the base simulation with the following results:

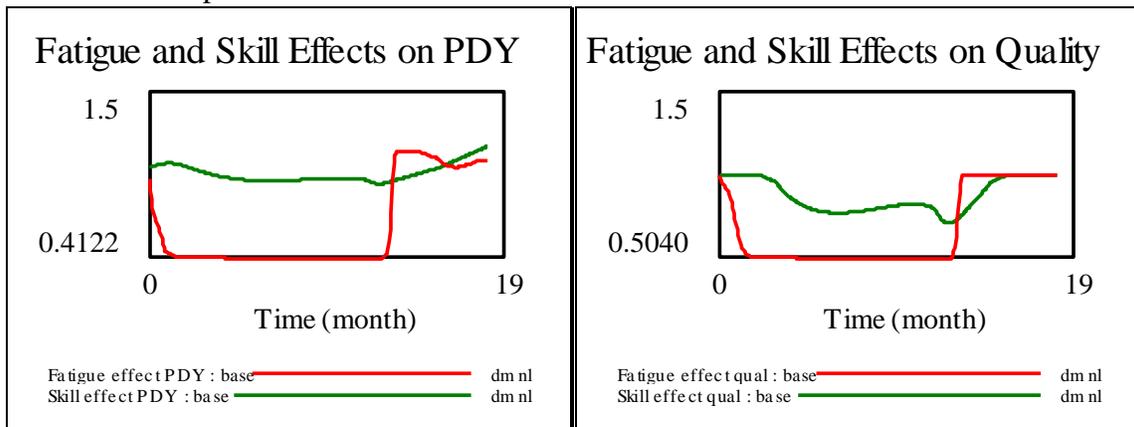
**A.** The simulated project completes over a period of 18 months (as shown in the two graphs below), even though the *scheduled completion date* is 12 months.



**B.** As anticipated, productivity and quality each show trouble. Surprisingly, perhaps, both recover in the final quarter of the project (see below).



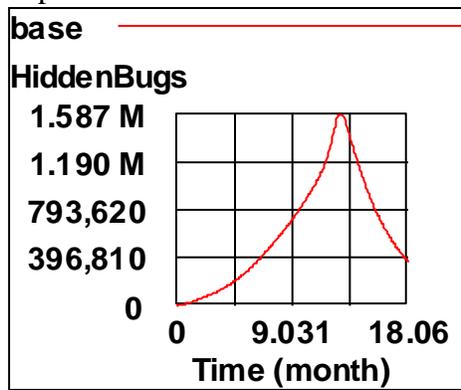
**C.** Interestingly, in the model problems with fatigue hurts productivity and quality much more than skill problems



You and Ken find that the fatigue table function is steeper than the skill table function. These table functions came from prior discussions with colleagues and programmers. But, now you know it's important so you follow up. You ask your informants to consider two

ways of “doubling” the effective workforce: Working 80 hours per week (instead of 40), or hiring twice as many people (and hence cutting skill in half). The consensus is that working 80 hours a week is much more damaging, particularly in the long-haul where fatigue continues to hurt productivity and quality of everyone. In contrast, skill levels (particularly of experienced folks) continue to improve the longer they work on the release.

**D.** You turn back to the graphs of productivity and quality (paragraph B, above). Why doesn't the project finish super fast when productivity and quality recover? Looking at the graph of the workforce (paragraph A), you realize that the model is not putting on many people at the end. Further examination shows that the people who *are* working during the final quarter are working less than 40 hours a week (i.e. there is *undertime*). A little more investigation shows that fewer people are working because there is very little work in the stock of work to do. Why doesn't the stock just empty out completely? Because a stream of new bug discoveries flow into the stock of work to do. As the plot below shows, the delay in finding bugs means there is a delay in finishing up the project. The bug discovery time is three months in the model – a figure that is roughly correct according to the people you speak to.



**Sensitivity testing.**

Ken suggests varying each parameter up by a factor of 2 and down by a factor 0.50 to further improving your growing understanding of the model. Your notes from this sensitivity analysis are:

- Sensitivity Notes
1. Bug find time: Reducing the bug-find-time causes the project to finish sooner, but also increases the total number of staff hours. The reason: Finding bugs faster means having more work to do, hence more overtime and more new people – therefore, lower productivity and quality.
  2. Initial due date: Starting the project with a due date that is further out causes lower cumulative staff hours, but also causes the project to finish later. The reason: With more time, there is less

overtime and less hiring, so higher productivity and quality. The flip side of this is that fewer people are on the project and they don't work as much overtime, so the project takes longer to complete. (Note: The project over-runs its new deadline. Managers in the model are still "surprised" by all the bugs (even though there are fewer) and by the fact that it takes a while to discover them.

3. Time to fire. Reducing the time to fire lengthens the time it takes to complete the project. The reason: Too many people are "fired", leaving too few to deal with the bugs that ultimately are on the way. Then it takes longer to "higher" people back. (Note: The time to hire is longer than the time to fire).
4. Time to hire. Reducing the time to hirer makes the project end sooner and requires fewer staff hours. The reason: Hiring faster means there is less pressure for overtime. And, skill-dilution is less harmful than fatigue. The trade-off works in our favor.
5. Time to get fatigued. Doubling the time to get fatigued has little impact. The reason: The time to get fatigued is small relative to the length of the project - doubling or halving the time constant doesn't make much difference. Also, lengthening the time constant means it takes longer to get fatigued, but also longer to recover.
6. Experience of new hires. Increasing the experience level of new hires (a parameter in the model) makes the project finish earlier and reduces the required number of staff hours to finish. The reason: The impact of hiring on average skill is lower.

## **Policy**

Your task is to improve the cycle time of software development and to reduce the total number of staff hours that go into each release. There are several policies that have been thought up by you and different members of the programming staff.

- Freeze hiring to avoid wasting the skilled programmers' time on training the new hires.
- Hire people instead of allowing overtime. This requires a faster hiring process.

- Hire people with more experience.
- Limit the overtime hours that an employee can work.
- Dedicate staff (i.e. programmers) hours to training the new hires in order to bring them up to speed more quickly.
- Find the errors in the code more quickly.
- Push out the initial due date to give the programmers some slack.

Think about these policies. Simulate as many of them as you have time for. Make sure you understand the behavior in the simulations. Decide which policies you like and which you dislike.

 You will need to convince other managers at Macrohard of what you think should be done and what you think shouldn't be done. Provide a discussion, aimed at a Macrohard audience, of each policy's pros and cons. Would the policy likely lead to a net benefit? If so, explain how you would *implement* the policy. Is the implementation going to be easy or hard? Expensive or inexpensive? Does the implementation have any downsides, perhaps not represented in the model? Summarize with a recommendation to either pursue or not to pursue the policy.

**IMPORTANT NOTE:** You should feel free to simulate the policies if you see an easy way to do so. However, your explanation of pros and cons should **NOT** rely on the model. That is, you should **NOT** say, for example, "Macrohard should do X because when we put X into the model the project finished faster and with fewer staff hours". Instead, offer a logical explanation that will make sense to someone who does not know you built a model and who will never see output from your model.

 What other policies would you recommend? Why?

 Why is it better *in this case* to refrain from mentioning the model or its output in your final explanations and arguments. Is there any situation in which it would be appropriate to base your final explanations and arguments on the model and its output? If you believe there such situations exist, please give an example or two.