CS 4488 Spring 2021 Project:
SoftPERT Software PERT Chart Tool

Building contractors know, with fairly good accuracy, how long it takes to lay 10 sq ft of brick. Civil, electrical, and nuclear engineers have similar numbers for most engineering tasks as do MBAs. This allows them to generate a dependency diagram; which tasks have to be completed before another task can be started, e.g. one has to add the wall studs to a house before the plumbing and electrical can be started. This dependency diagram in turn is used to generate a PERT chart (https://en.wikipedia.org/wiki/Program_evaluation_and_review_technique) which, given a start date for the first task in the dependency diagram, can calculate how long it will take to complete the entire project. Engineers and MBAs love PERTS and expect them to be used. It is almost certain that your software team leader or your leader's boss (or boss's boss) will be an engineer or have an MBA and will want a PERT chart for large complex software projects. (Note that I am using the term PERT somewhat broadly to encompass a fairly wide range of project charting tools. You might also look at GANT charts (https://www.projectmanager.com/gantt-chart).

But the variance or "fuzz" in task completion times complicates all this for some projects. Ideally each PERT task is provided with a mode (most likely) and 95% confidence "min" and "max" that can be used to calculate the min, mode, and max of all the dependent tasks and thus for the entire project.

Tasks for building a house have fairly well-known task times, so the added complexity of creating and propagating min, mode, and max task times has traditionally not been considered worth the bother. Many if not most project management tools only allow a single task time.

Computer scientists are appropriately suspicious of PERTS because our software tasks are far less well defined than those generally encountered by Engineers and MBAs. Software tasks and projects are not like laying brick, and have very-wide completion ranges that are difficult to estimate, particularly when one bad bug can increase the duration of a software task literally by a factor of 100. Remember Dorothy's project estimate rule: take the geek estimate, double it, and go to the next higher unit. ("If the geeks says the task will take 2 weeks, figure 4 months.") Given there is a factor of 10 to 20 difference between the productivity of the best and worse professional coders, Dorothy's rule is not a bad rule of thumb.

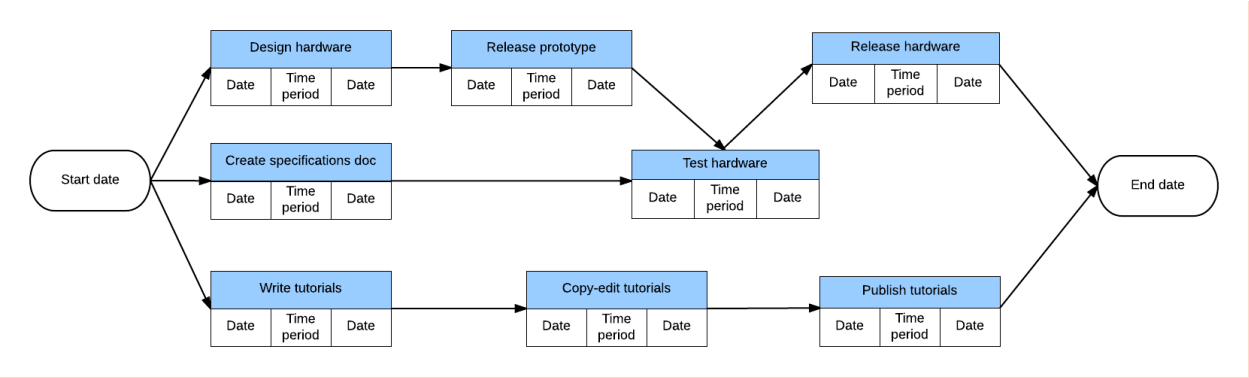
Further, Brooks points out that if the geek has underestimated the first 2-week task in a sequence by 100% (perhaps a new coding crew at the bottom of that "factor of 10 to 20 difference"), you cannot just move the entire project due date forward by 2 weeks. It is quite possible if not probable that the geek has underestimated all the tasks by 100%. This means that the entire project timeline may need to be doubled. Worse case, there may be months if not years of variation between the min and max times for a large software project, and the "most likely" or mode task time is an all but meaningless number.

Further still, computer scientists may fear that we may end up with a DILBERT manager (https://en.wikipedia.org/wiki/Pointy-haired_Boss), and if we provide a mode e.g. “most likely finish date,” he or she will blame us if the project is not complete by that “most likely” date. CYA says if you don’t provide a “most likely completed” date, then no one can blame you for not finishing “on time” (e.g. by that most likely date.)

Despite all these concerns, PERTS provide excellent visual analysis of dependencies and task decomposition that allows rapid comprehension of project status and facilitates “what if” analysis. This can be very helpful for larger software projects, particularly if your software is being integrated into a larger engineering project like a NASA mission, Nuclear reactor, Airbus xxx, or BMW 550.

Conventional PERT chart tools that just display numbers for each task can make it difficult to really comprehend the wide variation in project task estimated completion times (https://www.google.com/search?rlz=1C1GGRV_enUS751US752&q=pert+chart+examples&tbm=isch&source=univ&sa=X&ved=2ahUKEwipkZTUgJnjAhXUJjQIHXr8DqoQsAR6BAgHEAE&biw=1920&bih=969). Some software project managers are not computer scientists, have only written a few thousand lines of code, and don’t really understand complex large software development. Their bosses are holding their feet to the fire for hard due dates. Perhaps to be a bit cynical, if things are not done “on time,” such project managers may blame the coders by saying “they did not finish the project by the date they said it would be done” (e.g. the mode).

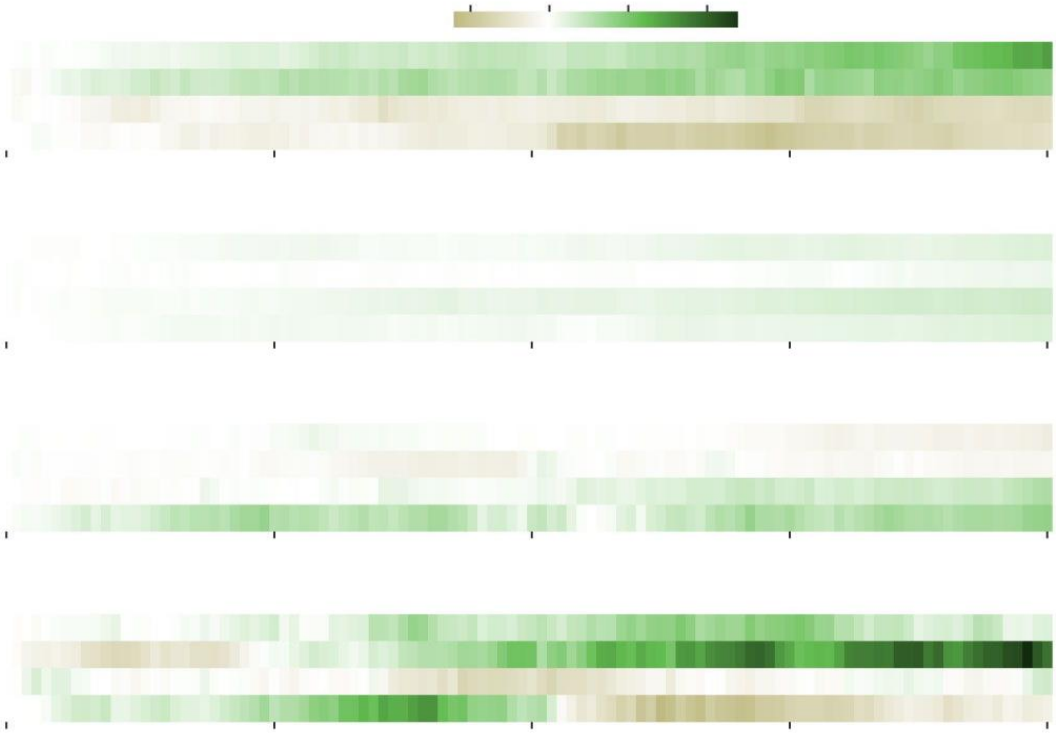
In https://www.lucidchart.com/blog/advantages-of-pert-charts-vs-gantt-charts we have an example PERT type chart. Time more or less goes from left to right. Note that all the tasks are the same width so even though time goes to the right, the task width does not indicate duration. Thus, this project visualization provides no indication of task duration let alone task duration variation. It difficult to understand the vagueness of the project completion estimates.



What is needed is a better PERT chart, designed by and for computer scientists, to provide useful information both to the software crew and to the bean counters who have to pay for everything and make the stockholders happy while not leading to incorrect assumptions of due dates. Such an improved PERT allows task mode, min, and max times, and some sort of visualization of the

PERT chart that makes clear to the Engineers, MBAs, and other "bean counters" how vague and "flapping in the breeze" the dates are for the later tasks in the project. The tool would provide an option to not display task MODE "due dates", but only the min and max dates and would provide some sort of visualization of the right-skewed distribution with the future to the right. The task bar can clearly have a length indicating the duration range.

One possibility might be a skew distribution shaped curve or a color heat scale or saturation indicating the distribution. The following figure from WSJ Moneywatch visualizes how saturation can be used to show changing data including distributions. Much better to see what is going on by following the URL:
https://www.wsj.com/articles/where-to-invest-when-the-fed-cuts-rates-11562491803



Another way to visualize the task's time distribution would be to have the actual box length reflect the task's mode time with a dotted line box extending to the max time and a skewed distribution curve superimposed on the box.

Your job is to design and implement a new PERT Chart gadget for real-world software development. The visualization for the tasks needs to show – either statically or dynamically -- the task name, min, max, mode and (optional) the skewed distribution. The entire PERT chart needs to be on top of a time scale across possible project duration with the future to the right. Once a task is designated as "completed," the system will have an actual duration that can be used with the visualization thought both the actual and estimated task times still need to be

displayed. The task should change color or otherwise indicate it is completed. However, you might want to still show the originally estimated range to allow everyone to see what the actual task times were in comparison.

(Bell and whistle.) Many coders will not want to have to enter a min and max, but just a "most likely" or mode, so you will need a default min and max as a function of the mode that optionally can be set for each project. As a default, use 85% of the mode for the min and 150% of the mode for the max (I just made these numbers up.) As an additional; "bell and whistle", keep statistics of the actual completion time vs the estimated completion time and eventually have sufficient data to automatically generate better numbers for default min and max. Both monitor and hard copy output will be needed. The actual PERT chart, needs to be able to be printed so it can be hung on the wall.

A relational DB schema that includes 2 key relation schemata might be considered. (Prime key is underlined. Not happy about the last attribute name but it does describe the contents):

Task<task#, taskName, estModeTime, actualTime, minTime, MaxTime, …..>
TaskDependency<DependentTask#, Task#OnWhichItDepends>

The initial version of the tool needs to run stand alone on a laptop. One approach might be to use C# and the free Microsoft RDBMS that can run stand alone with the DBMS in a single file that can have version numbers added to the end of the file name. Using Express, Developer or working in visual studio, you can run a stand-along database (*.mdf file). This may seem a bit "dorky" but it will make it much easier to debug and port for the initial release, and it will allow fast and easy "what if" and "PERT" version control. (https://www.microsoft.com/en-us/sql-server/sql-server-downloads) You will need to design on a minimal set of useful features for your initial release to insure you have something useful as soon as possible.

Consider the following for calculating PERT probabilities: https://workspirited.com/how-to-calculate-pert-estimate

Expand Existing Software from Previous 4488

In the potential team project folder there is a .zip file containing the code from a previous CS4488 team that worked on a software PERT/GANT project. While they have a nice start, there is much left to be done. Analyze the provided software including the mockup of an alternate task visualization using saturation and – in consultation with the "user" e.g., your intrepid professor --develop a list of tasks that need to be completed to make this a useful tool. As a start, use the SoftPERT tool to manage your project. Your goal is to make softPERT into a useful gadgt your team can use to manage your SoftPERT software development project (recursion is always reliavent 😊).

The provided task Visualization mockup code shows one possible improved task visualization

Test Problem: As a simple first pass, your PERT/GANT should handle the following simple problem:

FooProject
Starts 1Jan21

First setup the following:
Task1: most likely duration 2 staff days worse case duration 10 staff days
Task 2: most likely duration 1 staff days worse case duration 8 staff days
Task 3: most likely duration 3 staff days worse case duration 12 staff days. Task 3 is dependent on completion of tasks 1 and 2
Task 4 most likely duration 1 staff days worse case duration 10 staff days. Task 4 is dependent on the completion of tasks 3 and 2.

Then add the following:
Task 1 actual duration 4 staff days. Task 2 actual duration 2 staff days.
Task 3 actual duration 11 staff days.

Task 4 divided into task4a and task4b. task 4a is dependent on the completion of task 3 task 4b is dependent on the completion of task 2. Task 4 is dependent on the completion of tasks 4a and 4b.
Task 4a has a most likely duration of 1 staff day with a worse case duration of 5 staff days
Task 4b has a most likely duration of 1 staff day with a worse case duration of 5 staff days.

More complex situation: Project people: George, Mary, Sam, Jane
Who worked on which task? How many days put in by each person on each task?