Niwot Ridge Consulting 4347 Pebble Beach Drive, Niwot Colorado 80503
720.406.9164 www.niwotridge.com

Nine Best Practices
The Software Management Framework

3 June 1999

Authors, Approvals, and Reviews

Revision A: 4/26/99 ..... GB Alleman
- ✓ Original Publication

Revision B: 5/26/99 ..... GB Alleman
- ✓ Completed check list sections
- ✓ Completed essential elements sections

Revision C: 6/1/1999 ..... GB Alleman
- ✓ Final edits for all sections

Published: 6/2/1999 ..... GB Alleman
- ✓ Added Technical Publication comments

Distribution List

Men occasionally stumble over the truth, but most of them pick themselves up and hurry off as if nothing had happened.

– Winston Churchill

Table of Contents

Table of Figures

1 OVERVIEW

> *For companies where software is important to business operations and to the products that they market, excellence in software is a vital business goal. However, software has long been one of the most troublesome technologies of the 20th century, and one that has long been resistant to executive control. One of the main reasons that software is difficult to control is because it has been difficult to estimate software projects, and to measure software quality and productivity in an accurate way.*
>
> — Capers Jones [1]

The management of software development is fraught with risk: technical risk, market risk, requirements risk, and financial risk. This paper describes nine (9) key management principles for guiding the development of a software project. These principles are not original. They are taken directly from the work of Norm Brown, the founder and executive Director of the Software Program Managers Network (SPMN). SPMN is a consortium of Department of Defense Program Managers who are dedicated to improving the practice of managing software acquisition and development projects in both commercial and government domains. Full credit for the nine principles is given to Mr. Brown and the SPMN.

Although much of the work of the SPMN is focused on Department of Defense projects, all of the principles can be applied to commercial development activities, with the appropriate adjustments. The bottom line is, good software development practices are universal. There may be aspects of embedded navigation and guidance that are different from design and layout of newspaper pages, just as there are differences between design and layout and payroll programs.

This application to the software development project forms the basis of the Software Improvement Process (SPI), which will move the organization from an *unstructured* development organization to one that behaves in a *repeatable* manner. This repeatable behavior will be modeled after the Software Engineering Institute's Capability Maturity Model Level 2 compliance. [2]

1.1 SOFTWARE MANAGEMENT FRAMEWORK

The Nine Best Practices constitute a management control system that provides a disciplined set of processes for containing and directing the complexity of a software development project.

Figure 1 describes the relationship between these Nine Best Practices and their outcomes.

The Best Practices can be classified into the following major categories:

(a) Identify and correct defects and potential problems as early as possible

(b) Plan and estimate all project activities and deliverables

(c) Minimize rework caused by uncontrolled change

(d) Make effective use of the resources

---

[1] "What It Means to be 'Best in Class' for Software," Capers Jones, Software Productivity Research, Inc. February 1998. www.spr.com

[2] *The Capability Maturity Model: Guidelines for Improving the Software Process*, Software Engineering Institute, Addison Wesley, 1995.

The reader's first reaction to these statements may be *aren't you restating the obvious*?. The nine best practices are in fact obvious. The problem comes when it is time to deploy them into the project environment and attempt to reap the rewards.

These principles are practiced by all good project managers, whether they are building complex software systems or pouring concrete for the foundation of a building. They are obvious, they are intuitive, and they are simple. So why is it so hard to put them into practice?

That is the million–dollar question. In the case of many projects it may be the biggest question faced by company in some time.

How can the development team deliver "Next Generation" product on–time, on–budget, with a feature set acceptable to the initial customer base, while meeting all the intermediate demands of tradeshows, budget constraints, technology adoptions and other outside forces?

The source of these best practices is the *Software Program Managers Network*, www.spmn.com. What is presented here is a restatement of the concepts, tailored for the project and the culture of the organzation. The materials provided by the SPMN and other resources are available on the web. A brief overview of the best practices concept is provided in "Industrial–Strength Management Strategies," Norm Brown, *IEEE Software*, July 1996, pp. 94–102.

1.2 CONTEXT OF THIS DOCUMENT

The *Nine Best Practices* described here can be applied to software development projects that are undertaken in nearly every organization. Many of the practices are *common knowledge* in mature development organizations. In less mature organizations, the practices may be apparent, but not put to work on a daily basis.

This document is meant to stimulate process improvement, not provide the guidelines for process improvement. That is a complex process best worked out through other means, starting with the Software Engineering Institutes' Capability Maturity Model found at www.sei.cmu.edu.

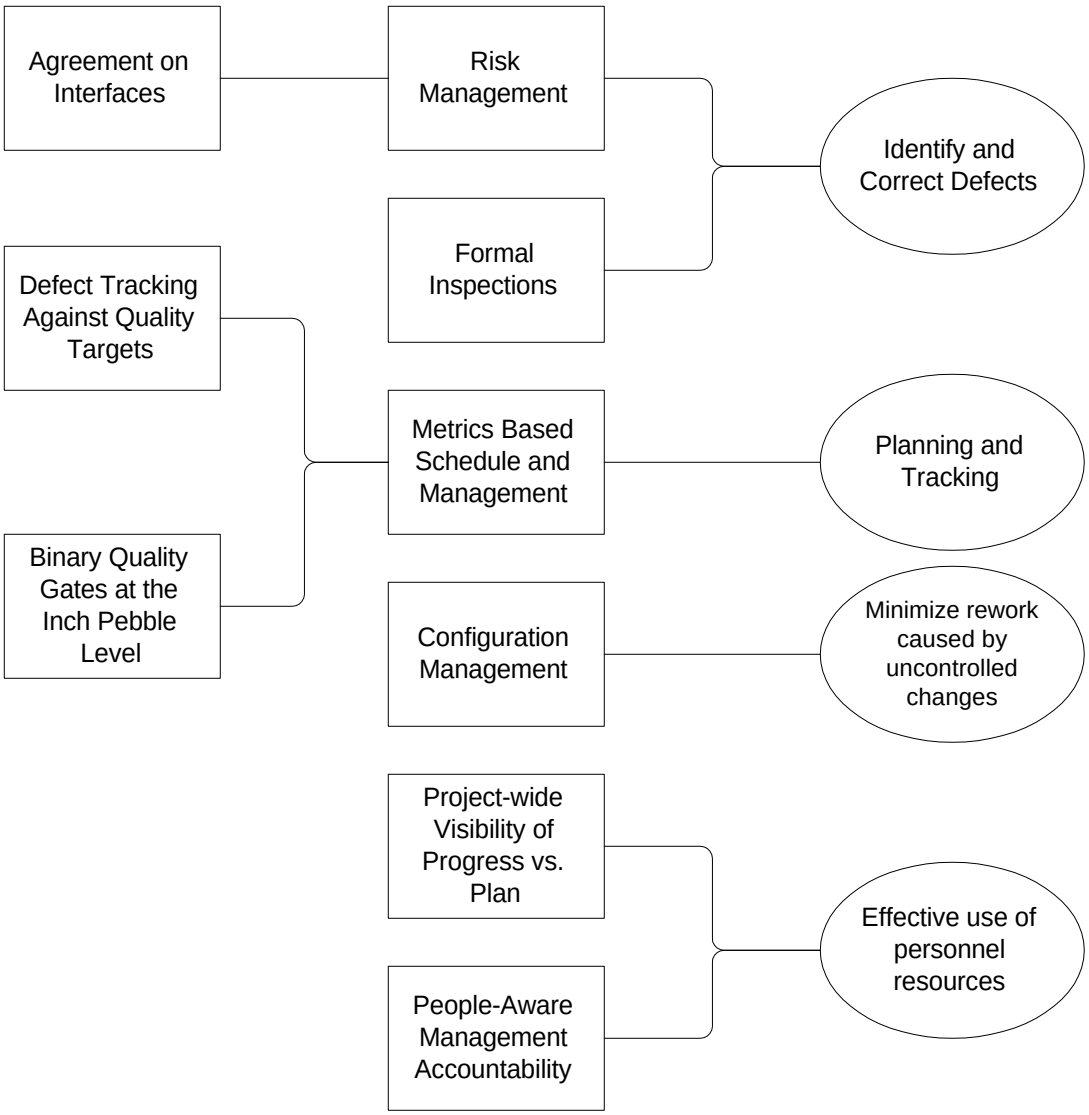

Figure 1 – Nine Best Practice Principles

1.3 MANAGEMENT ENGAGEMENT

In order for the Nine Best Practices to be effective management must be engaged in specific ways. This engagement involves:

- ☑ Commitment – to the practices and the consequences of the practices. This commitment usually comes in the form of a *formal endorsement* of the process and the deliverables from the process. This endorsement will take place through a *contract signing* in which top management agrees that the Nine Best Practices are the official activities of the company and the project team. By officially sanctioning the Best Practices approach both top management and all the participants agree *in public* that they are committed to make this work.
- ☑ Action – to implement the best practices. Have a commitment is easy, making good on the commitment is the hard part. A *call to action* approach will be used to deploy the Best Practices. The action needed to deploy the Best Practices will be managed just like any other project, with a detailed project plan, well-defined outcomes, and measurable deliverables.

- ☑ Funding – for the changes that will result from the practices. In order to accrue the benefits of the Best Practices, money must be spent. The actual funding details are not currently known. The total amount will be small compared to the total investment for the project. The return on this investment will be very large – a major contribution to the successful completion of the product.
- ☑ Follow-up – for the behaviors that result from the practices. Using the Best Practice of *Project-Wide Visibility of Progress Versus Plan* the deployment of the Best Practices will become a visible project. A project schedule will be posted, progress to plan meetings held, and progress reports published.
- ☑ Measurement – of the outcomes of the practices. With measurement, management can not take place. This is a Best Practice item that will be used for deploying the Best Practices.

2 NINE BEST PRACTICES

The following Nine Best Practices provide the foundation for the management of a software development project. They are listed in an order that could be construed as priority, although each best practice is a critical success factor to the project, with no specific priority or sequence.

2.1 FORMAL RISK MANAGEMENT

Risk management may be a new concept for a project team. Here are some warm-up statements to get us focused.

If you're not managing risk, you're managing the wrong thing – Rear Admiral Bill Carlson.

Risk management is project management for adults – Tim Lister, The Atlantic Systems Guild

These are strong statements, but they focus our attention on the current gaps in the project management process. A good project manager must be able to list the top 3 risks to the project when asked. More risks will surely exist. Capturing risk, managing the risk through mitigation strategies, and reporting risks to senior management must become a *normal* process for the development department.

One way to focus our attention on risk management is to realize that stupid risks are bad. Stupid risks are those risks that are taken even though they can be mitigated or avoided with little or no loss of benefit to the product and at only a marginal expense.

2.1.1 What is Risk

Many events occur during the course of a software development project. Risks are distinguished from other project events in the following way: [3]

- ☑ A loss associated with the event – the event must create a situation when something negative happens to the project. The loss associated with the risk is called the *risk impact*.
- ☑ The likelihood that the event will occur – the probability of the risk occurring must be understood. A likelihood figure will be used, rather than a probability figure. The likelihood of a risk will be measured from 0 (impossible) to 5 (certainty), in the Risk Radar.
- ☑ The degree to which we can change the outcome – for each risk, it must be determined what can be done to minimize the risk impact or avoid the impact of the event. This is called *risk control* and involves a set of actions to be taken to reduce or eliminate the risk.

Risk Management Practice	Call to Action
☑ All software development has risk. It is an unavoidable consequence of the design process. The cost of resolving these risks is usually low if addressed early in the project lifecycle. As the project matures, the cost of mitigating the risk increases rapidly. Nearly every developer and project manager has some experience with this effect, but many pay lip service to managing risk early in the project through a	☑ Recognizing and acknowledging that risks are present in the project is the first step. In the past, this was not part of the normal project management process. By publicizing the risks, along with the mitigation plan, everyone can participate in the solution process.

[3] *Software Engineering: Theory and Practice*, Shari Lawrence Pfleeger, Prentice Hall, reprinted in *Software Tech News*, 2(2), DoD Data and Analysis Center for Software, www.dacs.dtic.mil.

formal risk management process.	
☑ Effectively managing project risks involves acceptance and decriminalization of the risks and the processes used to discover the risks. In order to address this issue management must make the commitment to address risk as part of the normal development process. Without this commitment, identifying and addressing risks will become a no win process to be avoided at all costs.	☑ By making risk management part of the everyday project process, the participants can grow to trust management and vice versa. Only by decriminalizing the discovery and reporting of risks can this trust be built.

Remember, unlike cheese or fine wine – bad news does not get better with age.

2.1.2 Essential Elements

The essential elements of Risk Management are straightforward: identify the risk, plan for their mitigation, report progress to plan, adjust the schedule accordingly. If it were only so simple, we would not need formal risk management. The following items form the basis of the risk management process.

Essential Risk Management Elements	Call to Action
☑ Identify Risks – risk management must include risks identified over the full lifecycle of the project. Each risk item must be characterized by the likelihood of occurrence and the consequences of the risk occurring in an unmitigated manner.	☑ The project will make use of the *Risk Radar* tool. This tool is an Access database containing the risks, their probability of occurrence, mitigation plans, and the consequences of moving forward with any mitigation.
☑ Plan for and mitigate risk – the identified risks will be kept off the critical path of the schedule. This requires both a detailed planning process and a risk assessment process.	☑ By formally planing for risk management and including risk management in the project schedule, it becomes a normal process of project management. The *Risk Radar* will be used to capture risks and manage them.
☑ Provide frequent status reports – by reporting status of the project on a frequent basis a *rhythm* can be established. Any disruptions in this rhythm as well as changes from the norm in the reported information can be cause for alarm.	☑ In the same way progress to plan is reported, risk assessment and mitigation planning become the normal activities of the project. Using the *Risk Radar,* the project managers will publish the risk information in the same way they publish the project schedules and progress reports.

☑ Maintain an anonymous reporting channel – a mechanism must be put in place to capture risks through anonymous reporting. This capability ensures there is an open and honest environment for the developers as well as all other participants. This approach helps to foster a *no cover up* policy.	☑ This reporting channel will be a bulletin board on Lotus Notes. We may get some undesirable and harassing messages, but this will just be the price for an open and honest communication channel.
☑ Decriminalize risk – although risk management may seem like a good idea to the project managers, it will only work if there are no negative consequences to reporting risk.	☑ The only approach here is to formally contract with top manager to decriminalize the reporting of risks. It is incumbent on all levels of management to listen to the risks, capture them it the *Risk Radar*, prioritize them, and develop a mitigation plan. There are facilities in the Risk Radar to *zero out* the reported risks. Capturing the risk is more important in the beginning than having a mitigation plan.

2.1.3 Impacts in the Project

The introduction of Risk Management to the project creates a new paradigm for the participants. In the past, projects were managed with traditional waterfall schedules. Although some risks may have been identified in the past, introducing formal risk management will have significant impact by:

☑ Making visible all the risks associated with the project.

☑ Providing a formal feedback mechanism for the project participants to voice their concerns about project risk.

☑ Building a new level of trust, by identifying risks, mitigating those risks, and closing gaps in the project schedule.

2.1.4 Actions to be Taken

In order to cause some form of change to occur, actions need to be taken, including:

☑ Install *Risk Radar* and operate it as defined in the User Manual.

☑ Assign risk management to the Project Managers as a deliverable in the weekly status meeting.

☑ Brief senior managers on the Risk Management paradigm and have them assure all the participants that reporting risks through the Project Manager and through the anonymous bulletin board will be rewarded rather than punished.

2.1.5 Status Checks for Risk Management

Here are some question we need to ask ourselves to confirm that Risk Management is being deployed:

☑ Has someone been assigned to capture the risk and operate the *Risk Radar* database?

☑ Has a risk database been set up to include all the categories and responsible parties for the project or subproject?

☑ Do risk assessments have a clear impact on program plans and decisions?

- ☑ Is the frequency and timeliness of risk assessment updates consistent with decision updates during the project?
- ☑ Are objective criteria used to identify, evaluate, and manage risks?
- ☑ Do information flow patterns and reward criteria within the organization support the identification of risk by all project personnel?
- ☑ Is the integrity of information managed and controlled?
- ☑ Are risks identified throughout the entire life cycle, not just during the current phase of the project?
- ☑ Is there a management reserve for risk resolution?
- ☑ Is there a risk profile drawn up for each risk, and is the risk's probability of occurrence, consequences, severity, and delay regularly updated?
- ☑ Does the risk management plan have explicit provisions to alert decision makers upon a risk becoming imminent? Have all project personnel been chartered to be risk identifiers?

2.2 AGREEMENT ON INTERFACES

System interfaces typically form the essential elements of a system's requirements and architecture. In the case of most projects, the system interfaces exist in several forms:

- ☑ Graphical User Interface
- ☑ Workflow
- ☑ UNAC
- ☑ NewsGrams
- ☑ Internal objects
- ☑ Vendor objects (InDesign, Lotus Notes, etc.)
- ☑ Databases
- ☑ Wire services

Many of these interfaces, for example InDesign, are beyond the control of the project participants. Many others cannot only be controlled, but form the core of the UNAC architecture. The creation and management of the interface domain is the role of the architect and the principal developers. Users of the interface, the programmers and end users, will have influence on the interfaces, but these influences come through requirements rather than software design.

2.2.1 Essential Elements

The process of agreeing in the interfaces has occurred in an ad hoc manner. Some level of formality needs to be put in place before proceeding with the project development. The level of formality will depend on the specific interface and the impact of this interface on other components of the product.

Essential Interface Agreement Elements	Call to Action
☑ The system interfaces must be designed at some level before software analysis and coding. This assumes that the interfaces are identified and their basic functionality is stated. The user interfaces and the external system interfaces must be clearly defined before the system architecture can be partitioned. Without this definition process, the locality of the system' functions cannot be defined.	☑ The first approach to interface identification will use the UML notation for class collaboration. Starting at the highest level of the system, the meta–class (or macro–class) components will be defined and connected. It is through those connections that the interface components will be identified.
☑ Although there is a desire to fully define and baseline the User Interface, there is some risk that performing this task too soon will lead to *locking* in the look and feel of the system.	☑ A formal User Interface development process needs to be put in place. The design and implementation of User Interface's is a very subtle and complex problem. Simply looking at examples of User Interfaces is not sufficient. A deep understanding of the business domain is needed.
☑ All other interfaces need to be specified in a formal manner, kept under revision control and used to build the test interface components of the system.	☑ If CORBA is used as the middleware (and this decision has not been officially made), then the interface specifications can be written in IDL.

2.2.2 Impacts in the Project

In the current project design process the explicit statement of the interface may or may not have been done. In order to assure that the interface specifications have been properly made:

☑ A design review will verify that the interface specifications meet the needs of all the participants as well as the guidelines for good object–oriented programming.

☑ The User Interface specifications will be decoupled from the object–oriented aspects of the system.

2.2.3 Actions to be Taken

☑ Define the guidelines for programmatic and User Interface specifications.

☑ Conduct formal design reviews for all the interface designs to date.

☑ Install a design review process for all subsequent design activities.

☑ Define and publish the programmatic interfaces in UML.

☑ Place all interfaces under revision control.

2.2.4 Status Checks for Agreement On Interfaces

☑ Is there a complete census on the parameters for each interface?

☑ Are the parameters defined down to the data element level?

☑ Are the interfaces stable?

☑ Have existing and future interfaces been defined, including consideration of those that may be required?

- ☑ Does the system specification include a separate software specification to show the hardware interfaces?
- ☑ Are opportunities made available for users to provide input and review the user interface descriptions as they develop?

2.3 FORMAL INSPECTIONS

Rework done to fix defects not found when the defect was introduced accounts for 40% to 50% of the software development budget on large projects. Since the cost of fixing these defects increases dramatically as the software matures, finding and repairing the defects before they are absorbed by the next phase is critical to the success of the project. Formal inspections are one solution to finding defects before they move into the system. [4]

2.3.1 Essential Elements

Essential Formal Inspection Elements	Call to Action
☑ Use small teams of prepared reviewers with assigned roles. These teams will be responsible for delivery working software that meets the quality criteria.	☑ The primary issue here is to define the quality criteria for the deliverables.
☑ Track the duration of the inspection session defects identified per session, the duration–to–defect ratio.	☑ This will be done using the defect tracking software.
☑ Individual analysis has been shown to be more effective than team analysis. [5] Because of this, the formal team inspection process will only be used for major components or architectural decision processes.	☑ Develop a formal inspection process for architectural components and major impact competent. The individual inspection process will follow the guidelines shown in the reference materials. In the short term, *any* inspection process will be an improvement.
☑ In situations where geographically separated development teams exist, it is difficult to conduct formal inspections.	☑ Some form of control over the inspection process for these remotely located teams must be put in place. One approach may be to train the remote teams so they can conduct their own inspections at the same quality level as the primary team.

[4] "What Makes Inspections Work?" Adam Porter and Lawrence Votta, *IEEE Software*, November/December, 1997, pp. 99–102.

[5] "Understanding the Sources of Variation in Software Inspections," Adam Porter, Harvey Siy, Audris Mockus, and Lawrence Votta, *ACM Transactions of Software Engineering and Methodology*, 7(1), January 1998, pp. 41–79.

☑ The primary source of delay in the inspection process is progress blocking due to synchronization of the participants.

☑ These blocking delays will be reduced by reducing paper work, automatically generating the necessary reports, and reducing the synchronization delays. The last activity will be enabled by:

2.3.2 Impacts in the Project

The introduction of an inspection process to the development teams impacts several aspects of the project:

☑ The effort needed to conduct the inspections must be accounted for.

☑ The necessary training for the participants must be made before the inspections can have nay benefit.

☑ The inspection process may fit the current culture of the subcontractors. Because the code they are developing is critical to the functioning of the system, the subcontractors must be brought into the process as soon as possible.

2.3.3 Actions to be Taken

A formal set of processes will be put in place to create, manage, and measure the formal inspection process. These include:

☑ Defining the guidelines by which the formal inspections will be measured.

☑ The reporting processes for the outcomes of the inspections.

☑ Interactions between the reviewers and the repair process. This will include the creation of a Software Quality Assurance department which will be tightly coupled to the development activities.

2.3.4 Status Checks on Formal Inspections

☑ Are reviews identified and all baselined artifacts placed under control before they are released for project use?

☑ Is the conduct of reviews structured, and are they integrated into the project schedule?

☑ Are procedures, standards, and rules for the conduct of the reviews established?

☑ Are metrics used to gauge the effectiveness of reviews?

☑ Is there a documented process for conducting reviews?

☑ Are entrance and exit criteria established for each review?

☑ Are a significant number of defects caught as early as possible (before testing at minimum)?

☑ Are reviews specifically focused on a narrow set of objectives and do they evaluate a fixed set of data?

☑ Is there a clear rationale for the scheduling of reviews?

☑ Are defects from reviews tracked and catalogued?

- ☑ Are reviews conducted to assess the quality of all software products before they are released for project use?
- ☑ Is the detailed design reviewable?

2.4 METRICS BASED SCHEDULING AND MANAGEMENT

This practice is designed to identify problems early in the development lifecycle. The early identification of problems is the only reason to have software metrics. The project management metrics are the yardstick for measuring progress to plan. They become the *early warning* indicators for potential problems and assist in localizing the source of the problem.

The cost and schedule estimates should be based on empirical data. In the case most projects, there is no empirical data, so forecasting the future by measuring the past will be difficult. This situation will be one of the top ten risks to project.

2.4.1 Essential Elements

Essential Metrics Elements	Call to Action
☑ Plan short duration tasks with measurable outcomes.	☑ By planning at the *inch pebble* level, measurable deliverables can be used to build trends and replace the lack of empirical data from previous projects. There are additional benefits for the development team, since this lays the groundwork for the *continuous build* and *smoke test* processes that will be needed in the later stages of the project.
☑ Review key milestones to determine that progress to plan is being made not only on the long level tasks, but also the high–level integration points.	☑ The process of reviewing not only the code level activities, but the key integration points for quality, cost, and all the non–functional aspects of the system is a critical success factor.
☑ Monitor and take action on cost and schedule indices.	☑ By reporting progress to plan on a periodic basis (sometimes even a daily basis), both the project manager and the participants can be assured of complete visibility
☑ Monitor and manage defect closure time.	☑ Some form of SQA defect tracking is needed for any meaningful management process. *You can not control what you can not measure* must be the watch words of the project team.
☑ Track the *control panel* metrics to identify potential problems before they become major sources of rework.	☑ By using a simple *control panel* approach, where the key project indicators are displayed for all participants to see, the question of visibility is removed.

☑ Do not hide problems with rebaselining the project.	☑ Once the project schedule is set, restarting the schedule simply hides the faults. Make the faults visible for all to see. This is the learning process. By learning from our mistakes, we can attempt to avoid them in the future.
☑ Manage defect closure time.	☑ The management of defect repairs will become as important as the management of the development process.

2.4.2 Impacts in the Project

☑ The development of credible project schedules will be a critical success factor for the project. This will include the identification of not only the major software development activities, but a moving window of detailed task schedules.

☑ The project managers will develop these schedules from information generated by the team members and consolidated by the project manager. This *bottom up* and *top down* approach follows the incremental development paradigm at the low level and an iterative development model at the macro level.

2.4.3 Actions to be Taken

☑ Install a process of schedule and defect management.

☑ Establish a *public* area for all project–related information. This can be electronic, but some form of *highly visible* display is also needed. Posting the schedules, milestone goal, and demonstration targets in a public space has a powerful positive influence on the developers.

2.4.4 Status Checks on Metrics

☑ Is cost and schedule performance tracked against the initial baseline and the latest baseline?

☑ Are the number of changes to the initial cost/schedule baseline tracked?

☑ Does the plan identify progress measures to permit rate charting and tracking?

☑ Are inspection coverage and removal rates tracked for the entire product and for each component?

☑ Are project estimates continuously refined as the project proceeds?

☑ Is a project feedback loop established between project measures and updated schedules?

☑ Are productivity levels and schedule deadlines evaluated against past performance and reflected in the risk assessment?

☑ Are the planned versus actual cost and planned versus actual schedule monitored?

☑ Is there automated support for metric–based scheduling and tracking procedures?

2.5 BINARY QUALITY GATES AT THE INCH–PEBBLE LEVEL

The devil's in the details is a powerful concept for software development. When project planning and monitoring are based on insufficient detail, the discussion of the status of the project is illusionary. Project management without detail should be called *let's pretend*. By pushing the project participants to plan at the fine–grained level, specific development activities can be far more effectively identified, planned, and tracked.

There is a school of thought that says software development is a discovery process and planning at too small a detail does not contribute to the success of the project. This point of view does not take into account the level of detail needed to adequately plan and manage the project across its lifecycle. Not only do the core software components have to be planned, but the product's infrastructure, installation and support, and sustaining capabilities must be planned. For most projects an *inch pebble* should be not more than 5% of the duration of the project and be performed at least 95% by a single, lowest level organization.

For a one–year (50 week) project, this means that that the *inch pebbles* are on 2 to 3 week boundaries. This assumes that the deliverable from the inch pebble task is a single product or entity.

The concept of *binary quality* gates is that there is an objective acceptance criteria and tests for determining that the output of a scheduled task is acceptable. If the outcomes are not acceptable, then the efforts to produce the task are not considered complete until they pass *all* their predefined acceptance criteria.

2.5.1 Essential Elements

Essential Binary Quality Gate Elements	Call to Action
☑ Ensure that development tracking is based on actual products.	☑ Define the deliverables for each task in terms of quality, non–functional requirements as well as the functional requirements.
☑ Every low–level task should be of short duration, expending a small (less than 5%) of the total budget (time and dollars). These tasks should produce a tangible product necessary for a required deliverable, and should have a binary quality gate defined for the deliverable.	☑ For a 50 week project this means no more than two (2) weeks between measurable deliverables, with one (1) week the norm.
☑ No credit for the task should be given if it does not pass the binary quality gate.	☑ By strictly defining the exit criteria, all participants are clear on the success factors.

2.5.2 Impacts in the Project

☑ The definition of the exit criteria must be made before the task item starts. Since this has not occurred in the past, some effort will be needed to install this behavior.

☑ The burden on the development staff for quality assurance activities must be addressed. This should be handled through the existing quality assurance personnel. Lacking that support, other staff will be needed to address the SQA gaps.

2.5.3 Actions to be Taken

☑ Define the level of SQA involvement for each phase of the project.

☑ Define the quality guidelines for the software components. Number of defect per function point or 1000 line of code (KLOC).

☑ Define the task deliverables in inch–pebble boundaries (no more the 5% of the project duration). Collect these deliverables in the project schedule and measure progress against these milestones.

2.5.4 Status Checks on Binary Quality Gates

- ☑ Are there credible project statuses and planning estimates based on inch–pebble quality gates, which can be aggregated at any desirable level?
- ☑ Have all activities been decomposed into inch–pebbles?
- ☑ Has all near–term work been decomposed into tasks no longer than two weeks in duration?
- ☑ Have subtasks (within the two week interval) been identified? Have these subtasks been sequenced in the PERT to assess if the maximum parallel operation can take place?
- ☑ Have achievable accomplishment criteria been identified for each task?
- ☑ Are tasks based on overall quality goals and criteria for the project?
- ☑ Are quality gates rigorously applied for determining task accomplishment, without exception?
- ☑ Is there clear evidence that planned tasks are 100% complete before acceptance?
- ☑ Is there clear evidence of successful completion of reviews?
- ☑ Are inch–pebble tasks on the critical path defined, enabling more accurate assessment of schedule risks and contingency plans?
- ☑ Is the set of binary quality gates compatible with the WBS?

2.6 PROJECT–WIDE VISIBILITY OF PROGRESS TO PLAN

This practice is designed to get the entire staff actively involved in identifying problems and risks. When everyone is involved, the likelihood of missing problems is greatly reduced. This broad involvement strengthens risk management and increases the probability of the project's success.

2.6.1 Essential Elements

Essential project–Wide Visibility Elements	Call to Action
☑ Put the schedule, risk management, and all deliverables in a public location for access by all participants.	☑ This is more than using Source Safe for the source code. The paper schedules are currently on display. The electronic versions are being kept on a server, but this is not enough. A *Project Web* site will be created for all the materials related to the project.
☑ Establish an anonymous communication channel.	☑ This may seem to be unnecessary for the project, but it provides a mechanism to capture information not normally conveyed during the course of business. There may be *noise* that arrives, but vital information may also be communicated in this way. The cost of this channel is nearly zero, using the web and a bulletin board.

2.6.2 Impacts in the Project

- ☑ The creation and management of credible schedules now becomes the responsible of all the participants. The project manager can be the *keeper* of the schedule, but the contributors to the scheduled are the developers.

2.6.3 Actions to be Taken

- ☑ Define a process by which the schedules can be defined, maintained, and updated to match the rhythm of the project. This rhythm will be defined by the following guidelines:
  - ❖ No more than 5% of the total development cycle between measurable deliverables.
  - ❖ Construction of a fine–grained schedule for a 6 week moving period.
  - ❖ Construction of course–grained schedule for the entire project.

2.6.4 Status Checks on Project–wide Visibility

- ☑ Are status indicators for measuring the project's progress updated at least monthly?
- ☑ Are the status indicators integrated into the management decision process?
- ☑ Is basic project status known by all project personnel?
- ☑ Can the development staff report problems as well as successes?
- ☑ Are project goals, plans, schedules, and risks directly available to the project team and interested parties?
- ☑ Is anonymous channel feedback visible to all project members?

2.7 DEFECT TRACKING AGAINST QUALITY TARGETS

In many cases there is no defect tracking mechanism at the project level. Keeping bug reports is not sufficient, the metrics of who, what, where, when, how, and why are as important as the existence of a simple bug list. By tracking these metrics, basic indicators relating to defects, schedule, cost, requirements, documentation and staff can be made visible.

Avoiding the *software snowball* effect is one of the outcomes of the software metrics. The bigger the code base gets the greater the defect rate of occurs. [6]

2.7.1 Essential Elements

Essential Defect Tracking Elements	Call to Action
☑ Implement practices to find defects when they occur.	☑ Continuous *smoke testing* of the software will be instituted once the baseline components are stabilized.
☑ Establish goals for delivered defects.	☑ Since no historical data is available, industry data will be used. [7]
☑ Configuration management will be used to report and track all defects found through all techniques.	☑ Source safe will contain not only the source code, but also the test code, test scripts and the test reports.
☑ Monitor the organizations defect removal efficiency by tracking the number of defects found and fixed during development and during the system's first year of field operation.	☑ This will build the needed historical data for follow on projects.

[6] *IEEE Tutorial on Software Risk Management*, Barry Boehm, IEEE Computer Society Press, 1989.

[7] "What is Means to be 'Best in Class' for Software," Capers Jones, Software Productivity Research, Inc., 1998, www.spr.com.

2.7.2 Impacts in the Project

- ☑ The resources necessary to set up and manage the defect tracking system must be found. Automation is the key to success here, but the initial investment must be made in order to accrue the benefits.
- ☑ The development staff must understand and support the concept of defect tracking. This requires and open and honest participation of all developers and managers.

2.7.3 Actions to be Taken

- ☑ Define the infrastructure needed to capture, manage, and report the defects discovered in the code.
- ☑ Put in place the processes to capture metrics. This will include development processes, SQA processes, and product management processes.

2.7.4 Status Check on Defect Tracking

- ☑ Are defect rate targets established for the project?
- ☑ Are these defect rate targets firm?
- ☑ Are consequences defined if a product fails to meet the target?
- ☑ Do project quality targets apply to all products?
- ☑ Are there circumstances defined if a product fails to meet the target?
- ☑ What techniques are used to project latent defect counts?
- ☑ How are current projected levels of defect removal empirically confirmed as adequate to achieve planned quality targets?
- ☑ Is test coverage sufficient to indicate that the latent defect level achieved by the end of testing will be lower than the established quality targets?
- ☑ Are the inspection and test techniques employed during the project effective in meeting quality targets?
- ☑ Do all discovered defects undergo Configuration Management, and are accurate counts achieved for defects discovered and defects removed?
- ☑ Is there a closed–loop system linking defect actions from when defects are first detected to when they are resolved?
- ☑ Is the defect information defined at a level of granularity that supports an objective assessment of resolution on a periodic basis?

2.8 CONFIGURATION MANAGEMENT

Not controlling the products of the development organization dramatically increases the complexity of the system to a level approaching chaos. The result is certain failure. The management of the configuration is based on two simple rules:

- ☑ Any piece of information approved at the quality gate level must be controlled through the configuration management process.
- ☑ Any piece of information that is concurrently used by more than one individual or organization must be controlled through the configuration management process.

2.8.1 Essential Elements

Essential Configuration Elements	Call to Action
☑ Use change tracking to formally track the status of shared products, problem reports, cost and schedule base line, test programs, internal and external interfaces, reports used by more than one organization, and all concurrently nonbaselined information shared within the project or approved for internal release through the binary quality gates.	☑ Make complete use of the Source Safe capabilities. If there are not sufficient capabilities to manage multiple version of each component, then some other application must be found. This will be a critical success factor for the project, since not only are multiple components being built, they are being built at different sites, by diverse groups.
☑ Automation is essential for this process.	☑ Automation will start with the use of Lotus Notes and all the management tools available in this domain.
☑ Monitor status and maintain records describing the history, content, and release records for products controlled by the configuration management system.	☑ Use the tools of Source Safe, to generate management reports that track the progress of the code as well as the defect repair activities.
☑ Continually evaluate the information under control for content, baseline integrity, and change status.	☑ This will be a requirement for the project managers as well as the software quality assurance staff.

2.8.2 Impacts in the Project

☑ A disciple must be put in place for configuration management. This can be done very simply, by restricting the software build process to gather code only from a designated location in the Source Safe.

2.8.3 Actions to be Taken

☑ Assess the capabilities of Source Safe to determine of they meet our needs.

☑ Define the procedures need to manage the code using Source Safe.

2.8.4 Status Checks on Configuration Management (CM)

☑ Is the CM process integrated with the project plan and an integral part of the culture?

☑ Are all versions controlled?

☑ Are configuration control tools used for status accounting and configuration identification tracking?

☑ Are periodic reviews and audits in place to assess the effectiveness of the CM process?

☑ Are all pieces of information shared by two or more organizations placed under CM?

☑ Do we have a process to measure the cycle time?

2.9 PEOPLE–AWARE MANAGEMENT ACCOUNTABILITY

This practice is intended to focus on people as a critical foundation for the success of the project. The single most important factor in the success of any software project is the quality, experience,

and motivation of the technical and support staff. A significant part of software development requires levels of intellect and creativity not found in ordinary technical jobs. The work of Watts Humphrey clearly shows that these attributes not only vary significantly between individuals, but that training, environment, tools, and leadership also greatly influence the creativity process. [8]

No matter how talented someone is or how experienced they are from previous projects, there is a significant investment necessary to provide that person with the proper understanding of the application being developed or maintained. Despite all the efforts to document the software (and the project actually has very little documentation), some vital information about the project exists only in the heads of the developers.

Because of this, the project manager must assume responsibility for addressing the staff turnover rate and the quality of the staff. The key to effective use of personnel, is to establish an open and empowering environment based on truth and trust. The project manager must be rewarded and held accountable for staffing qualified people – those with domain knowledge and similar experience in previously successful projects – as well as for fostering an environment conducive to high morale and low turnover.

2.9.1 Essential Elements

Essential People–Aware Management Elements	Call to Action
☑ Minimize burnout.	☑ Manage the time allocated to each task so that the developer has a high probability of success without *heroic efforts*.
☑ Help the staff maintain their personal technical competency.	☑ The project must invest in timely training. The subcontracts should be selected that already have the necessary training.

2.9.2 Impacts in the Project

☑ Since the management of personnel has not been the highest priority on the project, some formal activities must be put in place soon. These may include personnel reviews, anonymous information gathering processes and other personnel management techniques found in a mature software development environment.

2.9.3 Actions to be Taken

☑ Define the guidelines for managing the personnel on the project.

☑ Define the success criteria for personnel management.

2.9.4 Status Checks on People–Aware Management

☑ Will the development management staff be assigned for the entire project?

☑ Does the Project Manager have software experience in a project of similar size?

☑ Are all personnel fully aware of their role in the project?

☑ Is quality of performance acknowledged?

☑ Is personnel continuity ensured in light of changing company or project needs?

☑ Are opportunities for professional growth available to all members of the project?

☑ Do the developers believe in the goals of the project and that the schedule is feasible?

[8] *A Discipline for Software Engineering*, Watts Humphrey, Addison Wesley, 1995.

☑ Is the motivation and retention of personnel a key part of management assessment?

3 REFERENCES

☑ "A Condensed Guide to Software Acquisition Best Practices," Software Program Managers Network, 1997. Can be found in HTML format at http://www_ied.belvoir.army.mil/odisc4/docs/blackbook/blackbook.htm.

☑ "Understanding the Sources of Variation in Software Inspections," Adam Porter, Harvey Siy, Audris Mockus, and Lawrence Votta, *ACM Transactions of Software Engineering and Methodology*, 7(1), January 1998, pp. 41–79.