Manufacturing and Embedded Tool Software Project Management Guide

11/22/91

CONTENTS

INTRODUCTION

This document outlines how to manage manufacturing and embedded tool control projects. The goal of project management is to document answers to the following questions:

1. What requirement(s) is this code going to fulfill?

   Do the users agree with this?

2. How much work will it take to put the system in place?
   How long will it take to put the system in place?

   Does management agree to staff it?; is the effort justified?

   Do the users agree to the cost and schedule?

3. Is the Project on Schedule? Is the Project Within Estimates?

4. Does the code work? Is it maintainable?

Scope

This document does not describe how work is assigned or flows.

Our department develops four types of documentation to accompany software:

Project Management Documentation
Software Internals (Programmer's Guide)
Software Externals (User's Guide)
Disaster Recovery

While some of the topics among these documents overlap, project management documentation is concerned with commitments, responsibility and schedules.

This document does not replace the common sense which arises from experience. In this vein, the guidelines spelled out in this document attempt to give the project leader as much liberty as possible in creating documentation to fulfill the above goals.

OUR PROGRAMMING ENVIRONMENT

Project Categories:

We do three types of projects in our area:

1. Development

   A new tool. Some design work is necessary

   A major functional upgrade of more than 4 weeks effort to an existing tool. The design must be fit into an existing framework.

2. Enhancement

   A functional upgrade of less than 4 weeks effort to an existing tool.

3. Maintenance

   Work done to maintain the intended function of the code.

   e.g.: Fixing a bug that arises due to hardware changes or lower level computer subsystem changes. Upgrading operating system software. Updating documentation, disaster recovery.

How Manufacutring and Tool Software Differs from other IS Departments

… as related to what procedures are used

The main considerations in our area (relevant to the volume of project management documentation)
are:

1. How many programmers will be working on the software?

   - Approximately 90% of our projects are staffed by one programmer and one "backup".
   - Approximately 9% of our projects are staffed by 2-3 programmers.
   - Approximately 1% of our projects are staffed by more than 3 programmers.

In general, our design and testing documentation is reduced and can be delayed because we do not have to coordinate multiple programmers.

2. How much effort will a project need?

   - Approximately 70% of our effort is spent on projects less than 4 weeks.
   - Approximately 25% of our effort is spent on projects 4 weeks to 6 months.

- Approximately 5% of our effort is spent on projects 6 months to 2 years.

3. Is the application "glasshouse" or is it dependent on non DP (data processing) hardware (instrumentation, robotics, vision, etc.)?

Our area is heavily intertwined with engineering, hardware and vendor supplied subsystems.

Additional complexity is added by heavy dependencies on non DP equipment. We program in an uncertain environment; e.g. parallel development of software with the process/technology, changing product requirements.

Approximately 80% of the (types of) programming we do are done by engineering departments at other sites.

The business case for new projects is done by engineering in the capital justification for the machine we are supporting.

Schedules are driven by manufacturing and marketing needs.

How Much Should a Project be Documented?

There is no one project management methodology that works efficiently for all types of application software development. The purpose of project management in our area is to use the minimal effort to set direction, reach agreement and inform.

> An alternative view of project management has the goal of uniformity; doing a wide variety of projects with uniform procedures and documentation as insurance against oversights, and to make the project easier to understand by more people.

Too much project management (i.e.-using "big" project management techniques on small projects) will waste time on non essential issues.

Not enough will foster "wheel spinning" and misunderstanding.

How much a project should be documented depends on:

1. How ambiguous/open to interpretation the requirements are.

   How likely the requirements are to change as you are coding them.

   Whether the requirements seem incomplete and are likely to expand to with follow ons.

   RISK:
   The less well thought out a project is, the more likely this is to happen. Try to get the whole thing thought through as a complete package.

2. How "hard" the need date is.
   Can the project be easily put on the "back burner"?

3. Are the users likely to pressure management about deliverables or schedule?

In general:

1. Any development project should have accompanying project management documentation.

2. Small enhancement efforts need less management documentation. Often the information contained on the request (or engineering change request) form is sufficient.

3. Maintenance projects (of less than 4 hours OR that gate production) do not need up front project management documentation.

Commonly Heard Pros and Cons of Project Management

Historically, our area has resisted doing project management. To not waste time on the same arguments, they are summarized here:

Pro	Con
	Can take the effort that would have been done documenting and have put it into the coding.
Unwritten word of mouth requirements have no validity, are subject to interpretation, and change as the project proceeds.	
	Doing the project management won't help if the software gates the equipment. The users only want the thing to work according to their schedule.
Don't write anything down and you are sunk in any case.	
	You can't write down what to do if the users don't know what they want.
Writing down what the loose ends are helps everyone to focus on what's needed, and agree to the tradeoffs and options involved.	
Maybe the project should be delayed. Writing down the case is more convincing.	
Knowing what's fixed and what's unsure helps you to design the right degree of generality into the system.	
	The users can change the requirements as the project proceeds.
The programmers can (and should) revise the efforts and schedules accordingly.	
	Management only cares about project management when things go wrong, when they are confused, during an audit, or at appraisal time.
	No one checks to make sure projects management is carried out at all, let alone in a consistent manner.
	Project management has too many bureaucratic items as insurance against oversights. Procedures are so thorough and all-encompassing that relevant items needed for communicating get obscured. The time it takes to do bureaucratic items is trivialized.

SOME SOFTWARE ENGINEERING CONCEPTS (relevant to our area)

Requirements Gathering and Design

The leading problem in the structuring of computer software is understanding the user's application.

Our job is NOT to simply take user requirements as dictated and translate them into code.

The art of gathering requirements and design is to give the user a system where minor changes in requirements will be easy to implement without changing the SCOPE of the project.

> This means that the system should have the right degree of "generality". The "art" of requirements gathering and design is figuring out what is certain, and what is liable to change. This determines what can be hardcoded for the specific application, and what functions to implement (and to what degree) in a more general manner.

None of our projects have solid requirements which remain fixed for a long time. Software of this type is done by outside vendors and comes with the hardware.

The only completely general system is a programming language itself. Somewhere the PROGRAMMER has to draw the line and spell out limitations.

Programming personnel should always create requirements documentation by talking to the users involved and writing down what they want in a form that can be programmed.

> The users do not have a feel for what changes are easy and difficult to implement in a given design.

A documented set of requirements is a bridge between what the user thinks they want/will get, and the programming team's methodology of getting a computer(s) to do it.

Multi Programmer Projects

It is false logic that:

> If one woman can have a baby in nine months, then nine women can have a baby in one month.

A project with more than one coder must be PARTITIONABLE. The above project management methodology will not work because the task is (an extreme case of) a non partitionable project.

> **RISK:**
> As obvious as it sounds, in the effort to make dates it is often overlooked that there must be a workable way to divide a project up among coders if more programmers are assigned to bring in the end date.

Estimates

There are two types of estimates for a project:

1. How many person-months EFFORT the project will take to do

2. What TIMESPAN it will take to do the project.

EFFORT can be estimated fairly accurately. TIMESPAN is difficult to predict accurately.

> **RISK:**
> Be careful when you say that a project is DONE. A project is not DONE when the coding has been completed. A project is DONE when:
>
> 2\. The user has signed off a project closure document stating that the functions agreed to have been provided and are working properly in the intended environment (e.g. debugged in manufacturing - rather than at your desk or in a lab).
>
> 2\. Documentation and disaster recovery have been completed.

Example

After the requirements have been collected and the design completed, a project in our area can be accurately estimated. The following example illustrates the method:

Coding:	10.0 months
Debug at 20-30%:	2.5 months
Admin at 15-20%:	1.5 months
Documentation and disaster recovery at 5-10%:	0.5 months
Project Total:	14.5 months

Note that we do not estimate effort for upfront requirements gathering and design in advance. For rough planning purposes, this is about 40% of the coding effort.

Coding: Is estimated figuring the AVERAGE PROGRAMMER'S experience and knowledge level, unless the particular person working on it is known. It is estimated figuring the programmer has only the coding to do, but no meetings, vacation, education, admin, dept, etc. It is to be considered that the programmer will take a few breaks during the day. It is NOT to be considered that the programmer will work overtime during the coding phase (overtime is reserved for the unanticipated/unforeseeable).

RISK:
Estimating a project you are familiar with as if you were doing it will get you into trouble if you wind up not doing it. Document staffing assumptions when giving "aggressive" estimates.

Debug: means debug in manufacturing and includes minor changes to enhance usability, which were unforeseen during requirements gathering. Statistically, debug takes an additional 20% over coding for "glasshouse" applications, and 30% for person-computer-machine systems. Error handling code is expanded greatly during debug (when errors you never dreamed of happening occur, and errors you anticipated never happen).

In some areas debug is performed through the use of simulation and test cases. In our area the most efficient use of time comes from making up a debug plan which involves running test parts through the actual machine (sometimes referred to as the "code and fix" model).

RISK:
Often, we are only allowed to do the bare minimum part of the debug plan due to production needs. If this happens, make sure you issue a memo stating that debug will be done concurrent with production. Without this, I.S. will be blamed for holding up production should a bug arise that you could not test for.

Admin: means coordination among programmers, training new people due to staffing changes, project management and reporting.

> **RISK:**
> If management changes the staffing personnel, be sure to log the effort bringing them up to speed under a separate ADMIN id for the project, in order to retain the integrity of the project estimates.

Documentation: means software internals (programmers guide/design), software externals (user's guide) and disaster recovery.

To reiterate:

> It is the project leader's responsibility to collect requirements, design and estimate the project. It is the project leaders responsibility to point out how the project can be divided into pieces (so it can be partitioned among multiple programmers).
>
> It is management's responsibility to commit staffing to a project. Project partitioning considerations aside, the TIMESPAN of a project depends on the level of staffing.
>
> **RISK:**
> Do NOT consider the user's dates in the EFFORT ESTIMATE; however, show the critical dates (e.g. delivery, PRS) on the schedule. It is up to management to add more people to bring in a date. Do not let yourself be browbeaten into reducing an estimate to make a date. You are likely to work overtime to produce the original effort in a shorter period of time.

On the average, a full time programmer is available to work from 28 to 32 hours a week (out of 40), for an availability of 0.7 to 0.8 out of a total of 1.0. The other time is for vacation, education, department, admin, etc.

> **RISK:**
> Unless explicitly directed (and documented) by management, do not prepare schedules assuming overtime will be worked. Overtime is reserved for the unanticipated/unforeseeable.

For example, the timespan to do the above project with one half time programmer is calculated as:

```
          14.5 months effort
          ------------------  =  36.2 months timespan
           0.4 availability
                                      with one half time programmer
```

This would be reflected in this programmer's personal planning chart, e.g.:

	week1	week2	week3	
Admin, Education, Vacation	0.3	0.3	0.3	
Big Project	0.4	0.4	0.4	<==
Widget Line Maintenance	0.1	0.1	0.1	
Project 2	0.2	0.2		
Project 3			0.2	
Total	1.0	1.0	1.0	

The schedule for a small project might have to show the exact time where vacation and education fall. These do not matter for a larger project because they can be factored into the availability.

Project Tracking

There are three interrelated considerations during development:

1. Deliverables - how much of the needed function has been written

2. Effort - how much effort has been expended to date. Is the project within cost?

3. Timespan - is the project on schedule?

To track a project, use the simplification:

> Halfway through a project timewise, half of the estimated effort should have been logged.
>
> Less than half means the project is behind schedule.
>
> More than half means the project is ahead of schedule.

At the present time, we report project statuses to management once per month (through "Technical Measurement Reviews" to first and second line management), and progress reports.

LIFE CYCLE PHASES

The following phases each represent a step in the development sequence.

The goals of project management documentation are stated in the introduction. The programming professional has been empowered to create the amount of documentation in the form they judge necessary to best fulfill these goals; i.e.:

> These phases can be combined or split as needed.
>
> The items or work products within the phases can be rearranged.
>
> > If phases are combined, a rearrangement will make the documentation more readable.
>
> This guide attempts to define the minimal items/work products common to all of our projects for good project management.
>
> > If an item is inapplicable, it should be stated as such (to avoid having people ask why it's not there).
> >
> > Items can be combined if it makes sense to do so.
> >
> > > e.g. - for smaller projects: combining the hardware and software architectures into one diagram, combining the requirements descriptions with the design descriptions.

Some project management environments are very structured. (e.g.: step by step instructions, fill in the blanks forms, checklists).

> The variety of projects we handle does not allow us to adhere to a rigorously defined procedure without wasting time on some items. If an item/work product not mentioned in this document should be done (to fulfill the stated goals of project management), it is the programmer's responsibility to foresee and add it.
>
> > e.g.: dividing design into high level design and low level design; dividing test into high and low level testing.

Phase 0 (Analysis): Proposal and Requirements

1. Give a high level description of WHAT has to be done in software, without going (too much) into HOW it will be done. Describe the machine/problem and how software will fit in. Point out the applicable factors; e.g.:

   a) Hardware
   b) Vendor computer subsystem functions and dependencies
   c) Server dependencies
   d) State how many machines/installations there will be and where they will be located if known. Point out differences between machines if known.

2. Give the user's schedule. Point out:

   a) Hardware availability
   b) PRS (Production Run Start)
   c) Qualification

3. Name the responsible programmers and engineers.

   Names, departments, telephone numbers, email addresses.

4. Justification:

   a) For a development project: defer to the user's original tool justification.
   b) For an enhancement project: restate the user's justification given on the ECR (engineering change request) form.

5. State the AHT (Actual Hours Tracking) Project Id that future hours spent on the project will be logged under.

Optionally:

6. Alternative Architectures:

   If there are several different plausible ways of doing the project, lay out the alternatives with pros and cons.

   RISK:
   Answering "why are you doing it this way?" questions can sidetrack you later in the project. Try to head them off at this stage of the project.

7. Feasibility

Feasibility is done if some "up front" programming is needed in order to get a feel for the complexity of the system, due to a lack of experience.

It is not a blank check to program with no end in site. The questions to be answered should be listed, along with a reasonable approach for their resolution (e.g. testbed development, writing of algorithms and stubs, experimenting with a prototype machine).

"Gut feeling" target dates should be shown.

> **RISK:**
> None of the software developed in feasibility is obligated to be used in manufacturing on the floor. Make sure your users understand this.

Phase 1: Definition and Design

Non Formally Documented Project Management Items:

1. For development projects, a design review (before preparing the design documentation) is usually constructive and worth the effort. It is advised that this be done before a group of colleagues familiar with the type of project underway, chosen by the programmer doing the work.

   RISK:
   While the programmer doing the work has the final say on how a project is to be done, meeting minutes of the design review are recommended (to record if significant issues were raised and what they were). This will be valuable in case oversights or controversy arise later in the project.

2. Please note that it is not mandatory to include details of the Internal Design documentation in the Project Management documentation, beyond a Software Architecture Overview.

   Software internal design documentation is not normally part of the formal "commitments and schedules" aim of project management documentation. It is part of what will eventually become the "Software Internals" (Programmers Guide) technical documentation.

Formally Documented Project Management Items:

1. System Description

   Outline how the system will work in a clear concise manner. Relevant sections might be:

   a) Hardware Architecture

      Use a diagram to help describe the machine/problem and how software will fit into it. Give a high level description of what has to be done in software, as well as a high level description of how it will be done.

      i) Hardware

      ii) Division of functions between lower level vendor computer subsystems, hardware and our software.

iii) Server/MES system dependencies

b) Software Architecture

This will eventually become part of the Software Internals documentation. Give an overview of what will not be apparent from looking at the commenting found in the code; e.g.:

How the system fits together and relates to the hardware.

Timing considerations. Sequencing considerations.

Methodology for handling multiple product types, multiple machines.

c) Functional Description

Describe the steps the user will go through to run the system, as related to the software.

2. Limitations

The best way to describe what will/won’t be done is to anticipate what the users are most likely to want that they will NOT get and list it.

Provided the design allows it, some of the functions that the users are not getting can be part of a follow-up enhancement/phase II of the project.

RISK:
Keep the design open for future enhancement; but, try to keep development projects as simple as possible and focused on what is critical for production. Try to code and debug the system one step at a time, and delay non critical sophistication as follow on features.

3. Technical Assumptions and Risk Assessment

a) Without "shooting yourself in the foot", list TECHNICAL assumptions which if not true, will cause you to redo the design and/or do a lot more work.

e.g.: data volume assumptions, speed and timing limitations, mid cycle state determination and restart, error recovery, startup/power up assumptions, data transfer rates, server availability.

b) List things that the users didn't seem sure of, or that you had a difficult time getting a consistent answer on.

4. User Ordered Data Processing Items

a) List the configuration of all equipment the user has to order (e.g. materials for prototyping/debugging runs, facilities).

b) List all software the user has to order (e.g. Windows, Excel).

c) List all communications lines to the user has to run.

d) List all other Server/DP Center items (e.g. logons, installation of program products) the user has to arrange for.

5. User Interface

a) What the operator and engineer will see when they go to use the function being provided.

E.G.: screens, lights, buzzers, bells, barcode wands... The exact screen layouts are not necessary, but a description of all of the major function being provided is necessary.

Show how screen functions will be divided into a menuing scheme.

If applicable, point out real time vs. on demand update features.

b) Programmer/Engineer/Maintenance diagnostic and error logging features.

c) Data Security: if applicable, point out how passwording in the system works. (e.g.: point out screens restricted to engineering).

If a tool is able to access sensitive data, it is mandatory to point out the security protection mechanism (e.g. passwording, badge protection for the room).

6. Features being provided that are not directly observable.

e.g.: control loops, logistical or product-process tracking database correlation, non apparent real time features.

7. Signoff

Name the responsible programmers, engineers and involved managers. Have them sign that they understand what we will do and (more importantly) not do in this stage of the project.

Phase 2: Estimates and Scheduling

All estimates and schedules prior to this phase are for rough planning purposes only and imply no commitment on anyone's part.

> **RISK:**
> It is common that you will be asked for an "off the cuff" estimate or schedule in writing prior to this phase. Make sure you mark it as "Non Commit Pre Requirements/Design Estimate" (any similar statement will do). You are likely to encounter a lot of grief later if you don't make this clear at that time. Do NOT commit to any estimate until the requirements are agreed to in writing. Effort estimates tend to increase as requirements and limitations are brought into focus.

1. Project Effort

   Break the software down into items, tasks or functions and estimate each one. If they are interrelated (not fully partitionable), it is OK to put in some "system integration" (or putting it together) effort.

2. Schedule

   a) Show critical dates, hardware availability needs and when what parts of the project will get done.

   b) Show staffing assumptions with names if possible.

   c) Show how much machine time for debug is needed.

   d) For one programmer projects - name the "Backup" programmer. If a Backup Programmer cannot be named, document it.

   See the "Project Closeout" phase for a description of the Backup Programmer's responsibilities.

3. Scheduling Assumptions and Risks

   E.G.: Hardware availability, staffing exposures, experience assumptions, vendor subsystem delivery, data supply dependencies.

4. Signoff

   a) The users should sign that they know what the I.S. committed dates are, even if they want it sooner.

   b) Management should sign that they agree to staff the project.

   c) If involved, host system (e.g. data supply) management should sign that they understand how the schedule depends on their piece.

Phase 3: Development

Non Documented Item:

On a larger project, reviewing the code with the backup programmer periodically during development is highly recommended.

> **RISK:**
> If disagreements arise, the programmer who is responsible for making it work has the final say in how things are to be done; however, document minutes of the review if significant issues/concerns/disagreements are raised. Add a hardcopy of this to the project file.

Documentation:

Once a month in our area a "Technical Measurements Review" is presented to first and second line management, and a progress report is written for first line management.

No formal documentation outside of that contained in the standard forms for these reviews are needed during the development phase of the project.

Completing the necessary information for the Technical Measurements Review presentation constitutes "tracking" the project to make sure it is within effort estimates and on schedule.

The relevant sections of the "Project Profile" form presented in this meeting are as follows:

1. Resource

   The planned headcount is taken from the Phase 2 scheduling and expressed in person-quarters:

   There are 3 months per quarter

   There are 4.3 (= 52/12) weeks per month

   The actual headcount is taken from the AHT (Actual Hours Tracking) time logged against the project.

   Whether the project is behind or ahead of schedule, and by how much, is based on the efforts to date and is calculated as follows:

```
                           time since schedule started
PLANNED EFFORT to date = --------------------------  x  total effort
                                total timespan



ACTUAL EFFORT to date =  total effort logged under AHT since schedule started
```

As described earlier in this document, this is a simplification because it does not include deliverables.

The most important things to the user are deliverables and schedule; however, until the deliverables are done, quantifying how "done" they are is very subjective (like rating how "done" an egg is before it is laid). Typically, the first 90% of the deliverable is completed according to the schedule and the remaining 10% overruns the schedule.

2. Problems

Problems involving scheduling, assumptions, etc. are reviewed. Emails/memos which supplement verbal agreements impacting scheduling, estimates and deliverables are reviewed.

Try to keep the technical nitty-gritty to a minimum. The essential issues for second level management are what we are providing and when it will be done.

3. Change Control

Changes in requirements which will impact the schedule are presented as updates to the project management documentation.

Other changes to the schedule (e.g staffing, hardware delays) are presented as updates to the project management documentation.

Reference how the users have been informed of these changes (e.g. signoff, emails of whatever date, meeting minutes of whatever date, etc.).

Significant changes should have a signoff or other specific acknowledgement by the user.

Phase 4: Debug

During the Debug Phase, the same project tracking documentation as was done during development are produced for the monthly reviews, but in addition, a test plan and user education plan are developed during this phase.

1. Test Plan

A test plan can be done prior to this phase, but (most of the time) the best ideas of how the system should be tested will form after coding.

It is assumed that the user will test all screens for acceptable performance with valid input, invalid input, no input, too much input, out of sequence input, etc. It is also assumed that the system should handle communication failures (e.g., unplugging a server cable or RS-232 line) in an acceptable manner. These do not have to be documented.

Most of the time we are dependent on the users for "shaking down" the system. An approach for testing the system should be documented. User dependencies should be concentrated on when documenting a test plan.

At the least, this should be an agreement stating how much product of what type should be run through the system before it can be said that the system is running in a bug free manner.

Some user dependencies that might be relevant to testing a given system are:

A "gut" feeling of how much product of what types should go through the system before it can be said it's working OK.

For Machine Control: If there are computer subsystems, how many power down/ups (with the machine in what state) should be performed to make sure the system recovers/comes up correctly.

How many times and at what stages mid cycle restart should be tested.

Test the timing/speed/product processing limitations of the system.

For Testers: How much product with known errors of what types should be available to put through the machine.

<u>Optionally:</u>

2. Training Plan

Most of the time the user will not need any extra education, since they have helped formulate the requirements for the system and they have done the acceptance testing.

Occasionally there will exist a need for a training plan. This will be when the system has a lot more function and room for error/confusion than the user is acquainted with. This usually arises when an enhancement is done to a program product and installed as part of the system. Part of this might include:

References to outside classroom training that the user should plan on getting on the program product.

An outline of the training the programmer should provide so that the user understands how the program product was tailored/altered/interfaced to the system.

It is the engineering user's responsibility to write the operator spec and to train the operators.

Phase 5: Project Closeout

System Documentation

Basically, there are three types of documents:

1. Externals (user) documentation
2. Internals (programmer) documentation: A list of the modules abd where they "live" in the system (Daemons, dlls, manually invoked utilities, cron invoked jobs). Relationships between the modules.
3. Disaster recovery documentation:

   This can be combined with Internals. List the modules, where they are stored and procedures for recreating the system software. Assume familiarity with DOS, Windows, Unix, ftp, etc. List program product ordering information.

Disaster Recovery

Upload non program product code to a host library so that the system can be recreated; e.g.:

1. Source code
2. Accompanying MAKE files (if C language), resource files, etc.
3. Executable code or object modules if source code is not available.

Quality Code Definition:

1. Does the Job:

   The user agrees that it performs the committed to function in a bug free manner.

2. Maintainable:

   A programming peer has reviewed the code and documentation and has agreed to act as a support backup.

Project Closeout Document

These factors have been incorporated into the project closeout document, which has the following items/work products:

1. User Signoff

   Normally, users do not willingly sign anything. They fear if they sign, nothing else will get done on the project. Point out that nothing else outside of the original set of requirements will be done until they signoff. Write down the common understanding to date of what has been accomplished (e.g.: Phase 1 is done, Phase 2 is yet to go). If necessary, make up a "laundry list" of items within the original scope yet to do, append it to the signoff, and get the users to sign.

   If there are no complications and the user agrees that everything has been completed, this item can be skipped. Processing the department "Project Closure Form" will be sufficient.

2. Code Review and Signoff by Backup Programmer OR

   Signoff by the department documentation and disaster recovery coordinator.

   A "backup" programmer is named on projects staffed by a single programmer.

   It is not necessary for a "backup" programmer to understand the code as if they wrote it. It is necessary for the backup programmer to:

   a) Know basically what the system does.

   b) Be able to recreate the system from the source code.

   c) Reinstall the system.

   d) Be prepared to alter the source code, given a large enough learning curve, should the need arise.

   The backup programmer can therefore expect that a reasonable level of coding structure, commenting and documentation be provided to aid them.

3. AHT (Actual Hours Tracking) Summary

Prepare a chart showing the estimates for the project and the actual AHT hours expended.

It's OK for discrepancies to exist. Put in some explanations.

e.g.: staffing changes, requirements changes, debug problems, hardware changes, false assumptions.

SPECIAL PROBLEMS

The following project management problems are listed in the approximate order of their likelihood:

Not Enough Time to Work on the Project

Because the development programmers in our area also have unpredictable code maintenance responsibilities, this will be the leading cause of I.S. delays.

It might arise that the project has fallen behind schedule and cannot get caught up with the allocated staffing. This should be detected in the course of preparing monthly tracking information. When this happens:

1. Meet with your manager and try to get some help; e.g.:

   Someone to attend to the maintenance calls, so that your availability on the project will go up.

   Someone to help you code the project, if it is partitionable.

2. If you cannot get help, it will be necessary to reschedule the project based on the actual staffing you have experienced. Document a new schedule, review it with your manager, and call a meeting with the users to explain it. Have them sign off that they understand the new schedule.

   Note that understanding a new schedule does not mean they will like it or accept it; but make sure they at least signoff that they have seen it.

   RISK:
   Do not depend on working a lot of overtime at the project end to make the scheduled date. The earlier you can identify a scheduling exposure the better.

Changing Requirements

Should the users insist on adding function which will impact the promised dates, redo the applicable part of the project management documentation to describe the new requirements, effort and new schedule, and have both the users and I.S. management sign off.

Hardware Delays

If these cannot be worked around, at least send out a memo stating there will be an end date delay due to the hardware delay. Copy I.S. management and present the memo as a problem at the monthly review.

Post Installation Support

RISK:
Train equipment maintenance on diagnosing hardware related problems. Try not to accept calls from the operators after project closeout, or you will be forever running out to the floor. Engineers should be the first line of support in answering the "how does this work" questions. Equipment maintenance should be the first line of support for hardware vs. software problems.

Managing Vendor Coders

If coding on the project is to be shared, clear dividing lines should exist in the design to partition the function between organizations.

RISK:
Without this, there will be a lot of "finger pointing" during debug.

With shared maintenance responsibility, who updated the code last should be clear.