Project management in a complex

international project

by

Michael Grabinski
University of Applied Science
Steubenstr. 17
89231 Neu-Ulm
Germany

Originally written as a master's thesis in the year 2000

Contents

Abstract 3

Introduction 4

Part 1: General Considerations **6**

1.1 The background of the underlying project 6

1.2 General remarks about software development 7

1.3 The project organization 10

1.4 Desired approach 13

1.5 Choice of software tool 14

Part 2: The planning manual for engineers **17**

2.1 The process 17

2.2The resource pool 19

2.3 The program 22
2.3.1 The plan 22
2.3.2 The links 25
2.3.3 The workload 27
2.3.4 Checking 31

2.4 Merging the projects 32

2.5 Using the program 35

2.6 Appendix Part 2 38
2.6.1 Bugs in MS–Project 38
2.6.2 Metrication 40

Part 3: Conclusions **44**

3.1 How the software project went along 44

3.2 Lessons learned 48

3.3 Alternative organizations 53

References **61**

Abstract

Building on a real life example, project management is described. The stress lies on planning.

In the first part the real life example is described. A giant software development project (250 engineers for over ten years) is used as a case study. Some general remarks about software development are given.

The second part contains a hands–on description how to plan by using the software tool MS–Project. It can be used at least as a skeleton for a planning handbook in almost every imaginable project. Its appendix gives an overview of metrication methods.

Part three gives the conclusions. It stresses the importance of proper planning. It concludes that planning is an integral part of development rather than some administrative work. A fundamentally new profit center organization for project is defined. Its controlling (project management) is discussed in detail.

Introduction

Skills in project management are crucial. More and more tasks in modern business life do not repeat. Quite often there is a goal for a particular time. To reach it one has to involve particular people. This group or team needs management. (For a more rigorous definition of the words "project" or "project management" one may consult standard literature, cf. Ref. 1.1) In contrast to ordinary structures project management is only necessary for a certain period of time. After the project is finished the team will be dissolved. From this one may conclude that project management must be the same as ordinary management. Cum grano salis it is true. However, this "grano salis" (= grain of salt) can cause tremendous difficulties. As mentioned above project management is limited in time. Ordinary management structures will develop over years. They rarely work at all before say half a year (or much later in big organizations). While a project is normally "short", there is no time to develop a management structure in the same way ordinary structures grow. From this one may conclude that project management is just a superior form of ordinary management. This is a statement I would agree to. So why does ordinary management still exist? Well I would predict that it will fade sooner or later. New concepts like workflow management are nothing more than a project plan for ordinary (and maybe repeated) tasks.

From this it is clear that project management is *the* discipline for every manager. Needless to say, one has to consider international project management in our modern world.

Having stressed the importance of project management it is clear why I have chosen it for my thesis. However project management is in some sense close to driving a car. One will learn it by doing only. Purely theoretical considerations will improve your driving skills little to not at all. The same is true of project management. Therefore I will describe a real life project in my thesis. The project under consideration is a software development project. It was so big and complex that almost every theoretical thought can be brought in. For a short description of the software project please see chapter 1.1. I will always call it *software* project. This is not because my considerations are limited to software development projects. (They are almost universal.) I will use the term software project when I mean the particular project described in chapter 1.1. I will also talk of projects in general. I personally was involved in the software project as a management consultant. I was called in for a management consulting project to improve the project management of the software project. (Again another use of the world project.)

The rest of this thesis is organized as follows. It consists of three parts. Part 1 will describe the software project (1.1, 1.2, and 1.3). It will also describe the goals of our management consulting project (1.4 and 1.5). Part 2 is a manual for planning a project. It is very much "hands–on". It assumes the use of the planning tool MS–Project 98. However, it contains many useful thoughts beyond MS–Project 98. Its appendix 2.6.2 gives a brief theoretical summary of metrication methods. Part 3 concludes the thesis. It will give the lessons learned from the described project. It will also give ideas for alternative project organizations.

1. General Considerations

This first part contains general considerations about project planning and project management. It will also explain the background of the underlying software project and management consulting project, respectively.

1.1 The background of the underlying project

Our client had to develop a flight control system (FCS) for a military plane. It is a big software development project. The FCS is the central computer of any modern airplane. The main task in building it is to develop its software. In the present software project there was a particular difficulty. The military plane under consideration was completely unstable. A "normal" plane is always stable. That is to say, a small steering command will lead to a small change. If a stable plane is slightly out of the desired aerodynamic regime it will fall into a stable trajectory by itself. That's why it is called stable. On the contrary, an unstable plane will not find its way back into a stable trajectory. It is like a car where you steer slightly to the left and than take your hands off the wheel. An ordinary stable car will find its stable trajectory and drive straight. Not so in an unstable car (which does not exist in reality). It will drive to an almost arbitrary direction if you take the hands off the wheel.

In the present case the military plane is not unstable by accident. It is made so intentionally. Being unstable is a big advantage for a fighter plane. It can change direction almost immediately. Aerodynamics prevents that in an

ordinary stable plane. This obvious advantage has a drawback. You can't fly such a plane without a sophisticated FCS. The software is so complicated that up to now nobody has build such an airplane. (Seeing the difficulties in the present software project the management is convinced that nobody will ever develop such an airplane again.)

This short explanation of the technical background is given only to stress two points:

a) The software project is big and complicated
b) The outcome can't be predicted. During the development period nobody could say for sure whether a certain functionality of the software will be achieved. Maybe it is impossible for principle reasons. (It will be impossible, if the chaotic regime is reached, cf. ref. 1.2)

1.2 General remarks about software development

A professional software is developed in three steps. These are denoted by the three boxes in fig. 1.1. (There the software development process is

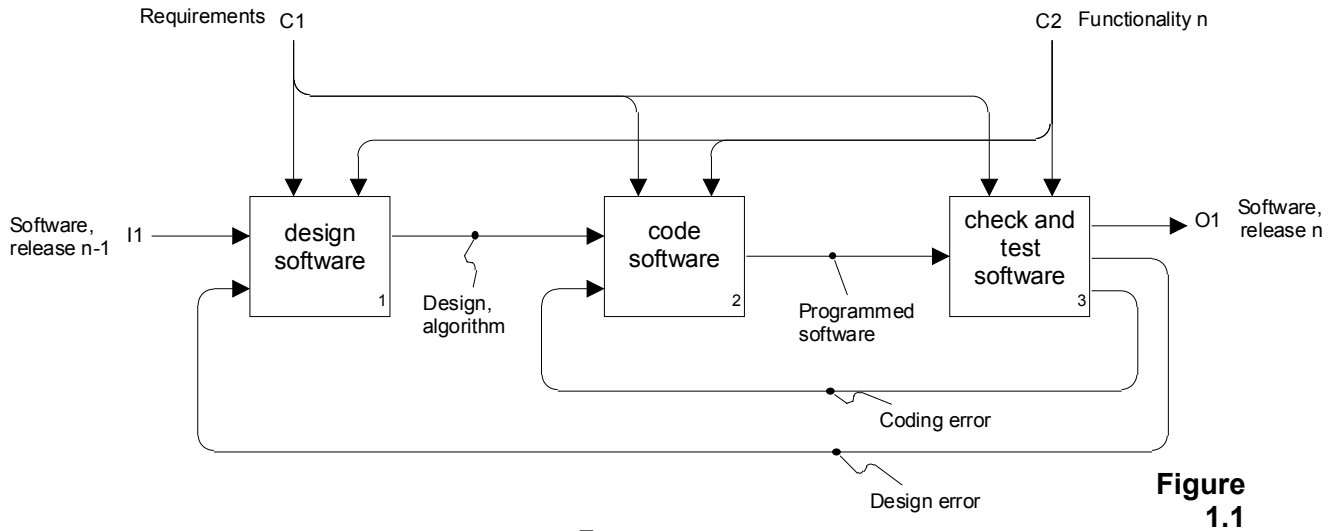
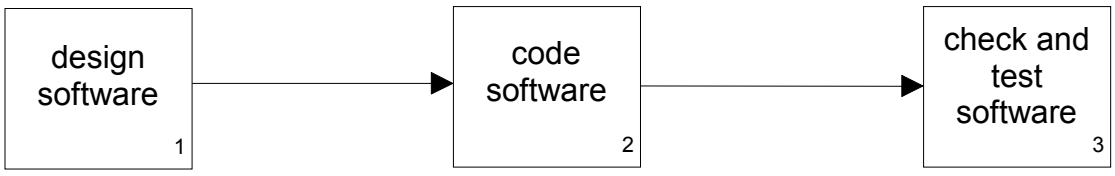


Figure 1.1

displayed in SADT (=strategic analysis and design technique).) Firstly, the general design is developed. There the problem is solved in principle. An algorithm for solving the problem with the computer is normally developed. This first step normally decides whether the problem is soluble at all. Especially for high tech problems lots of creativity is needed there. One may call it the brainwork. Secondly, the design or algorithm is transformed into a suitable computer language. The output is a programmed software consisting of lines of code. The second step is normally straightforward. However it can be tedious and extremely manpower consuming. Assuming that errors do not exist, the software development is finished after the second step. Of course this is an unrealistic assumption. Therefore the third step "check and test software" is indispensable. It contains a lot of responsibility. Depending on the requirements of safety this step may vary in workload substantially. In any case it is always straightforward. (Note that the software will be never error free as Goedel's theorem predicts for principle reasons. The output of testing can be at most a low probability of errors.)

In the actual software project the three general process steps had subtasks



define requirements
analyze requirements
design level II
prototype/model
design level III

produce SW

qualify and certify system

Figure 1.2

or different names (see fig. 1.2). The subdivision of the first step stresses the

fact that the brainwork is done there. Furthermore, errors in the first step will produce double work in all following steps.

As one can see from fig. 1.1 the whole process is triggered or controlled by the general requirements (e.g. safety) and a given functionality. They are marked by the controlling variables C1 and C2, respectively. Normally the demanded functionality is so big that the software can't be developed in one step. The job would be too complex. The number of design and coding errors would be very big. Therefore a given functionality f is normally cut into N pieces:

$$f = f_1 + f_2 + f_3 + ... + f_{n-1} + f_n + f_{n+1} + ... + f_N$$

The software is developed in N steps. Each steps increases the functionality by f_i. The development cycle to go from the functionality $f_1 + f_2 + f_3 + ... + f_{n-1}$ to $f_1 + f_2 + f_3 + ... + f_{n-1} + f_n$ may be called a phase. In this case phase n. The choice of the number of phases is normally done in order to minimize the total effort. That the total effort varies is due to the fact that the effort for design grows typically exponentially by the functionality increase f_i. The coding effort is about linear in f_i, and the effort for checking and testing is about constant for any f_i. The total effort E takes the form

$$E = E_0 * 3 \ (\exp(a*f_i) + b*f_i + c)$$

where the sum runs from i = 1 to i = N. (The E_0, a, b, and c are constants depending on the problem.) This function has a minimum for a certain N.

Normally the problem can't be described exactly by mathematics. Therefore some guessing and experience is necessary to set the optimal number of phases N. In the software project discussed here five phases had been chosen in the beginning. During the project some phases had been cut into subphases, say phase 2 was divided into subphase 2A and 2B. Towards the end of the project phases were merged in order to reach the deadline for the software completion.

1.3 The project organization

The project organization of the software project was complicated and showed all signs of "natural growth". The details are not worth to be discussed here. A pretty much schematic view is given in fig. 1.3. There you see that the

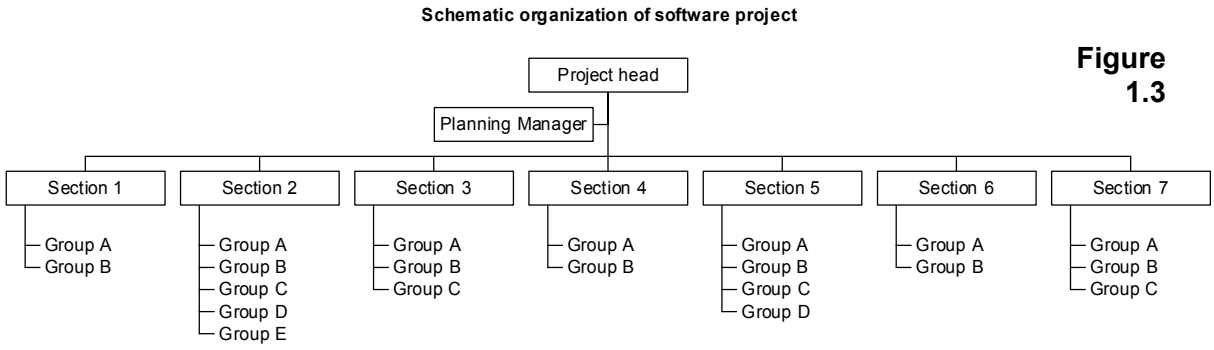


Figure 1.3

project was divided into sections, while each section has a varying amount of groups. In fig. 1.1 one sees three process steps. A process organization should have had three sections with varying numbers of groups (=subactivities) The sections weren't designed in accordance to the subactivities of fig. 1.2 either. For example one section worked for "design software" and "check and test software". However, this anti process structure

was next to impossible to overcome. This was especially due to the fact that the project team consisted of people from four different countries and five different companies. There was a lot of politics and national pride involved. This situation is typical for a project team, though it is extreme here. Quite often a project team consists of people from different divisions of a company and maybe different locations. Similarly, local policy may force unwanted structures. One should avoid them from the very beginning. As part of the general project agreement the project head should have unlimited power to organize as he or she pleases. It was not the case in the software project here. As already said, this unnecessary complication couldn't be removed when our management consulting project was called in later.

Prior to the start of our consulting project each section and/or group had its tasks to do in order to reach the common goal. The details for the work were discussed in a weekly meeting of the section heads. In this way the project was working for round about ten years when our consulting project started. It was far behind schedule. Computer simulations of the status quo showed the probable completion somewhere between 2005 and 2010. Today the probably successful end of the project will be 2001 or 2002. (After we have installed a rigorous project management.) The first planning assumed 1995! However, this final delay of six or seven years was only partly due to insufficient project management. About half of it came from the fact the principle software design was far from being straightforward. As mentioned in chapter 1.1 nobody could predict beforehand whether the total functionality can be reached at all. From this an underestimate of workload is easily possible and unavoidable. This is true for most projects though it maybe

extreme in this case. A reasonable project planning should take such facts into account. Especially one should not promise anything which maybe impossible for principle reasons.

Was there any project planning going on? Yes there was. For each group there existed a project plan in MS–Project. However nobody in the software project team ever looked at it. Normally the group leaders and/or section leaders made a detailed project plan. (The detailed plan was called level 3 plan, a condensed version level 2 plan. A level naught plan was designed for the top management.) The individual plans were not connected with one another. However the actual work of an individual group or section was very much linked to the work of fellow groups or sections. In other words these individual plans were worthless for any kind of project management. The Planning Manager (cf. Fig. 1.3) collected the individual plans and tried to create a total plan. In order to limit his effort he condensed the individual plan. For this reason the final document was called "Total Level 2 Plan". He also compressed the duration for individual tasks in the plan. He did it in order to reach the final deadlines given by the top management. The final document pleased the top management because the promises were fulfilled. From this it is clear why nobody in the team ever looked at the planning. It was a purely political document. Such unrealistic planning is rarely found in other projects. However, some of the fatal mistakes occur quite often. Here it is the role of the Planning Manager. He had the final authority to plan. Quite often planning and actual working in the project are done by different people. If this is the case, nobody will be managed by the plan. As I will show later, planning is normally the first step of the development work. It can't be seen

separately. Because it is the first step in development it is very important to do it as careful as possible. Any error occurring there will cause further errors and double work in following steps.

1.4 Desired approach

The last chapter made the shortcoming of the software project clear. There was no project management. Needless to say, the goal of our consulting project was to install a proper project management. The corner stone of project management is proper project planning. Therefore planning is the core of this thesis. With a proper project plan in your hands project management reduces to a controlling process. It will be discussed only briefly in the present thesis (cf. chap. 2.5 or part 3).

Two "constitutional laws" about planning can be set:

a) It must be realistic. That is everybody of the project team must agree upon that the goals of the planning will be reach.
b) The planning must be a mirror image of the daily work. All tasks done can be found in the planning. Everybody can consult the plan in order to see whether his or her work of the past week is according to schedule. He or she can also see what to do next week.

From these "constitutional laws" details of the planning process can be derived. Details will be given in part 2.

To close this chapter I will give a few remarks that planning must be process based. The work of almost any project can be displayed as a process. In the case of software development the top level of the process is given in fig. 1.1. Of course a more thorough consideration is needed for practical purposes. This is to say that subactivities must be included. The proper process description for the present project may contain a couple of hundred boxes. To develop such a process display was actually part of our consulting project. However, it is not part of this thesis. A process is nothing more than a display of the general tasks with their dependence upon one another. If one just adds the time when a particular task should be done, one would end up with a PERT chart of a project plan. From this it becomes clear: A process map is a prerequisite for planning. For projects where something is developed a further remark is important. In development the true output is not a physical product. It is a description of something. The main thing is normally to find out how things must be done. This is in the simplest case a list of tasks. A project plan which is arbitrarily fine is already a description of what to do and how. Therefore a project plan for a development project is not outside the actual development work. It is the development work. The difference of the project plan and the finally developed product is just the degree of detail. From this it is clear who must do the planning: The person who develops.

1.5 Choice of software tool

From a theoretical point of view it is not necessary to have any software tool for project planning. In practice it will be unavoidable in most cases. Having

about 10,000 activities in the present software project an IT support is mandatory. There are a great number of project planning tools on the market. I will not discuss their pros and cons here. In the software project under consideration Microsoft (MS)–Project 98 was chosen. It has its clear drawbacks. First of all, the software contains quite a few bugs. (A few of them are discussed in 2.6.1.) Furthermore many almost trivial features are missing. (E.g. any form of calculation. It is not possible to have say a task x which has the duration of task y divided by 5.) Times for automatic re–planning of MS–Project 98 can be incredible long for complicated projects. (In the present software project a shift of one task can mean an automatic shift in many of the other 10,000 tasks. Such operation took ten to fifteen minutes even with a 200 MHz Pentium processor. A super PC with two 400 MHz processors brought the time hardly under ten minutes.) From all this any planning expert would not choose MS–Project 98. However there are two factors of MS–Project which are unbeatable:

i) It is very common.

ii) It is relatively easy to use.

From i) one can conclude that many would-be planners have some experience with MS–Project. ii) implies that it is easy to learn and that the use of the tool itself won't consume too much time. Both factors are essential if the engineer in charge of development is also in charge of planning. As discussed in the previous chapter it was the case in the present software project, and it was also a critical success factor. Quite often one finds the situation that planning is done by a separate planning department. In such cases other tools can be recommended. However such organization bears

the danger that the planning will never be used as the necessary project management tool.

2. The Planning Manual for the engineers

In the second part a handbook for the planning process is given. It contains a hands–on description how to plan the project by using MS–Project 98. It will also give some hints about controlling the project.

2.1 The process

Before starting with the program one has to think about the process. That is one has to be sure who is doing what. In order to get a clear picture about it one has to write down the process pattern in a well–defined way.

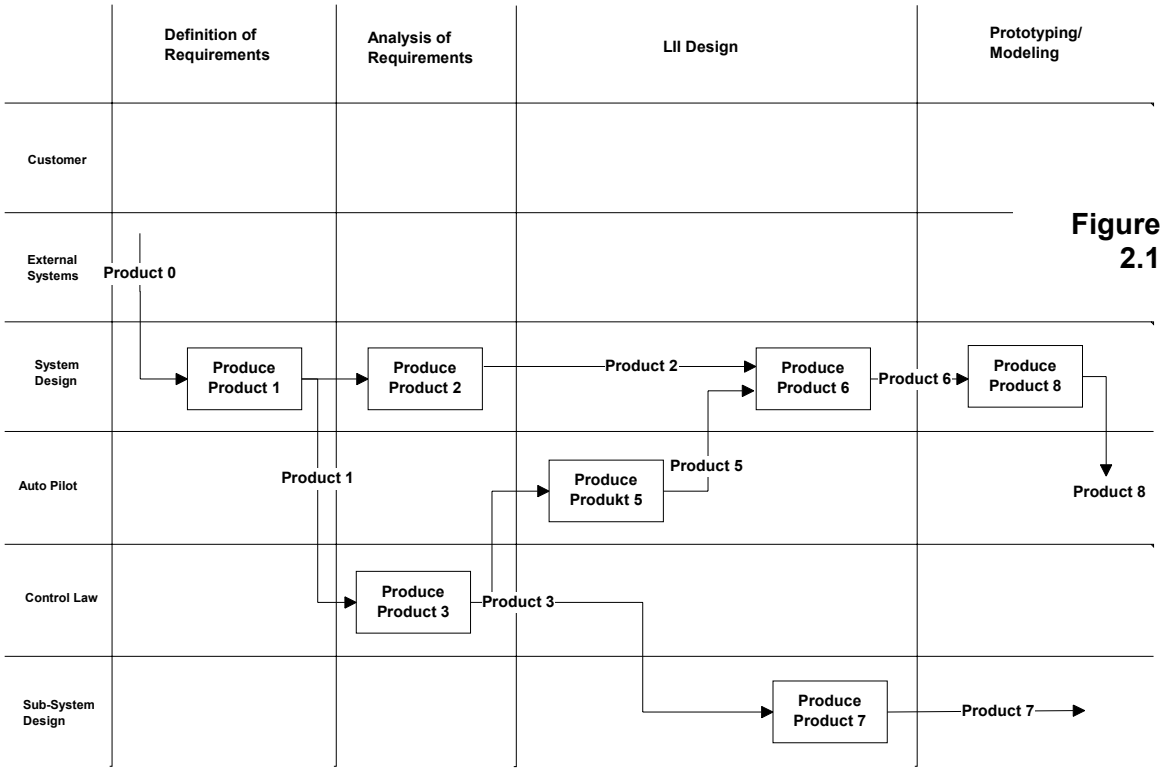


Figure 2.1

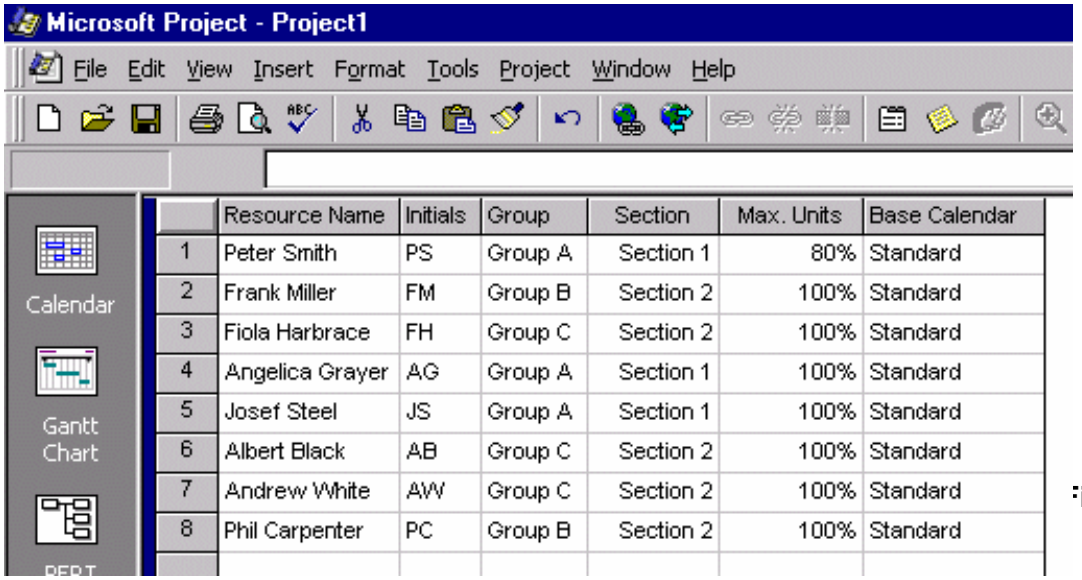
The process should consist of a set of rectangular boxes connected by lines with arrows. Each box should contain the task to be done. Needless to say it'll start with a verb. The output of each box will be a product. The name of each product should be written to the line leaving each box (see fig. 2.1). Every box must have input and output. Exceptions are the first and last boxes. They will show either outputs or inputs, respectively.
The whole process network should be written on a grid. Each horizontal line contains boxes with activities of one group/section or external partners. The lines are ordered as follows: Customer, External Systems, System Design, Auto Pilot, Control Laws, Subsystem Design, Air Data System, SW Design, Flight Mechanics, System Development, EPCs, Safety, Qualification & Certification, Rig Test, Flight Test, EDR, HW Qualification. The vertical lines denote roughly the process steps. They are ordered as follows: Definition of Requirements, Analysis of Requirements, L II Design, Prototyping/Modeling, L III Design, SW Production System Qualification and Certification.

Almost all level II activities should be displayed on the process map. The software tool "Process Guide" should be used. It supports the above stated format. If a box contains important subtasks, these should be included as an underlying process in the corresponding box. (These boxes will show a shadow. By clicking them one can edit the subprocess)

There are further rules for mapping the process. For example every box should have an underlying text describing the task verbally. However, these are not necessary for planning. It will become necessary for other purposes (e.g. training). The complete set of rules will be given elsewhere.

2.2 The resource pool

In order to have an overview over all available resources the entire team needs one resource pool. It is a separate MS Project file containing all the names of the team.



	Resource Name	Initials	Group	Section	Max. Units	Base Calendar
1	Peter Smith	PS	Group A	Section 1	80%	Standard
2	Frank Miller	FM	Group B	Section 2	100%	Standard
3	Fiola Harbrace	FH	Group C	Section 2	100%	Standard
4	Angelica Grayer	AG	Group A	Section 1	100%	Standard
5	Josef Steel	JS	Group A	Section 1	100%	Standard
6	Albert Black	AB	Group C	Section 2	100%	Standard
7	Andrew White	AW	Group C	Section 2	100%	Standard
8	Phil Carpenter	PC	Group B	Section 2	100%	Standard

Figure 2.2

(see fig. 2.2) It will also show the individuals initials, the group, and the section they belong to. Unlike the picture in fig. 2.2, the names should be ordered by group and section. A later reordering can be problematic. The column “Max. Units” will contain their maximum capacity. Normally it will be set to “100 %”. However, e.g. section leaders might have other responsibilities. Their capacity for project tasks may be significantly lower. Allowing for special blackout dates (e.g. holidays) one can assign a non–working time for each individual. This is done by clicking the corresponding line and choosing “working time” (see fig. 2.3). In the present example line 5

is clicked and a holiday period is assigned for Josef Steel from October 5 till October 23.

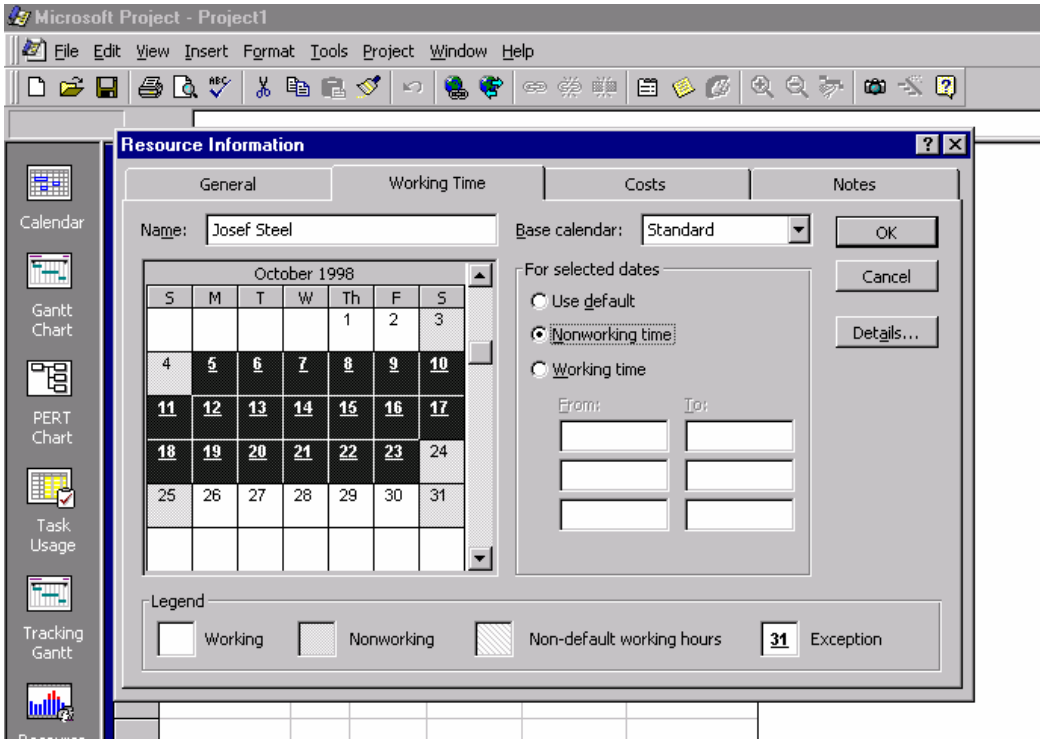


Figure 2.3

Such black out dates should be set long in advance (at least 6 months ahead). For longer periods holidays must be assigned statistically. Instead of 2.5 black out days per months for everybody or 12 % less capacity, one should use a typical picture from former years. For example most vacation occurs through the summer months and over the winter holiday season. Assigning non–working time means that these dates are blocked for the corresponding resource (like Saturdays and Sundays for everybody). Instead of assigning a capacity below 100 % (see above) one can also assign less working time. Note that a limited capacity just means that the resource is overallocated earlier. In contrast assigning non–working time will postpone the task.

The planning manager is responsible for creating the resource pool file. The group managers are responsible for delivering proper input about planned vacations, capacities, etc. They are responsible for its content.

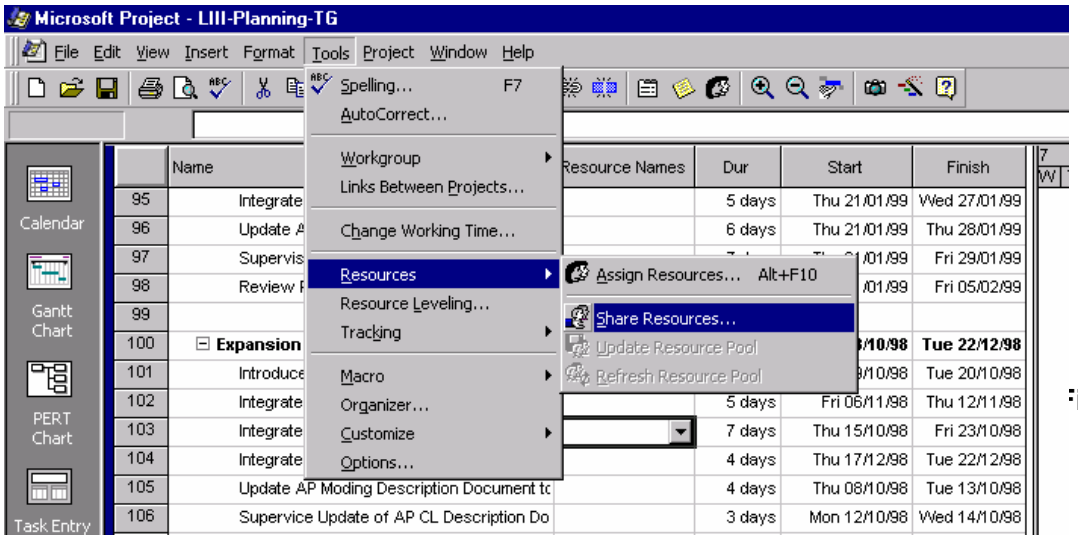


Figure 2.4

The resource pool file must be saved with read/write privilege for every planner. However, the individual planner must not open the resource pool file directly. It will automatically open as “read only” by opening an individual planning file. Before this automation works one has to link the individual planning file to the resource pool. Opening the resource pool is the first step

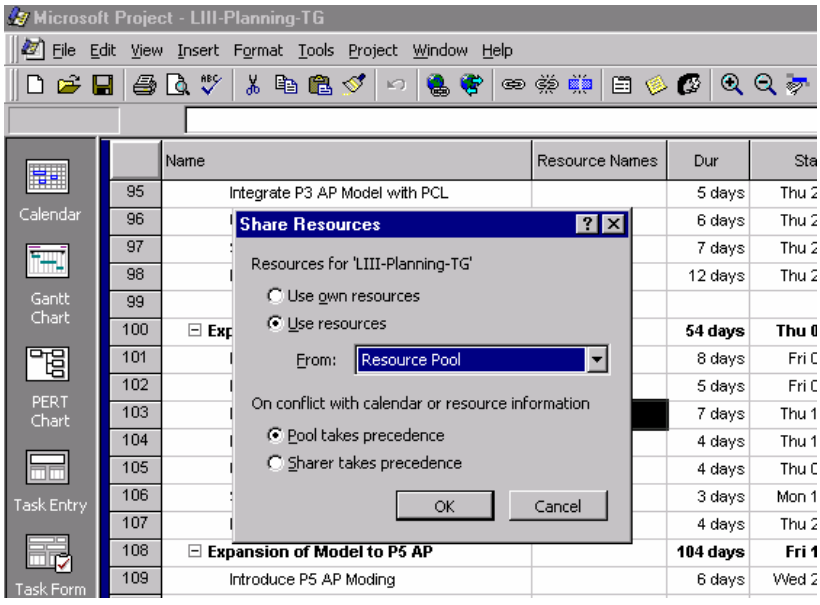


Figure 2.5

(this is the only time when it is allowed, see above). Then one has to open the individual planning file in a second window. Choose “Tools”, “Resources”, “Share Resources…” from the menu (see fig. 2.4). In doing so one will have a choice to assign all presently open files as resource pool. Of course, one has to choose the file “Resource Pool” here (see fig. 2.5). Now one can close both files. The next time the individual planning file is opened the resource pool will open as “read only“ simultaneously. One can assign resources to every task now. The choice of all possible resources of the pool will appear automatically (see also 2.3.3).

2.3 The program

“The program“ is defined as the list of tasks, their links, the assigned workload,... I.e. all one needs for proper planning. “The plan“ is defined as the list of tasks only.

In what follows chapter 2.3.1 will give an overview of how the plan (=list of tasks) should be created. Chapters 2.3.2 and 2.3.3 will give the rules how the plan must be transformed into the program.

2.3.1 The plan

The draft version of the plan should follow from the process (cf. chap. 1). The plan should start with a list of “Input Dependencies“. This is a list of products

one needs from other groups or external partners. They are the nouns on the lines coming in from some other group. After that a list of “Output Dependencies“ should follow. These are the products from one’s own group being delivered to another group or external partner. Taking the group System Design from fig. 2.1 the list of input and output dependencies looks as follows:

– **Input Dependencies**	– **Output Dependencies**
Product 0 (from External Systems)	Product 1
Product 5 (from Auto Pilot)	Product 8

An output should show up as input in another group and vice versa. In order to make linking (see below) easier one should denote where the input dependencies are coming from (e.g. “External Systems” and “Auto Pilot” in the example above).

All input/output dependencies are milestones. They have duration zero. They will be marked as milestones. They are the milestones of each program.

After the dependencies the list of tasks or activities must follow. Note that tasks are always verbal. I.e. they are starting with a verb. First all activities from that part of the process concerning the own group must be written down. Taking again the group System Design from fig. 2.1 the list looks as follows:

+ **Produce product 1**

+ **Produce Product 2**

+ **Produce Product 6**

+ **Produce Product 8**

In general they will need some subactivities. It is indicated by the little + signs in front of each activity of the above stated list. A length of an activity should be of such to be controlled weekly (cf. chapter 5). I.e. its length must be of order one to three weeks. For a better overview one should create a hierarchy of sum activities. Normally, there will be three to four levels of hierarchy.

Note however that the detailed planning period should stretch over nine months (or a period close to it where a particular phase is ending). After that a rougher planning period with longer activity lengths will be sufficient. Every three months there should be an update. I.e. three months of the rough planning period must be transformed into detailed planning. This assures that a period of six to nine months is always planned in detail.

The entire list of tasks should stretch over all phases. In many cases it might be ordered by phases. However, this is not always reasonable and therefore not mandatory. In order to filter separate phases a mark for the phase name should be added. For this purpose several text columns must be added. A “yes” in the corresponding column means that this task (or dependency) belongs to the corresponding phase. The following columns should be chosen:

Text 2	for	Phase 2B2
Text 4	for	Phase 3
Text 5	for	Phase 3A
Text 6	for	Phase 4
Text 7	for	Phase 5

Note that it may be reasonable to assign a “yes” in more than one column to indicate that this item belongs to several phases. This will be typical for headlines or sum activities

The new level 2 plan will contain all information of the individual level 3 plans. In order to make filtering easier the author of each level 3 plan should decide whether an activity is considered level 2 or 3. To indicate his or her choice the column Text 8 should be added. A “yes” in this column means that the task should be considered level 2.

2.3.2 The links

After the list of tasks is created the tasks must be linked. The input dependencies are milestones (duration 0). An assumed but realistic date must be assigned to them (assign the constrain “Must Start On”). It should be discussed with the delivering fellow group beforehand. Output dependencies are milestones (duration 0), and will also have a fixed date (assign constraint “Must Start On”). Normally the level 0 program or demands of other groups

will determine it. The tasks themselves are all marked as “as soon as possible“. Their duration will be set later. For the time being the default set of 1 day will be fine.

After these settings have been performed, the linking can start. Every input dependency will get at least one successor and no predecessor. Every output dependency will get at least one predecessor and no successor. The tasks themselves must all be linked with successors and predecessors. Note that a successor must always have a later date then its predecessor. If this causality is violated MS Project will not execute the link. Marking the two tasks to be linked and pressing the linking symbol can do the linking. As an alternative one can fill in the proper line number in the column predecessor or successor. With the setting described above MS Project should always execute the linking. This is because no duration has been set up to now. However, if an output dependency isn’t dated sufficiently later than its corresponding input dependency a violation of causality may occur. In this case the given dates for input/output dependencies must be questioned.

There is one small class of activities that are excluded from most of the rules set. They are e.g. “Prepare air show in ...“. These activities may or may not have links inside their class. But they will not have any links to the “normal“ activities described above. For them a fixed duration and start date must be set. Of course, a resource must be assigned to them. This ensures that all workloads are taken into account.

2.3.3 Workload

At the end of chapter 2.3.2 all activities are linked. Now a duration, workload, and resource must be assigned. These things are all connected. Therefore one has to deal with them simultaneously. First one has to assign duration for

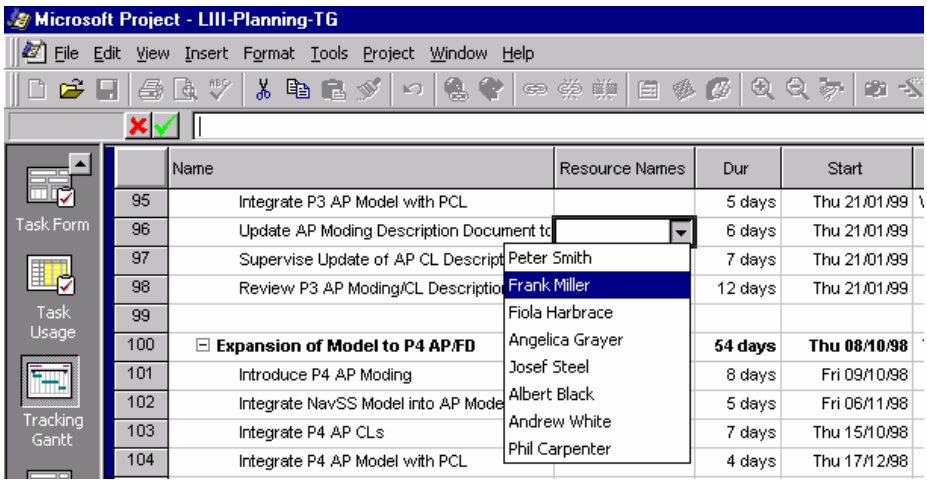


Figure 2.6

each task and one or more proper resources. By clicking the resource field all possible names will appear (see fig. 2.6). Any resource outside the resource pool must not be assigned. For technical reasons we will start without a common resource pool. I.e. every group leader may choose his or her own resources for the time being. Normally a name from the planning group will be chosen. However, an assignment of anybody may be suitable. Note that choosing a resource from your colleague's group should be done after a discussion with your colleague only. It will be allowed to assign group resources (e.g. "group xy"), if one has no special person in mind. Every group leader is responsible to transform group resources into a named resource (e.g. Group A[50%] must be transformed into say JS[50%]). Assigning group resources may be reasonable for the entire time before the linking. Because

after linking activities will be shifted and a reassignment will become necessary. Assigning group resources will avoid most of the tedious job. However one should make sure that the capacity is available. A sufficient capacity of group resource does not necessarily imply a sufficient capacity of resources in reality. This is because not everybody can do every job. The individual planner should decide what is necessary. After linking named resources should be assigned for a period of two months (with an update every month).

For each activity one has to decide whether it is of the type “Fixed Duration“ or “Fixed Work“. Fixed duration means if something changes (e.g. doubling of resources) that it won’t change its duration, but other parameters such as workload. On the other hand fixed work means that the workload will stay constant. That is changing the duration will demand more or less capacity of

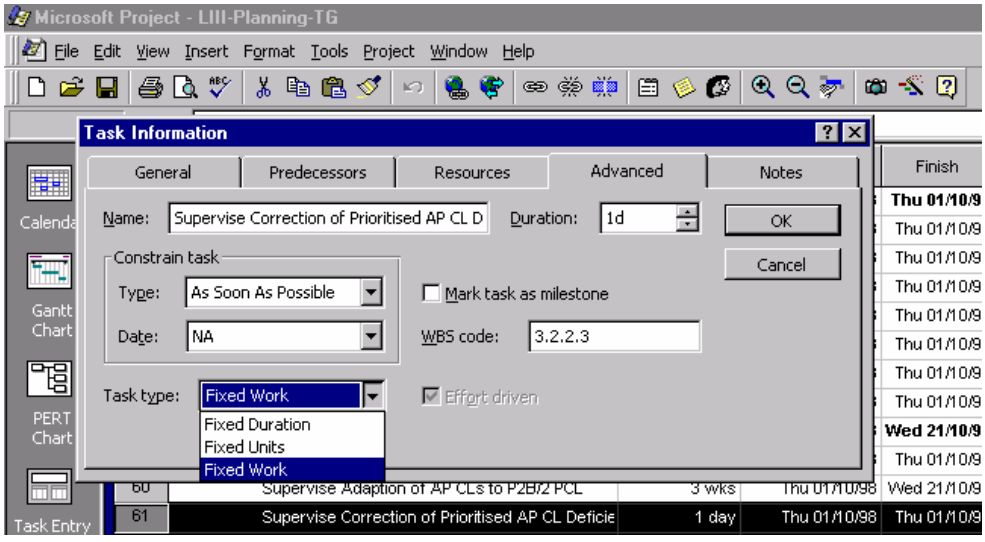


Figure 2.7

the resources assigned. A realistic decision has to be made by the planner. To set this switch in MS Project one has to double click the activity and

choose “Advanced“ (see figure 2.7). The checkmark in front of “Effort driven” should always be there.

The art of estimating the proper duration and/or workload for every activity is called “metrication“. For an overview see fig. 2.8. All methods are based on trial and error. With no historic data at hand one has to make a guess. For a more detailed overview please see appendix 2.6.2. Note that it will become significantly easier to find an “effort driver“ for each activity, if one subdivides tasks.

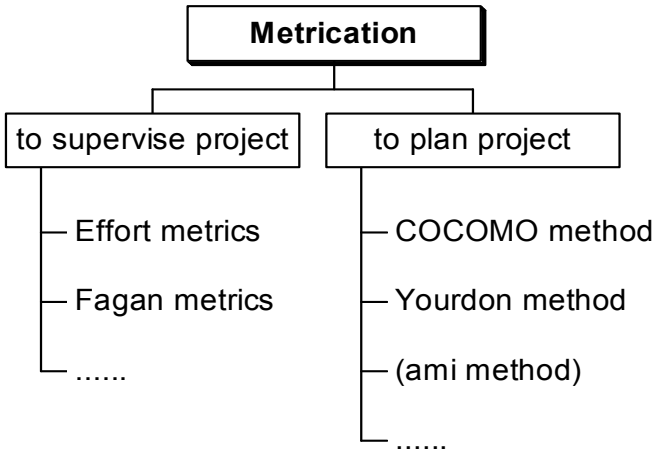


Figure 2.8

Assigning duration and workload for every activity in a linked program can be tedious, but it is absolutely necessary. Because one has set fixed dates for the input dependencies and output dependencies the sum over all durations on the critical path will be limited. So one may be forced to go back and forth by assigning duration and workload. Of course the entire plan must remain realistic. Realistic means that the program is doable as described. Resources and inputs must be available. The staff must accept this. If simple “re–planning“ does not work three steps are suggested:

a) Make assumptions.

b) Try to get input dependencies earlier.

c) Talk to the management about postponing output dependencies.

Point a) means the following: Is it possible to make proper assumption to reach the goal? E.g. to assume more resources or the acquisition of a new tool, etc. Such measures should always be discussed with the appropriate management. If an assumption is made, one has to take proper steps and measures to assure that it will become reality. An additional resource must be noted as NN_i. It must be added to the resource pool. Point b) means that one should talk to the appropriate fellow group or the external partner. Such talks are always difficult. Nevertheless every chance should be taken. Point c) is the last escape. Maybe it is possible to skip a few functions temporarily in order to reach the milestone. Of course the functions must be delivered later.

After all tasks have resources and duration the workload must be adjusted. I.e. one has to check the allocation of workload of every group member. If a resource is overallocated, re–planning and/or reassigning must be performed.

If one finally gets through the entire above then one is almost ready. As a last step one should make the output dependencies free floating. I.e. just mark them as “as soon as possible” like the rest of the activities. Now everything is done and the file can be merged as described under chapter 4. However, proper checking is highly recommended.

3.4 Checking

In this chapter two checklists are given: one for the plan and one for the program.

Checklist for the plan:

- Are all activities from the process also in the plan?
- Are the input/output dependencies products from the process graph?
- Is there an indication where the input dependencies are supposed to come from?
- Are all activities formulated verbally (i.e. starting with a verb)?
- Is every necessary activity in the plan? (E.g. support other groups, prepare air shows,…)
- Did one take into account the backlog? (I.e. delayed activities from former phases)
- Is the corresponding phase name indicated at each activity?
- Is there an indicator for Level 2 (“yes” in column “Text 8”)?

Checklist for the program:

- Are all activities linked?
- Have all input/output dependencies the duration zero?
- Are all activities and the output dependencies marked as “as soon as possible”?

- Is the duration of every activity in the order of one to three weeks?
- Is a resource assigned to every activity?
- Are there any overallocations of work to any resources of the group? (One can check it by switching to “Resource Graph” on the l.h.s. menu bar)
- Are sum activities pure sum activities? (I.e. they are not linked and no resources are assigned)
- Is the critical path reasonable? Do delayed tasks delay the output dependencies correspondingly?

2.4 Merging the projects

After all individual group plans are produced they can be merged into one project file. The planning manager and not the individual group planner will do this. Nevertheless everybody should be interested in the general way in which it is done. First one has to open an empty project with proper settings

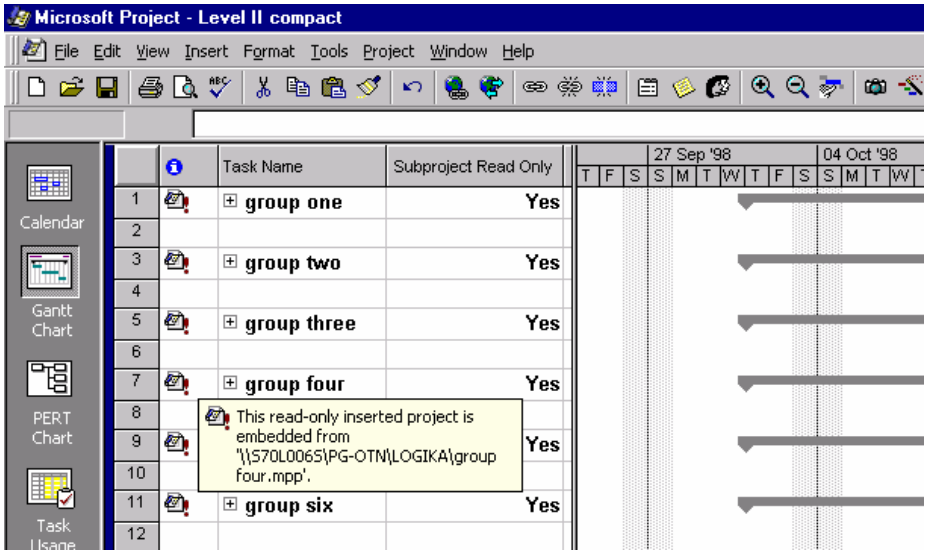


Figure 2.9

for the calendar, etc. By choosing “Insert” and then “Project…” from the menu one can insert the individual plans easily. In doing so a picture like in figure 2.9 will appear. Note that the names of the inserted

project may have funny names by default. This is some bug in MS Project (cf. appendix 2.6.1). However the inserted subproject can be renamed easily. If one clicks the “+” sign in front of the subproject it will open in full detail. Of course any kind of filter can be applied. Figure 2.9 also shows a column “Subproject Read Only”. This column should be inserted. The switch should be set to “Yes” at almost all times. This ensures that the underlying level 3 plans will be assessed as “read only”. Only if links must be modified should the switch be turned to “No”. (Note that the level 2 planning manager must have read/write access to all group files. The group level planners must have read/write privilege to their own files only and “read only” access to all other files including the level 2 plan. Everybody in the FJT must have read access to every planning file!)

To link the files all “Subproject Read Only” switches must be turned to “No”. Note that this will block every group manager from editing and saving his or her file as long as the level 2 file is open. After that all the group plans must be opened by pressing the “+” sign in front of each subproject. Now “Filter for: Milestones” from the Menu “Project” should be applied. In doing so only the input and output dependencies will be displayed. They are the only ones needed for linking. No activities may be touched or changed by the planning manager. Any changes there belong to the planning group only. The linking

itself is done in the same manner as in an ordinary project file. The easiest way is by marking two activities and using the linking symbol. The correct rooting between the files will show up in the corresponding “Predecessor” or “Successor” column automatically. Step by step the linking can start now. A pair of input dependency and output dependency of two groups must be linked. One has to make sure that one will identify the right links. Most likely the individual planning managers must be called several times. There is also a good chance that the input/output dependencies won’t fit to each other. I. e. an output is later than the corresponding input. In this case the level 2 planning manager has to make sure that there is no trivial mistake. He should talk to the individual planning managers. In the end he has to connect any input/output dependency. After or shortly before connecting input with output the input dependency must also be switched to “as soon as possible”.

After the last link has been set everything should be marked as “as soon as possible” except for the input dependencies from external partners. The task of linking ends by closing the file and saving all subprojects. After this is done the file should be opened again immediately. Then the “Subproject Read Only” switch must be turned to “Yes” again. Closing and saving it again assures that every future opening won’t block any group manager.

After this last task of merging is finished a standard check can be performed. Two areas should be checked in order to explore whether the project is manageable.

i) Are there any delays in any critical output dependencies (customer milestones)? If yes, a detailed report should be given (how long, cause,…).

ii) Are there any overallocated resources? If yes, a detailed report should be given (who, how much,…).

Now the project steering can start. Details must be given elsewhere. Some statements will be found in chapter 2.5

2.5 Using the program

Once all programs are ready and merged as described under chapter 2.4 the management and all participants must agree upon it. If this is the case the entire program must be stored with a baseline. (Don't do this before this time with any of the level 3 or level 2 programs) To do it, open the level 2 plan and all subprojects. Turn the "Subproject Read Only" switch to "No". Choose "Save As…" and click "Save '…' with a baseline" for all projects.

Now the actual project management can start. I.e. one may harvest the fruits of the efforts described above. The first thing is that the program should be everybody's working guide. E.g. Monday morning everybody will see his or her task by watching the program online. Because all tasks are in the plan, nobody is allowed to do anything else. He or she will be told what to do and until when. At the end of the week the group manager has to fill in the status. This can be done by inserting a column "% Complete", and filling in a number

between 0 % and 100 %. If everything is on track and nothing else has happened, no further action is necessary. However, a delay in completion may indicate that further delays will occur. For certain tasks it may be impossible to be completed in time. If that is the case re–planning is mandatory. I.e. a postponed task will require shortening the duration for future tasks or restructuring the whole program. If only one's own group is concerned the whole thing should be relatively easy. In order to see any effect on other groups and important milestones one should also look into the level 2 program and do the re–planning there. Note that the group manager has read only access to the level 2 program. Of course the whole action is only taken to simulate the effect on the entire project rather than to make unauthorized changes in the program. If this simulation leads to the conclusion that re–planning on group level does not cause serious harm to anybody else, the group manager may proceed. Otherwise he or she has to discuss further measures with fellow groups and the management.

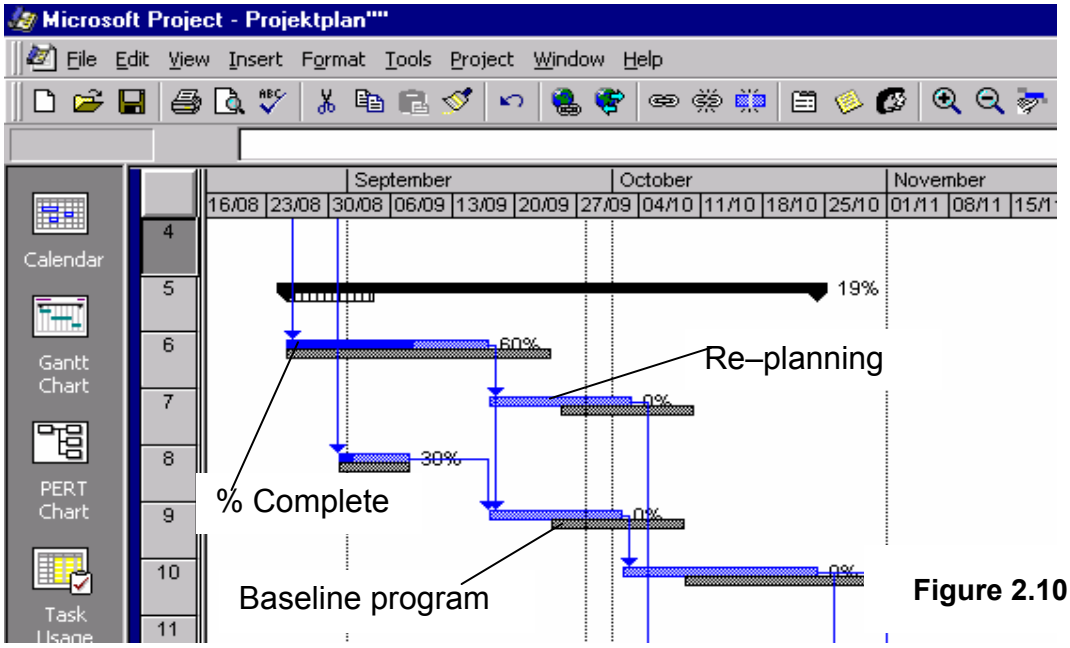


Figure 2.10

The whole reason behind the procedure described in the last paragraph is to realize delays as early as possible. With the described weekly review the cause of a potential delay can be at most a couple of days old. Under such circumstances fixing is almost certainly possible.

A hopefully typical situation is displayed in figure 2.10. There one can see a positive deviation from the baseline program. Because task number six made good progress, its completion was estimated to happen earlier. With this re–planning all following tasks can be executed earlier (under the assumption that the resources are available, see below). Of course no further action is required in this case.

Another useful tool to steer the project is the resource usage evaluation. There are two typical views to choose from. First to mention is the “Resource Graph”. Here bars display the resource usage over time. Their heights show the % of usage. It gives a good overview whether or not a particular resource is over- or underallocated over some period of time. For a more detailed analysis “Resource Usage” should be chosen. Here a calendar is displayed. In each row the work in hours of a resource is displayed for each day, week, month or whatever. By clicking the “+” in front of each resource all assigned tasks will be displayed. By watching these two views closely two important questions can be answered: Are there any overallocated resources which will cause delays in the future? Where can one find an underallocated resource to perform a job? However, there is a note of caution. To be sure to get the correct information about resource allocation one should always look into the level 2 program with all subprojects open (cf. appendix 2.6.1)).

2.6 Appendix Part 2

2.6.1 Bugs in MS–Project 98

In this chapter a few hints about possible shortcomings of MS–Project 98 are given. The first note is about the copy and paste function. Used in the simple manner it works as perfect as in all Microsoft programs. If one chooses “Paste Special…” and there “Paste Link” some undesired effects may occur. If one copies one or more tasks and adds or deletes tasks in the source file very undesired results may show up. Furthermore the numbers of the predecessors and successors are not transferred correctly. Due to the memory effect (see below) links abandoned long ago may cause fatal errors. Because of all that “Paste Special…” and “Paste Link” must not be used here.

As all Microsoft programs MS–Project shows a memory effect, which can lead to very undesired side effects. Memory effect means that the underlying code contains information deleted long ago on the user surface. E.g. one uses the function “Paste Special…” and “Paste Link”. If one later deletes the link on the user surface some remains will stay in the underlying code of the source file. They are undeletable by using MS–Project. However, these remains can cause serious problems when one tries to insert the file into another as described in chapter 2.4. Similar effects do exist with resources that had been assigned and were deleted at some time in the past. To avoid such problems one should proceed as follows. Firstly, open the old and

possibly “infected” file. Secondly, choose “File” and “New”. Thirdly, copy all lines of the file and insert it in the new file by simple “Paste” (no “Paste Special…” and “Paste Link”). The new file can be stored under a proper name. Note that the source and target fields must have the same format for copying. I.e. both files must have the same columns. Note further that only almost every information is copied by the described procedure. E.g. information in the calendar is not transferred.

Some warning must be given about the shared resources. As described in the previous chapters all programs share one resource pool. It is automatically opened (as read only) if one individual file is opened. Therefore one should be able to evaluate the resource allocation from each individual file. However, this function does work fine sometimes but not always. To be on the safe side one has to open all participating files simultaneously. Looking into the level 2 program (cf. suggestion in chap. 2.5) does this most easily.

Another shortcoming in MS–Project is its PERT chart. It usually leads to graphically awkward networks. In addition they are sometimes wrong. In the case of inserted subprojects the PERT charts are oversimplified. In order to get proper PERT charts one has to use another tool. (e.g. Artemis).

A last warning must be given about using MS–Project too automatically. If one uses e.g. “Resource Leveling…”, one will end up with a proper resource allocation. However, in general all milestones are at a completely different position. Because the picture looks so different one has no chance to find the

spot, which caused the delays. Generally speaking this function is completely useless for any complex program. A similar wisdom applies if someone makes several changes in a row in a highly linked program. The final changes may be dramatic. One will not find out what caused which delay. Therefore any changes in the program should be done step by step. After each step the possible changes on all other programs must be monitored. The best way to simulate the outcome is by using the level 2 program. If one has found a proper way eventually, one can go back and install these changes in the particular subprogram.

2.6.2 Metrication

Metrication is the art of estimating and judging the effort required for a particular work package in a project. In this chapter a few words of explanation are given to the methods of figure 2.8.

The effort metrics is a method to judge about progress of a running project rather than a planning tool. There are two main quantities that should be watched carefully:

$$\%\text{Complete} = \frac{\text{Effort of completed}}{\text{Current estimate to}}$$

$$\%\text{Effort} = \frac{\text{Effort of completed}}{\text{Original planned effort to complete}}$$

100 % minus %Complete gives the amount of work or effort still needed to complete the project under the latest available estimate. %Effort tells you about how much work or effort of the originally planned has been done or spent. Of course both quantities can be used for an entire project, a subproject or an individual task. They are the main indicators for the status quo. One suggested view graph is a plot of %Complete over %Effort. Assuming the same scale on both axes a 45° line means that the project runs exactly as planned. A lesser slope means that the effort for completion starts to become bigger compared to the plan. A bigger slope indicates less effort compared to plan. By carefully watching this graph one has a perfect early warning system about over– or underestimation of effort.

Another method to judge about the project quality is the Fagan metrics. It is a statistical judgement about the performance. Needless to say one needs a database from an earlier project. E.g. one can gather data from former projects about at which stage an error originated and at which (later) stage it was observed. Maybe one finds that the average error done in L II Design is observed most likely in L III Design with a Gaussian distribution toward the adjacent stages such as “Prototyping/Modeling” and “SW Production System Qualification and Certification”. If this is a fact from former projects, one can easily judge whether error detection is earlier or later in the present project. Another statistical evaluation is a histogram of average actions per review or average pages per review in former project. Again one will get something close to a Gaussian or maybe Poisson distribution. Comparing the data of the present project with these distributions one can judge whether the review

process was not thorough enough or the work of poor quality. For more information about the Fagan method one may consult Ref. 2.1.

The COCOMO method is a procedure to estimate the workload for a particular task or a subproject. If one plots the effort for a particular part in former projects over the project size one will get a more or less exponential dependence. Of course other mathematical functions are possible, though theoretical models suggest an exponential dependence. Be it as it may one will get a benchmark curve for the effort of future projects. This method is used very successfully in software production. However other areas are not excluded. For further reference please see Ref.2.2.

Mathematically similar to the COCOMO method in how it works is the Yourdon method. It is also based on former project data. It takes the effort there as a base. Then one has to judge how much additional effort the additional functionality needs. Mathematically speaking if a particular task of a former project consumed the workload w_0 and yielded the functionality f_0 the present project will need

$$w_1 = w_0 + \Delta w \qquad \text{by a given functionality} \qquad f_1 = f_0 + \Delta f$$

In order to calculate the workload w_1 one now has to estimate Δw for a given functionality Δf. The solution is typically a minor problem. It uses the same idea as scientists do by applying perturbation theory. For more information please see Ref. 2.3.

In connection with metrication the ami method is often cited. Actually it is no direct metrication method. The ami project was part of the ESPRIT program to improve the competitiveness of European companies between 1990 and 1992. The ami improvement process consists of the four steps assess, analyze, metricate, and improve. Details can be found elsewhere (e.g. Ref. 2.4). As all improvement processes should, the ami method contains a measurement part (i.e. to metricate). There it is suggested to find an “effort driver” (e.g. lines in program) for each project part or task. In order to find a linear relationship such as effort = amount of effort driver * specific effort (e.g. workload per program line) one has to break down the task more and more. Eventually one will end up in the linear regime with sufficient accuracy. The procedure is analogous to activity based costing (ABC). There one has to find the cost drivers.

3. Conclusions

The third part of this thesis deals with conclusions. I will tell how the software project went along with its new project management. Of course the difficulties are most interesting. They will lead to the lessons learned. I will close with thoughts about a completely alternative project organization.

3.1 How the software project went along

The goal of our consulting project was to improve the project management of the software project. In particular we had to shorten its duration (cf. chapter 1.4). The main step was a proper planning as described in detail in part 2. To get this planning took longer than we expected. From start to completion it took roughly five months. The main limitation was not our consulting manpower. Three factors contributed to the five months planning period:

a) The quality of the original planning was very poor. We had to start from scratch.
b) The main workload for planning had to be carried by the developing engineers. Because the software project was delayed already they weren't willing to invest "useless" time for planning.
c) The planning was complex. It resulted in difficulties with the software tool (MS–Project) which reached its limitations. Furthermore, the planning itself reached an almost chaotic behavior (cf. ref. 1.2).

Point a) was clearly an underestimate of the status quo. There was even no common agreement how the development process (cf. chapters 2.1 and 1.4) was running. There was absolutely no agreement as to who was using the

output of a particular group or who was doing what. The main surprise was that the software project had gone along for years with at least some results. Point b) was the typical situation of self–amplification. Of course the engineers had to do the planning (cf. end of chapter 1.4). They had no time to do it, because the project was delayed already. The reason for delay was that they had no proper planning... The way out is clear: Take some time for planning. (Easy to say, hard to do!) Like point a), point c) was also an underestimate. The complexity was not due to the fact that the project was big. (250 engineers had eventually planned about 10,000 tasks.) Similar situations are given in other projects without causing complexity. The main reason was the dependence of tasks upon one another. In technical terms: the average number of predecessors and successors of every task. The high number of successors and predecessors was not caused by the great detail of planning. If one plans an ordinary eight hour working day in steps of minutes all 480 tasks may be highly connected. However it does not imply that this particular working day is complex. In most cases the successors and predecessors of the 480 tasks are tasks out of the same set of 480 tasks. Real complexity starts when tasks of different people are connected. Especially when these people are from different groups the intersection must be clearly defined. Exactly this latter situation was given in the software project. These inter–group or inter–section dependencies were essentially not caused by the truly inefficient organization of the software project (cf. chapter 1.3). They were rooted much deeper. In the development of standard products like e.g. cars there may be many people involved, though the development is by no means complex as defined above. In automobile development the whole car is divided into pieces. Each piece depends more

or less weakly on the others. Take for example the fuel injection pump, and the motor block and pistons. Of course the pump must be powerful enough for the number and volume of the pistons. But there won't be much more optimization. In the end the pump will be over engineered for the engine or the power could be increased by a modification of the pump. The engine must be efficient enough to be a valuable product on the market. In high tech military goods the situation is different. Quite often the final frontier of efficiency is met. This is allowed because economic considerations play only a minor role. It is often unavoidable to be superior to the enemy whatever the costs are. From this it becomes clear that complexity is inherent in the software project discussed here. One has to cope with it. This unavoidable complexity brought the software tool to its limits. The bugs in it became really painful (cf. chapter 2.6.1). Independent of the software tool chaotic behavior was almost reached due to the complexity. Minor (incorrect) changes at one end of the project plan caused tremendous confusion in some other far away area. To find the root was extremely tedious. A truly chaotic behavior (cf. ref. 1.2) was finally excluded in the planning of the software project. Chaotic behavior would mean a change of the project finish date by years, if say a single task takes a day longer. In principle such a situation is thinkable in complex projects. (The simplest example is a project where a telescope is built to observe a particular comet which shows up for one day every ten years. If the telescope is ready one day too late, the project takes ten years longer.) In the process of the planning we had chaotic situations. Moving a single task for some days shifted the project end for several months. Luckily this chaotic behavior was only caused by planning mistakes. If it were

chaotic, the project would be unmanageable for principle reasons (like a too unstable airplane, cf. chapter 1.1).

Of course after the planning was completed the fruits of the effort could be harvested. The essential idea of our project management or project controlling was quite simple. Every group had to insert the %complete for every task presently worked on. This 'as is' was compared with the 'to be' of the program. A programmed macro in MS–Project produced an automatic list of delayed tasks, if the 'as is' was behind 'to be'. For every task on the list action steps had to be taken. If re–planning was unavoidable, the group leaders had to check whether it would move the final ending date of the project. Luckily, almost all delayed tasks were not on the critical path. Though the described way of project controlling was quite simple, it took some time to really implement it. At first the engineers felt as if they were under permanent surveillance. Such pressures imply a defensive attitude: Try to hide delays or find somebody else who is guilty for it. One had to convince the engineers that project controlling was something undertaken by themselves and especially for themselves. The first breakthroughs were messages like: "This new project plan helps me to make resource planning realistic." Once it became truly operational the project management helped in two ways:

a) Creating discipline to fulfil the tasks as planned.
b) Make it possible to check for improvements in order to decrease the project's total duration.

Point a) helped that the project end date didn't slipped further as it had done continuously over the years before. However this was not enough for the high expectations of the top management. The stable end date was fine but not

good enough. Therefore one had to undertake improvement measures. Of course everybody in the team had hunted for improvement measures for many years. However, in complex projects as discussed here it is next to impossible to see whether a measure will really improve anything. This is due to the fact the almost any improvement has some side effects which are counterproductive. How can one guess the net effect? Furthermore a net improvement need not shorten the total project duration. To shorten the duration of a project one has to shorten the tasks on the critical path. Note that the critical path may change completely if some re–planning is done. From this it becomes clear: The net "worth" of an improvement can't be estimated in a complex project. One needs a computer simulation. IT–based project planning makes such computer simulation very easy: Just insert the change and see what comes out. In that sense the above stated point b) helped to shorten the project duration in the order of several years. The success of the implementation of improvement measures was also easily controlled by the same project management tool.

3.2 Lessons learned

The above mentioned difficulties give some valuable suggestions for everybody who wants to install project management in a similarly complex project. Advice can be stated as follows:

i) Reserve lots of time and manpower for planning.

ii) Follow the general process of work as closely as possible during planning. Reserve a proper amount of time for developing a process map if necessary.

iii) Avoid complexity in the planning. It is better to have slightly incorrect or better fuzzy planning than to have an over complex planning or even a chaotic one.

Point i) becomes clear from the remarks of the last chapter. It is especially important to realize that planning is developing. The process of a development project does not have the separate steps "to plan" and "to develop". As stated already earlier the step "to develop" contains the important subactivity "to plan". A thorough project planning is by no means a waste of development time. It helps to shorten the project's total duration.

Point ii) can be put in other words: First determine the main activities to be done and their dependence. If there is an overall and complete agreement on this process chart, plan *when* the activities should start and how long they should take. The main secret is the separation of "what" and "when". It is only natural to spend too little time to develop a proper process chart (the "what"). To develop a process chart can be tedious and time consuming. However its outcome does not answer the most important question: "When will the finish date be?" Therefore the managers try to rush to the "when". They want to know the final end date. This is very understandable. However, such "rush" can be extremely counterproductive. A mistake in the process chart will create a planning mistake. Such mistakes will lead to an unreliable end date. What is even more important the whole acceptance of the planning may

suffer. Especially when the going gets tough people will point to the mistakes in the process chart. They will claim that the project planning can't be binding due to such mistakes.

Point iii) may be the most serious point. It was absolutely not honored in the software project discussed here. Almost all problems which occurred during planning can be routed back to it. It is only natural to be as precise as possible. However, everything has some margin of error. If one combines several things their margin of error will be combined too. The resulting net margin of error is not always easy to determine. A rough estimate should avoid to be over precise. An easy example is cost calculation. Let us assume that 8,600 parts are produced by a complicated machine during 1 hour. The operational cost for one hour may be $ 5,800. What are the costs for 100 parts? Is $ 67, $ 67.44, or $ 67.44186 the best answer? Many people might prefer the last number because it is so precise. Maybe they need the costs as precise as possible for hard bargaining. Reality can be different, however. It is quite possible that the machine will produce between 8,550 and 8,650 parts per hour. The hourly cost aren't known precisely either. Costs for true depreciation or maintenance are often just assumptions. From this the hourly costs aren't known any better than lying between $ 5,600 and $ 6,000. The correct output of the machine is therefore 8,600∀50 parts per hour, and its cost is $ 5,800∀200 per hour. The cost for 100 parts is therefore

$$\frac{5{,}800 \forall 200 \text{ \$ per hour} * 100}{8{,}600 \forall 50 \text{ parts per hour}}$$

The result is something like $ 67.5∀2.7 per 100 parts. The average of $ 67 is therefore by no means less accurate than $ 67.44. (The value $ 67.44186 is even ludicrous.) Note that an increase in accuracy in one of the two input numbers will hardly improve the accuracy of the final quantity. To see this let us assume that the engineers will improve their process quality so that they can guaranty output of 8,600∀5 parts per hour (An increase of accuracy by a factor of ten!) Inserted in the equation above yields costs of $ 67.4∀2.4 per 100 parts. The accuracy improved by a factor of 0.11 or 11 %. In project planning the precision is quite often much less precise than in this example of cost calculation. And as we have seen the inaccuracy of just a few quantities might lead to a low precision of the final result. If a certain task will take 10 days an its "exact" duration will be 10∀2 days. Adding proper margins of error for every task will end in some "period of time" where the project end date will be. Due to the mutual dependence of tasks their start and end date will vary accordingly. There are special simulation programs on the software market to simulate the effect. An add–on software makes such a simulation possible within MS Project 98. As mentioned above, the calculation time of MS-Project 98 can be quite long in complex projects as discussed here. A rough estimate for the simulation time yielded several months even on a very fast PC. For that reason it was never undertaken. Be that as it may it is very plausible that too much precision in project planning may have three nasty side effects:

- The effort for planning was partly useless.
- The project plan is hard to handle, because it is unnecessarily complex.
- Due to the unnecessary complexity chaotic behavior may occur.

The first point maybe tolerable, though it can stifle motivation. The huge effort for planning is at least partly a complete waste of time. The second point implies e.g. long calculation times or a tremendous effort to eliminate planning mistakes which will occur almost certainly. The third point can be a show stopper. It is plausible that due to over precise planning chaotic behavior will appear at least partly. It makes the entire plan worthless, if an increase in duration of one activity will lead to shifts of months or even years in some other area.

I will conclude this chapter with a suggestion which will at least reduce the risk of over complex planning. In the planning of the software project we had allowed that every single of the 10,000 activities could have been connected with some other. The engineers tried to be as precise as possible ending up with high complexity due to over precision. The way out would have been the following: Develop a reasonable process map of perhaps 300 process steps. Take these 300 process steps as headlines in the planning. These should be connected with one another as indicated in the process map. For each of these summary tasks one may define subactivities as desired. The subactivities may be connected with one another as appears necessary. However, subactivities under different headlines must not be connected. This rule is visualized in fig. 3.1. It is allowed that activity 1.1 is the predecessor or successor of activity 1.2. However, activity 1.25 must not be connected to activity 2.1. Different groups of activities may be connected by its summary tasks only as indicated by the connection of activity 1. to 2.

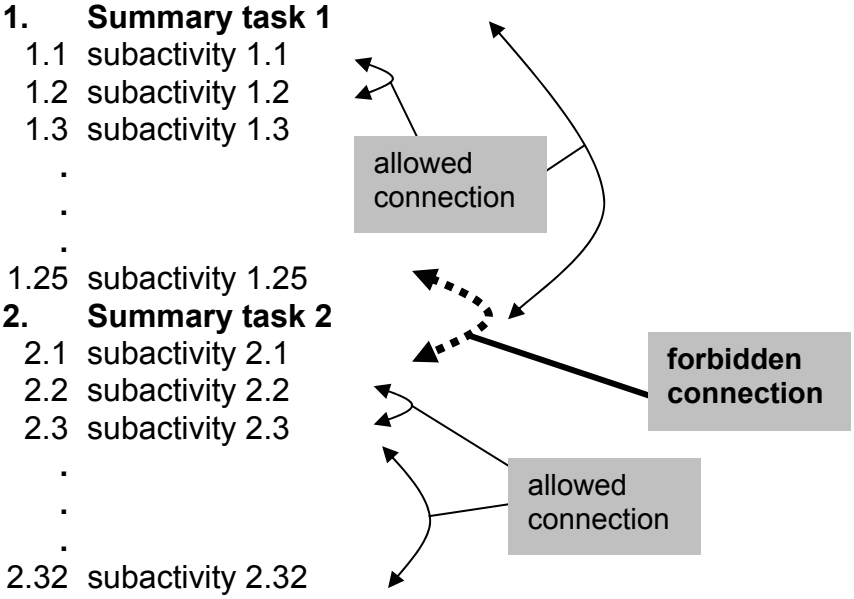


Figure 3.1

The suggested planning of fig. 3.1 yields an additional advantage for visualizing the planning. A standard way is to draw a PERT chart. It gives a good overview. However, having 10,000 boxes connected with many lines, as in the present example, makes a PERT chart worthless. Of course, one may suggest to display the summary tasks only. However, if there are important connections like the one between 1.25 and 2.1 in fig. 3.1 the position of the summary tasks may be determined by it. Without having the subactivities in the PERT chart the placement of the summary tasks may look completely illogical. Therefore a reasonable summary PERT chart is only possible, if the rules of fig. 3.1 are obeyed.

3.3 Alternative organizations

In this last chapter I will discuss project organization. The organization of the software project discussed here is indicated in fig. 1.3. As discussed in chapter 1.3 the organization was far from being optimal. It was not in

accordance with the process (cf. fig. 1.2). We never improved the organization within our management project. It was a very political issue. To construct a process organization is pretty much straightforward. But even with such process organization the project will consist of 10,000 activities done by 250 engineers. Even by avoiding complexity as much as possible (cf. fig. 3.1) the management of such a behemoth will be cumbersome. The idea for improvement can be borrowed from ordinary organization. In general a centralized organization can be optimal in a mathematical sense. This led to centralization in the 1970s. However, the early 1980s showed an over complex behavior in centralized organization. Therefore the 1980s and early 1990s were periods of decentralization or profit center organizations. (Up to my knowledge it was McKinsey, a management consultancy, who first framed the word "over complexity" in 1991.) From this it becomes an almost trivial idea to cut projects into "profit centers". Of course the profit centers of a project should be in accordance with the underlying process. (The same is true for ordinary profit center organizations.) From fig. 1.2 one will easily design a profit center organization as indicated in fig. 3.2. Its layout is in

Process Organization

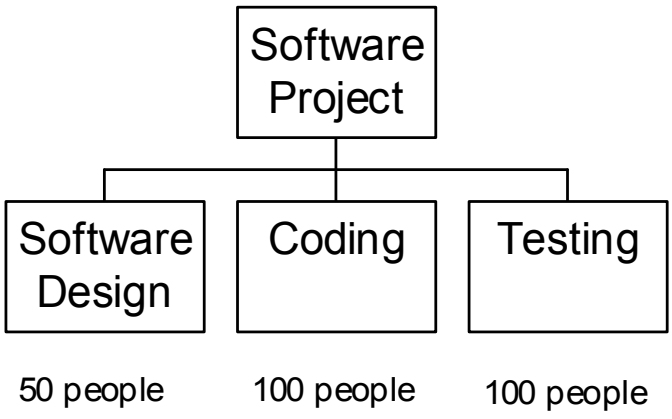


Figure 3.2

principle identical to a process organization. This is not surprising because a profit center organization is nothing more than a process organization with a proper controlling. The profit and loss responsibility makes it to a true profit center organization. To construct a profit center organization for a project is not very difficult. The key to success lies in the controlling mechanism including a personal benefit scheme.

Unlike in ordinary profit center organization the process steps in a project do not produce a product with a certain market value. Normally, a project (at least such as the software project discussed here) will produce "ideas". Therefore a profit center organization for a project is more like a profit center organization for the administrative branch of a company. To my knowledge nobody ever implemented such profit center organization, though it may be very desirable.

The controlling of the project should go top–down as in a typical balanced score card approach (cf. ref. 3.1). The controlling variable of the entire project is its worth. Unlike in ordinary organization there is no direct market value of an output. In general the cost of the project won't be a good substitute for its worth. Of course, in ordinary profit centers the total costs will come close to the total revenue. The free market will press cost close to revenue. This may be completely different in projects. Take for example the discussed software project. 250 engineers are working there for about ten years. Let the annual cost per person be $ 50,000 and one will end up with $ 125,000,000. On the other hand, the company expected to make many billions of dollars in profitable revenue out of the final product. From that the worth of the software

project comes closer to a billion dollars than 125 million dollars. This huge difference between cost and worth of a project is typical. (The same is true for an administrative branch of a company.) From this one can formulate an important statement:

➔ **The worth of a project is equal to the net benefit it will create.**

The net worth of the project must be controlled, if it is treated as a profit center. Once having defined the controlling variable an ordinary controlling process should be performed. The typical controlling process is displayed in figure 3.3. There one sees the important steps measure, compare 'as is' with

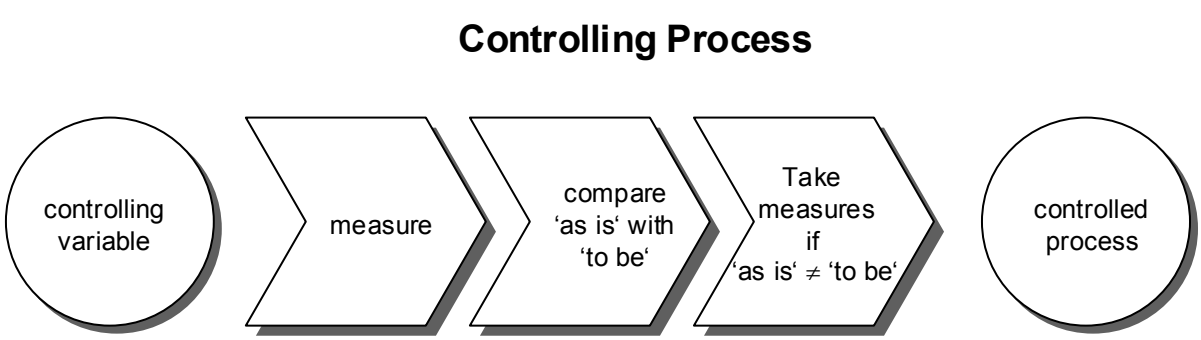


Figure 3.3

'to be', and take counter measures. Especially the last step (take counter measures) is quite often ignored. Despite the goal to keep the worth of the project as high as possible the controlling process will also give the actual worth of the project. It will be adjusted, e.g., if the project is ultimately delayed or the quality of the final outcome is poorer than expected. Such lowering of worth is the same as lowering the profit in an ordinary profit center. Once the general idea of controlling the entire project like a profit center is understood one can go on and create sub profit centers. Lets say that the net worth of the software project discussed here is $ 1 billion as estimated above. (One should make a more thorough calculation for a real project in order to get a

realistic value.) Once the total worth is an agreed number, one has to divide it under the process steps. This is definitely a crucial task. Again one has to decide which process steps contributes how much to the final worth. As an example take again the software project discussed here. Its process steps are again visualized in figure 3.4. Of course the $ 1 billion must not be

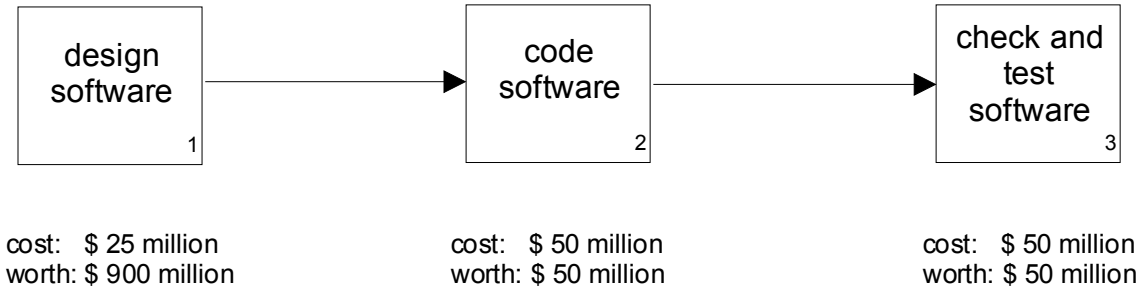


Figure 3.4

distributed equally to the process steps. It is also not correct to divide it proportionally to the costs of each steps. Insiders of the software project will easily agree that the last two steps are tedious but very straightforward. They are a "commodity". Therefore their worth must be about equal to their corresponding cost. Having 100 people in step 2 (code software) and 100 people in step 3 (check and test software) one will end up with costs of $ 50,000/person and year * 100 people * 10 years = $ 50 million. Therefore the net worth of step 2 and 3 may be estimated as $ 50 million each. Because the worth of the total project is $ 1 billion, the first step is worth $ 900 million. Step one consumes about 20 % of the cost, but it creates 90 % of the net worth. The result of this rough calculation is understandable. As stated much earlier (chap. 1.2) the brainwork is done in step 1. Therefore its worth should be dominant. Having defined the worth for each process step these numbers can be taken as controlling variables to perform the process of fig. 3.3. In the

language of ordinary profit centers the following will happen. The people of software design (cf. fig. 3.2) will create a "product" which is worth $ 900 million. Their costs are about $ 25 million. In that sense they will make a huge profit ($ 875 million). Of course they are also bearing the biggest risk. They will almost ultimately define the quality of the product. If they perform suboptimally the worth of the entire project will shrink to say $ 500 million. By the same calculation as done above their profit will shrink to $ 375 million. The cause of delay may also be in step 1. In the software project there was a penalty of $ 500,000 per day of delay. Therefore their profit could decrease substantially by being too late. (In the actual project the delay was in the order of several years. Reducing the profit to about zero.) Doing everything in accordance with schedule the people of software design will sell their product for $ 900 million to the people of coding. Being 100 days late would reduce the price to $ 850 million. If the people of coding will work according to schedule they will just add just $ 50 million to the cost of purchase and sell it with no profit or loss to the people of test. If the product of software design was delivered late, coding could easily make money by working faster than planned. Say the product is late by 100 days. Than the people of coding may decide to invest $ 1 million in order to be 100 days quicker. If they succeed they would make $ 50 million out of an investment of $ 1 million. The same is true for the people of testing.

In order to make such profit center organization really work one has to put a personal benefit scheme into play. This is very easy now. I have defined a profit and loss for each of the profit centers of fig. 3.2. All one has to do is to share a certain part of the profit with the management of each profit center.

Normally a couple of percentage points are justified. Note that the profit center software design can make a profit of up to $ 875 million. Three percent of it are over $ 26 million. It sounds like a huge sum although it is absolutely justified. If such profit is really reached the entire company will gain an awful lot. Why not share a small part of it with the people responsible for it? What if a profit center makes a loss? In almost all contracts senior managers (including CEOs) never share the loss. So one cannot expect it in a profit center of a project. However a real entrepreneur should be willing to take both: Share in profit *and* loss. Such real entrepreneurs are hard to find. This may be the biggest concern of all in regard to profit center organizations. Their bosses are quite often good administrators, mediocre managers, but not entrepreneurs. In my personal opinion most failures of profit center organizations are due to the fact of missing entrepreneurship. Another problem installing a profit center organization in a project may be its political acceptance. As mentioned earlier it was impossible to build a reasonable project organization in the software project. Just imagine the "crazy" idea of a profit center organization.

As a conclusion I would say that a profit center organization is by far the best way to control a giant project. However, there exists next to no experience with such organizations inside projects. Therefore some research is necessary before one can start within a real project. The best way might be to take a project which is already concluded successfully and is well documented or a running, well managed project. One should then simulate a profit center organization as a shadow organization. That is one should define the controlling variables exactly and question every major decision in

the project. Would these have been the same if profit and loss considerations had been taken into account? The result would be a shadow project with more or less success than the original one. This will finally lead to a confirmation or modification of the profit center approach for projects.

References

1.1 R. Sellien and H. Sellien, Gablers Wirtschatslexikon, 10th edition, vol. 4, Gabler, 1988

1.2 H. G Schuster, "Deterministic Chaos", Weinheim 1984;
Michael Grabinski, "Is there chaos in Management or just chaotic management?", Complex Systems, Intelligence and Modern Technology Applications, Paris (2004)

2.1 T. Gilb and D. Graham, "Software Inspection", Addition-Wesley, 1993

2.2 B.W. Boehm, "Software Engineering Economics", Prentice-Hall, 1981

2.3 S. Goldsmith, "A practical Guide to Real-Time Systems Development", Prentice-Hall, 1993

2.4 J. H. Hollom and K. J. Pulford in GEC Journal of research, **12** (1), 1995

3.1 R. S. Kaplan and D. P. Norton, "The Balanced Scorecard: Measures That Drive Performance", HRB January 1992