VERSION 5.2

16 CRITICAL SOFTWARE PRACTICES FOR PERFORMANCE-BASED MANAGEMENT

Purpose

This paper outlines the 16 Critical Software Practices that serve as the basis for implementing effective performance-based management of software-intensive projects. They are intended to be used by programs desiring to implement effective high-leverage practices to improve their bottom-line measures—time to fielding, quality, cost, predictability, and customer satisfaction—and are for CIOs, PMs, sponsoring agencies, software project managers, and others involved in software engineering.

The "16 Critical Software Practices for Performance-based Management" and Templates contain the 16 practices (9 best and 7 sustaining) that are the key to avoiding significant problems for software development projects. These practices have been gathered from the crucible of real-world, large-scale, software development and maintenance projects. Together they constitute a set of high-leverage disciplines that are focused on improving a project's bottom line. This document is intended to define the essential ingredients of each best and sustaining practice. These practices are the starting point for structuring and deploying an effective process for managing large-scale software development and maintenance. They may be tailored to the particular culture, environment, and program phases of a program. Of course, these practices cannot help "death march" programs that are expected to deliver under impossible schedule deadlines with inadequate funding and without the required staffing with essential skills.

The 16 practices are more effective and efficient when there is an organizational infrastructure that facilitates their implementation. This infrastructure termed the "organizational overlay", addresses important aspects such as organizational commitment and training. The organizational overlay will be addressed separately.

PROJECT INTEGRITY

1. Adopt Continuous Program Risk Management

Practice Essentials:

1. Risk management is a continuous process beginning with the definition of the concept and ending with system retirement.
2. Risk management is a program responsibility impacting on and supported by all organizational elements.
3. All programs need to assign a risk officer as a focal point for risk management and maintain a reserve to enable and fund risk mitigation.
4. Risk need to be identified and managed across the life of the program.
5. All risks identified should be analyzed, prioritized—by impact and likelihood of occurrence—and tracked through an automated risk management tool.
6. High-priority risks need to be reported to management on a frequent and regular basis.
7. The culture of all stakeholders must support the risk process (reporting, incentives, award fee plan, fee distribution, resources).
8. For medium and high risks, a risk control profile should be developed.
9. The risk identification process should be a well-planned, well-documented, and tracked process. Metrics should be used to determine if the risk process is identifying risks and if these risks are being used in decision-making.

Implementation Guidelines:

1. Risk management should commence prior to contract award and shall be a factor in the award process.
2. The DEVELOPER needs to establish and implement a project Risk Management Plan that, at a minimum, defines how points 3 through 8 will be implemented. The plan and infrastructure (tools, organizational assignments, and management procedures) will be agreed to by the ACQUIRER and the DEVELOPER and need to be placed under configuration management (CM).
3. DEVELOPER and ACQUIRER senior management should establish reporting mechanisms and employee incentives in which all members of the project staff are encouraged to identify risks and potential problems and are rewarded when risks and potential problems are identified early. The ACQUIRER needs to address risk management explicitly in its contract award fee plan, and the DEVELOPER needs to provide for the direct distribution to all employees in furtherance of establishing and maintaining a risk culture.
4. Risk identification should be accomplished in facilitated meetings attended by project personnel most familiar with the area for which risks are being identified. A person familiar with problems from similar projects in this area in the past should participate in these meetings when possible. Risk identification should include risks throughout the life cycle in at least the areas of cost, schedule, technical, staffing, external dependencies, supportability, and maintainability and should include organizational and programmatic political risks. Risk identification need to be updated at least monthly. Identified risks should be characterized in terms of their likelihood of occurrence and the impact of their occurrence. Risk mitigation activities need to be included in the project's task activity network.
5. Both the DEVELOPER and the ACQUIRER should designate and assign senior members of the technical staff as risk officers to report directly to their respective program managers and should charter this role with independent identification and management of risks across the program and grant the authority needed to carry out this responsibility.
6. Each medium-impact and high-impact risk should be described by a complete Risk Control Profile.

7. Periodically updated estimates of the cost and schedule at completion should include probable costs and schedule impact due to risk items that have not yet been resolved.
8. The DEVELOPER and ACQUIRER risk officers need to update the risk data and database on the schedule defined in the Risk Management Plan. All risks intended for mitigation and any others that are on the critical path and their status against the mitigation strategy should be summarized. Newly identified risks should go through the same processes as the originally identified risks.
9. Risk management should commence prior to contract award and be considered for inclusion as a factor in the award process.
10. Learn from the mistakes of others – today's problem is tomorrow's risk.
    - One common Risk area is subcontractor management.
    - Another common risk area is inadequate planning.
    - A significant risk in projects is miscommunication.
    - Another Risk area is lack of focus.
11. Schedule risk-resolution tasks with sufficient slack to implement a workaround.
12. The supplier/developer should propose, establish, and implement a project Risk Management Plan that, at a minimum, defines how the next six points will be implemented. The plan and infrastructure (tools, organizational assignments, and management procedures) should be agreed to between the acquirer and the supplier/developer and placed under configuration management (CM).
    a. Supplier/developer and acquirer senior management should establish reporting mechanisms and employee incentives in which all members of the project staff are encouraged to identify risks and potential problems and are rewarded when risks and potential problems are identified early. The acquirer should address risk management explicitly in its contract award fee plan when risk is a major or significant consideration, and the supplier/developer should provide for the direct distribution of all non-trivial risk data to all employees in furtherance of establishing and maintaining a risk management culture.
    b. Risk identification should be updated at least monthly.
    c. Risk mitigation activities should be included in the project's task activity network.
    d. Both the supplier/developer and the acquirer should designate a senior member of the technical staff as a risk officer to report directly to his/her respective program manager and charter this role with independent identification and management of risks across the program and grant the authority needed to carry out this responsibility.
    e. Each medium-impact and high-impact risk will be described by a complete Risk Control Profile, which should include four basic items: definition of risk index, which is impact likelihood; transition scenario, which includes initiating key pivotal events and likely risk occurrence; potential mitigation strategies; and key action triggers tied to metrics that initiate various mitigation activities.
    f. The supplier/developer and acquirer risk officers will update the risk data and the integrated supplier/developer database on the schedule defined in the Risk Management Plan. All risks intended for mitigation and any others that are on the critical path and their status against the mitigation strategy will be summarized and reported, and an agreement will be reached on planned action and mitigation. Newly identified risks should go through the same processes as the originally identified risks.
    g. One of the most significant risks a project faces is miscommunication. Effective planning is a necessary step for minimizing this risk. At the highest level this starts with a clearly defined, coordinated and approved statement of work (SOW). The SOW sets the project framework by identifying the scope of work, goals and objectives (implementation, quality, maintenance), identifying users and operational environment, standards that must be followed, project cost and schedule constraints, dependencies on outside agencies, and resources. The SOW becomes a key source document in establishing high level program risks.
13. Planning is a critical step in the project process because it outlines and organizes the activities needed for success. One common risk is inadequate planning. To minimize this risk, there should be a documented procedure for developing a plan.

Items to be addressed by the procedure include the parties involved in planning, the types of plans needed, the form and format, and the plan contents. Guidance for plan contents typically includes the fact that the plan should be based on and conform to customer standards and needs, project standards and needs, the SOW, and project requirements. Planning is conducted throughout the life of the project from acquisition/solicitation through system removal. Planning should include all direct and support activities needed (e,g QA, CM,).

14. Management must remain actively involved in the project. Periodically the project should be reviewed with senior management (of all participants). Items to be covered include technical/cost/schedule performance, staffing, Issues, Risks. These reviews should be formal with minutes and action items. The PM should hold similar. more frequent reviews, based on program milestones or events. All participants should participate. Progress should be tracked against the SOW and requirements. Dependencies should be focused on. Issues should be addressed. Risks should be addressed. This review also should be formal with minutes taken and action items tracked. Management reviews are conducted throughout the life of the project, from acquisition through removal.

15. When a sub is involved, they should follow the same oversight, and management principles as the prime, though the scope may be scaled.

16. One common Risk area is subcontractor management. To minimize risks in the area the following should be accomplished: A subcontract SOW is prepared/reviewed/agreed upon revised when needed, and that a plan to select the contractor is prepared concurrently with the subcontract SOW. Further, the software subcontractor should be selected based upon an evaluation of the subcontractor's ability to do the work. Selection should follow a documented procedure covers the evaluation of proposals submitted, prior performance on similar work, location of the sub relative to the prime, software engineering and management capabilities, staff available, prior experience, and available resources (facilities, hardware, software, training).

17. One common Risk area is subcontractor management. To minimize risks in the area the following should be accomplished: The agreement between the prime and the sub should be the basis for managing the contract (SOW, Terms and Conditions, Requirements for the products to be produced, Dependencies between the sub and the prime, products to be delivered, conditions under which revisions to products will be submitted, acceptance procedures and criteria, and procedures to be used for monitoring the sub). The sub should produce an SDP for prime review and approval. The plan should cover the appropriate items from the prime's SDP.

18. Planning is a critical step in the project process because it outlines and organizes the activities needed for success. One common risk is inadequate planning. To minimize this risk, the planning process should coordinate dependent activities. It should also address support of the software team in other support activities and delineate budgets/resources. The plan should be reviewed and approved by all program participants at a senior level; The SDP which results should cover all activities needed to achieve project success. Items to be included are: Project purpose, scope, goals, and objectives, Project life cycle, Project procedures, methods and standards, Software products to be developed (including interim), Size estimates of products, Estimates of project cost and effort, Estimates for use of critical assets, Milestones and schedules, Identification and assessment of software risks and the risk process to be followed, and plans for necessary tools, facilities and training.

19. Planning is a critical step in the project process because it outlines and organizes the activities needed for success. One common risk is inadequate planning. To minimize this risk, Planning for support activities (e.g. SQA) should be as rigorous as other activities. For SQA, there should be a plan that covers the responsibilities and authority of SQA, identifies the SQA resources, Identify the funding for SQA, Identify SQA's role in establishing an SDP, standards and procedures, identifies evaluations to be accomplished, identifies audits to be performed, identify the process and standards to be followed in reviews/audits, Identify the procedures to be used for recording and tracking problems, Identify the records SWA must keep, and identify the frequency/method of providing feedback to software engineering groups.

20. A significant risk in projects is miscommunication. A documented plan (SDP, etc) is an excellent communication tool. The plan provides baseline schedule, contractual commitments, assignment of responsibilities, coordinates activities between groups, should be available to all team members, reflects inter-group commitments and dependencies, is kept current, and is reviewed by all team members and approved by the PM.

21. The project should have a mechanism for identifying, negotiating, and tracking dependencies between organizations. Each dependency should be identified, each should be negotiated, need dates and delivery formats should the resolved, agreements should be documented and approved by all parties. When groups cannot resolve an issue, there should be a documented process for resolving (e.g. elevate to managers)

22. Project planning includes developing and documenting, in the SDP, project software processes tailored from the organization's standard. The process accounts for all life cycle phases, and contractual waivers. Process plans are reviewed by Quality and management and approved by senior management. Changes to the software process are managed and controlled with reviews and management approvals.

2. Estimate Cost and Schedule Empirically

Practice Essentials:

1. Initial software estimates and schedules should be looked on as high risk due to the lack of definitive information available at the time they are defined.
2. The estimates and schedules should be refined as more information becomes available.
3. At every major program review, costs-to-complete and rescheduling should be presented to identify deviations from the original cost and schedule baselines and to anticipate the likelihood of cost and schedule risks occurring.
4. All estimates should be validated using a cost model, a sanity check should be conducted comparing projected resource requirements, and schedule commitments should be made.
5. Every task within a work breakdown structure (WBS) level needs to have an associated cost estimate and schedule. These tasks should be tracked using earned value.
6. All costs estimates and schedules need to be approved prior to the start of any work.
7. A software size estimate should be an input to the cost estimate. The size estimate should include an estimate of the equivalent new development lines of source code for planned legacy reuse, commercial-off-the-shelf (COTS) and government-off-the-shelf (GOTS) software.
8. When COTS application software is involved, an analysis of the fit of the COTS to the current application should be made prior to cost estimation.
9. The schedule should not be unreasonably compressed. If experience shows that similar projects have taken a given length of time, the new project should not be expected to complete in less than 85 percent of that time.
10. Schedules and cost estimates are influenced by a number of factors, not all of which are under a program manager's control. Do not expect an estimate to be extremely accurate. A good estimation technique will be within 20 percent for cost and 5 percent for schedule.

Implementation Guidelines:

1. Estimate the cost, effort, and schedule for a project for planning purposes and as a yardstick for measuring performance (tracking). Software size and cost need to be estimated prior to beginning work on any incremental release.
2. Software cost estimation should be a reconciliation between a top-down estimate (based on an empirical model; e.g., parametric, cost) and a bottom-up engineering estimate.
3. Software cost estimation should also be subjected to a “sanity check” by comparing it with industry norms and specifically with the DEVELOPER’s past performance in areas such as productivity and percentage of total cost in various functions and project phases.
4. All of the software costs need to be associated with the appropriate lower-level software tasks in the project activity network. Allocate the estimated total project labor effort among all the tasks in the activity network
5. Productivity estimate should be a function of software size and type of software: Real-time, Engineering systems, Business systems.
6. Productivity estimate should be based on productivity metrics from past projects of the performing organization using the planned processes and technology.
7. Productivity estimate should be reduced for new people hired into the organization.

8. Estimate should be adjusted for experience base (for example: new application -- no valid comparison available, no experts with experience in application).
9. Estimate the cost of code reuse by first identifying the specific code modules to be reused and then estimating the percentage of the reuse code in the application that is new or modified.
10. Explicitly identify all source code modules to be reused and estimate the amount of code modification and new code to write to integrate each identified module.
11. Estimating cost and schedule should be consistent with organizational policy and procedures. Likewise the evaluation of cost and schedule estimates for software acquisition should be consistent with organizational policy and procedures.
12. Subject cost estimate to sanity-check rules of thumb: percentage of total cost for individual phases and functions, productivity computed from size and labor effort estimates, and cost estimate for the effort to modify and integrate reuse code, industry norms.
13. Need system perspective (hardware interfaces, sensitivity to operational requirements)
14. There should be a documented procedure for estimating costs and schedules to ensure consistency. This procedure should be adapted over time based on the experience from the project. The procedure should call for estimation of detailed work packages, based on historical data, documenting assumptions and review/approval of estimates.
15. There should be a documented procedure for estimating use of critical assets. The procedure should call for assets to be identified, the estimates to be based on the software products to be used, traffic and the operations concept. The estimates should be reviewed and agreed to by all parties at the system level.
16. Measurements should be made for all tasks to determine cost and schedule status. This includes support tasks such as SQA and SCM.

3. Use Metrics to Manage

Practice Essentials:

1. All programs should have in place a metrics program to monitor issues and determine the likelihood of risks occurring.
2. Metrics should be defined as part of definition of process, identification of risks or issues, or determination of project success factors.
3. All metrics definition need to include description, quantitative bounds, and expected areas of application
4. All programs need to assign an organizational responsibility for identification, collection, analysis, and reporting of metrics throughout the program's life
5. Metrics information should be used as one of the primary inputs for program decisions. The metrics program needs to be continuous.
6. Early warning metrics from all organizations involved in the program should be reported to the buyer's PM in monthly status report.
7. Early warning metrics reported to project managers should be based on data current to at least one week before the date the report is submitted.
8. At a minimum, metrics should measure and track schedules, budgets, productivity, defects, quality, and risk. In addition to this basic set, metrics should be added, as necessary, to monitor specific project concerns.

Implementation Guidelines:

1. Every project should have a project plan with a detail activity network that defines the process the team will follow, organizes and coordinates the work, and estimates and allocates cost and schedule among tasks. The plan should be broad enough to include each sub-process/phase. The project plan needs to include adequate measurement in each of these five categories.
    a. Early indications of problems,
    b. Quality of the products,
    c. Effectiveness of the processes,
    d. Conformance to the process, and
    e. Provision of a basis for future estimation of cost, quality, and schedule.
2. Metrics should be sufficiently broad based. Data that provides insight should be collected for each process, sub-process and phase.
3. To use these metrics effectively, thresholds need to be established for these metrics. These thresholds should be estimated initially using suggested industry norms for various project classes. Local thresholds will evolve over time, based upon experience (see 1.e above). Violation of a threshold value should trigger further analysis and decision making.
4. Continuous data on schedule, risks, libraries, effort expenditures, and other measures of progress should be available to all project personnel along with the latest revision of project plans.
5. Process-effectiveness metrics should include defect removal efficiency, percentage of development cost due to rework, defect leakage through inspections, initial size estimate and size of delivered software, initial estimates of changes to reuse/COTS/GOTS code and actual changes made
6. Collect and report at least the early warning metrics from the SPMN Control Panel.
7. Quality metrics should be established at program inception and include code complexity and delivered defect density.
8. Collect and report the number of defects found in each formal inspection and leakage of defects through inspections.
9. Early-warning metrics reported to project managers should be based on data current to at least one week before the date the report is submitted.
10. Metrics data should be easily available to all persons on the development/sustainment team.
11. The metrics selected to be a project's main indicators of health or dysfunction should provide program wide visibility into the project's status and an accurate comparison of progress against the plan.
12. At a minimum, metrics should measure and track schedules, budgets, productivity, defects, quality, and risk
13. In addition to this basic set, metrics should be added, as necessary, to monitor specific project concerns.
14. Continuous data on schedule, risks, libraries, effort expenditures, and other measures of progress will be available to all project personnel (supplier/developer and acquirer) along with the latest revision of project plans.
15. Data to support future cost estimation should include:
    - Actual cost
    - Actual productivity
    - Actual size
    - Percentage of total code that is reuse code
    - Percentage of initial reuse code SLOC that is modified or added code to allow reuse in the new application
    - Type of software (IS, engineering, real-time, hard real-time)
    - Technical and management methods and processes followed

- Average monthly requirements and staff volatility

---

4. Track Earned Value

Practice Essentials:

1. Earned value project management requires a work breakdown structure, work packages, activity networks at every WBS level, accurate estimates, and implementation of a consistent and planned process.
2. Earned value requires each task to have both entry and exit criteria and a step to validate that these criteria have been met prior to the award of the credit.
3. Earned value credit is binary with zero percent being given before task completion and 100 percent when completion is validated.
4. Earned value metrics need to be collected on a frequent and regular basis consistent with the reporting cycle required with the WBS level. (At the lowest level of the work package, the earned value reporting should never be less frequent than 2 weeks).
5. Earned value, and the associated budgets schedules, and WBS elements need to be re-planned whenever material changes to the program structure are required (e.g., requirements, growth, budget changes, schedule issues, organizational change).
6. Earned value is an essential indicator and should be used as an essential metric by the risk management process.
7. When defects are fixed or new requirements implemented on the product of a task for which earned value credit has been given, this work should be in a new rework task.

Implementation Guidelines:

1. Progress towards producing the products should be measured within the designated cost and schedule allocations.
2. THE DEVELOPER should develop and maintain a hierarchical task activity network based on allocated requirements that includes the tasks for all effort that will be charged to the program. All level of effort (LOE) tasks need to have measurable milestones. All tasks that are not LOE should explicitly identify the products produced by the task and have explicit and measurable exit criteria based on these products.
3. No task should have a budget or planned calendar time duration that is greater than the cost and schedule uncertainty that is acceptable for the program. The goal for task duration is no longer than two calendar weeks of effort.
4. Each task that consumes resources needs to have a cost budget allocated to it and the corresponding staff and other resources that will consume this budget. Staff resources should be defined by person hours or days for each labor category working on the task.
5. For each identified significant risk item, a specific risk mitigation/resolution task should be defined and inserted into the activity network.
6. The cost reporting system for the total project needs to segregate the software effort into software tasks so that the software effort can be tracked separately from the non-software tasks.
7. Milestones for all external dependencies should be included in the activity network.
8. Earned value metrics need to be collected for each schedule level and be made available to all members of the DEVELOPER and government project teams monthly. These metrics are: a comparison of Budgeted Cost of Work Scheduled (BCWS), Budgeted Cost of Work Performed (BCWP), and Actual Cost of Work Performed (ACWP). A

comparison of BCWP and ACWP, a Cost Performance Index, a Schedule Performance Index, and a To-Complete Cost Performance Index.

9. The lowest-level schedules should be statused weekly.
10. The high-level schedules should be statused at least monthly.
11. Earned value reports should be based on data that is no more than two weeks old.
12. Document, track, and manage critical path and schedule dependencies according to a defined process.
13. Structure the hierarchical activity network to roll up charges into cost accounts.
14. Try to structure the hierarchical activity network so that task predecessor/successor relationships are only between child tasks of the same parent task.
    - Have always been able to do this
    - Makes activity network less complex
    - Provides structure for repeatable processes
    - Write a description for each task in the activity network that includes:
        - Identification of the task input
        - Identification of the task product(s)
        - Identification of the methods to be followed to convert task input to output products
    - Unambiguous definition of task exit criteria
15. Enter the initial activity network and resources consumed during the project into an automated project management tool. A spreadsheet is not an adequate project management tool.
16. All tasks/activities should be tracked and metrics gathered to determine task effectiveness. This requires both data collection and analysis
17. Task entry and exit criteria should be defined for all tasks
18. Resources should be allocated to each task (labor hours by labor category, resources other than labor)
19. Enter the initial activity network and resources consumed during the project into an automated project management tool (a spreadsheet is not an adequate project management tool)
20. Manage your resources actively
21. Develop accurate resource projections
22. Critical path and schedule dependencies are tracked, documented and managed according to a defined process.
23. Progress Towards Producing Products Measured: Cost v. Schedule
24. Develop a hierarchical Task Activity net
25. Earned Value Metrics for each schedule level & made available monthly of: BCWS, BCWP and ACWP CPI, SPI (Schedule CPI), To-Complete Performance Index (TCPI), which is BAC - BCWP/EAC - ACWP SPI (Schedule performance index) TCCP
26. Use EV Reporting data the is less than two weeks old
27. Segregate Software Tasks to have greater visibility

5. Track Defects against Quality Targets

Practice Essentials:

1. All programs need to have pre-negotiated quality targets, which is an absolute requirement to be met prior to acceptance by the customer.
2. Programs should implement practices to find defects early in the process and as close in time to creation of the defect as possible and should manage this defect rate against the quality target.
3. Metrics need to be collected as a result of the practices used to monitor defects, which will indicate the number of defects, defect leakage, and defect removal efficiency.
4. Quality targets need to be redefined and renegotiated as essential program conditions change or customer requirements are modified.
5. Compliance with quality targets should be reported to customers on a frequent and regular basis, along with an identification of the risk associated with meeting these targets at delivery.
6. Meeting quality targets should be a subject at every major program review.
7. Product quality goals for delivered products should be unambiguously defined at project inception.
    - Understandability
    - Modularity
    - Defect density
    - May have different goals for different types of applications
    - May have different goals for different parts of the same application
8. Source code cyclomatic complexity < 15 for code units should be one understandability goal.
9. Task products should be inspected against task exit criteria and defects fixed or formally deferred before earned value credit is given for the task.
10. From project inception
    - Defects should include defects in requirements.
    - Classify defects by level of severity.
        - Unambiguous definitions of severity levels at project inception
        - Rules about when fixes must be made for each of the severity levels
11. Customer, project managers, and supplier/developer executives achieve a culture that rewards, not penalizes, finding defects when made.
12. Defect metrics should be collected and periodically reported, including:
    - Number of defects found in each inspection and each test
    - For each defect
    - When each defect was created
    - When each defect was found
    - The number of inspections in which the defect was present but not found
    - The number of defects closed and currently open by category of defect: requirements, architecture, detailed design, code, and documentation.

13. CM should manage defects.
    - All verified defect reports turned over to CM
    - CM maintains status of defect closure
    - CM performs version control of products in the developmental and operational baselines
14. Baseline of customer-accepted deliverables is made.
15. Developmental baseline for stability of products shared among the development team is made.
16. Treat tools as a subset of CM (addressed in CMM)
17. Automated tools solve project problems more efficiently than manual techniques.

Implementation Guidelines:

1. The ACQUIRER and the DEVELOPER need to establish quality targets for subsystem software depending on its requirements for high integrity. A mission-critical/safety-critical system may have different quality targets for each subsystem component. System Quality Assurance needs to monitor quality targets and report defects as per the Quality Plan.
2. Quality targets can be under change control and established at the design, coding, integration, test, and operational levels.
3. Quality targets should address the number of defects by priority and by their fix rate.
4. Actual quality or defects detected and removed should be tracked against the quality targets.
5. Periodic estimates of the cost and schedule at completion should be based on the actual versus targeted quality.
6. Use an empirical cost model, such as SLIM, periodically during development to project latent defects and estimate rework remaining to project completion based on defects found to date.
7. Defect removal efficiency for development is computed as the ratio-number of defects found from project inception to delivery: number of defects found from project inception through the first year

---

6. Treat People-as the Most Important Resource

Practice Essentials:

1. A primary program focus should be staffing positions with qualified personnel and retaining this staff through the life of the project.
2. The program should not implement practices (e.g., excessive unpaid overtime) that will force voluntary staff turnover.
3. The staff should be rewarded for performance against expectations and program requirements.
4. Professional growth opportunities such as training should be made available to the staff.
5. All staff members need to be provided facilities, tools, and work areas adequate to allow efficient and productive performance of their responsibilities.
6. The effectiveness and morale of the staff should be a factor in rewarding management.
7. Compensate key software staff as well as competitors for the same level of skill.
8. Personal Satisfaction
9. Provide Adequate Resources

Implementation Guidelines:

1. DEVELOPER senior management needs to work to ensure that all projects maintain a high degree of personnel satisfaction and team cohesion and should identify and implement practices designed to achieve high levels of staff retention as measured by industry standards. The DEVELOPER should employ focus groups and surveys to assess employee perceptions and suggestions for change.
2. DEVELOPER senior management should provide the project with adequate staff, supported by facilities and tools to develop the software system efficiently. Employee focus groups and surveys should be used to assess this adequacy.
3. The training of DEVELOPER and ACQUIRER personnel should include training according to a project training plan in all the processes, development and management tools, and methods specified in the software development plan.
4. The DEVELOPER and the ACQUIRER should determine the existing skills of all systems, software, and management personnel and provide training, according to the needs of each role, in the processes, development and management tools, and methods specified in the Software Development Plan (SDP)
5. Training is an essential element in recruiting and retention of qualified employees. It is also essential to maintaining a high level of productivity. Important training considerations are:
    - The training of supplier/developer and acquirer personnel will be according to a project-training plan in all the processes, standards, development and management tools, and methods specified in the Software Development Plan (SDP). Soft skills such as teamwork should not be overlooked. The supplier/developer and the acquirer will determine the existing skills of all systems, software, and management personnel and will provide training to the supplier/developer staff and acquirer staff involved in monitoring the project activities, according to the needs of each role, in the processes, development and management tools, and methods specified in the SDP.
    - Qualified individuals or organizations will provide training.
    - Training programs will be consistent with organizational and personal objectives and plans by determining training needs, identifying methods and sources of training, and providing waivers when knowledge and skills exist.
    - Training records will be kept concerning course completion vs. individual training plans and project requirements.
    - In an effort to provide quality training, students, instructors, and management will review all courses. The training program will be independently reviewed on a periodic basis to ensure consistency, relevance.
    - When different organizations must work together, tools, training and procedures should be compatible.
6. Exceptionally good performers should receive much larger raises, bonuses, and better promotions than poor performers.
7. Senior management should continuously survey industry to identify personnel practices that are successful for hiring and retaining people with software skills for which market demand is much greater than supply. For example:
    - Telecommuting for mothers of young children
    - Summer jobs for college software majors
8. Companies should pay for training in software skills to career-transition non-software employees into software jobs. Underemployed engineers should be one source.
9. Training per training plan in categories specified in SDP: Soft skills such as teamwork should not be overlooked. Team members, especially leaders, should receive orientation training in the specific methods, processes, and standards used by other groups they must cooperate with and work with in order to better understand how to develop processes and procedures that are compatible and supportive. The training plan should outline al activities needed to realize training, including, resources, procedures for identifying needs, approvals, and alternative training vehicles. The project training plan should feed an organizational plan, which manages and controls the training program as a whole to meet organizational goals as efficiently as possible, The organizational plan should provide mechanism for project input, review, coordination and plan changes.
10. Determine existing skill sets & provide needed training
11. Make professional growth opportunities such as training available to the staff.

12. All staff members should be provided facilities, tools, equipment, and work areas adequate to allow efficient and productive performance of their responsibilities.
13. Companies should pay for training in software skills to career-transition non-software employees into software jobs (underemployed engineers should be one source)
14. Provide staff with the tools to be efficient and productive (software, equipment, facilities, work areas)
15. The training of supplier/developer and acquirer personnel will be according to a project training plan in all the processes, development and management tools, and methods specified in the Software Development Plan (SDP).
16. The supplier/developer and the acquirer will determine the existing skills of all systems, software, and management personnel and will provide training to the supplier/developer staff and acquirer staff involved in monitoring the project activities, according to the needs of each role, in the processes, development and management tools, and methods specified in the SDP.
17. Domain expertise and software engineering experience are the valuable assets. In staffing a team, maximize these characteristics. Organize so that team members with expertise and experience can most readily identity problems and elevate them for correction.
18. It is critical that Management put someone in charge. A software project manager with the necessary background and skill should be identified and assigned responsibility for building an effective team, developing a software development plan and process that is consistent with the systems development process and successfully delivering the project. Management should provide the PM with the assets needed to succeed as well as clear, objective measures of success. The PM should also have the authority necessary to commit the organization.
19. It is critical that staff members have visibility into their accomplishments and the impact it has on the project. The planning process should result in a detailed program work breakdown structure that identifies clear, measurable objectives and responsibility for each task. Each task should be traceable to the overall project objectives. Task managers/teams should be held accountable for their assigned, agreed to tasks.
20. Training will be provided by qualified individuals or organizations.
21. Training programs will be consistent with organizational and personal objectives and plans by determining training needs, identifying methods and sources of training, and providing waivers when knowledge and skills exist.
22. Training records will be kept concerning course completion vs. individual training plans and project requirements.
23. In an effort to provide quality training, all courses will be reviewed by students, instructors, and management. The training program will be independently reviewed on a periodic basis to ensure consistency, relevance.
24. When different organizations must work together, tools, training and procedures should be compatible.
25. Each organization should have a training plan that addresses long and short range training needs based on the organizations needs and objectives. The plan should address what training is needed and when, how training will be obtained (internal/external), funding and resources, standards for instructional materials, schedules for in-house developed materials, training schedules and procedures for selecting people to be trained, registering, maintaining records, evaluation and feedback. For in-house materials, there should be a course description, a documented review process and a process for control/managing materials.
26. Companies should pay for training in software skills to career-transition non-software employees into software jobs (underemployed engineers should be one source)
27. Make professional growth opportunities such as training available to the staff.

.

CONSTRUCTION INTEGRITY

7. Adopt Life Cycle Configuration Management

Practice Essentials:

1. All programs, irrespective of size, need to manage information through a preplanned configuration management (CM) process.
2. CM has two aspects: formal CM, which manages customer-approved baseline information, and development CM, which manages shared information not yet approved by the customer.
3. Both formal and development CM should uniquely identify managed information, control changes to this information through a structure of boards, provide status of all information either under control or released from CM, and conduct ongoing reviews and audits to ensure that the information under control is the same as that submitted.
4. The approval for a change to controlled information must be made by the highest-level organization which last approved the information prior to placing it under CM.
5. CM should be implemented in a centralized library supported by an automated tool.
6. CM needs to be a continuous process implemented at the beginning of a program and continuing until product retirement.
7. A CM manager should be specifically assigned.
8. The CM process together with automated tools used by CM will maintain version and semantic consistency between configuration items (CIs).
9. CM tasks should be integrated with quality assurance (QA) tasks that verify that products input to CM baselines have met their acceptance criteria.
10. CM should monitor and control the delivery and release-to-operation process.

Implementation Guidelines:

1. CM plans need to be developed by the ACQUIRER and the DEVELOPER to facilitate management control of information they own. The CM procedures of the ACQUIRER serve as the requirements for the CM plan that describes and documents how the DEVELOPER will implement a single CM process. This plan should control formal baselines and will include engineering information, reports, analysis information, test information, user information, and any other information approved for use or shared within the program. The CM process should include DEVELOPER-controlled and -developed baselines as well as ACQUIRER-controlled baselines. It should also include release procedures for all classes of products under control, means for identification, change control procedures, status of products, and reviews and audits of information under CM control. The CM plan needs to be consistent with other plans and procedures used by the project.
2. The two types of baselines managed by CM are developmental and formal. Developmental baselines include all software, artifacts, documentation, tools, and other products not yet approved for delivery to the ACQUIRER but essential for successful production. Formal baselines are information/products (software, artifacts, or documentation) delivered and accepted by the ACQUIRER. Developmental baselines are owned by the DEVELOPER while formal baselines are owned by the ACQUIRER.
3. All information placed under CM as a result of meeting task exit criteria need to be uniquely identified by CM and placed under CM control. This includes software, artifacts, documents, commercial off-the-shelf (COTS), government off-the-shelf (GOTS), operating systems, middleware, database management systems, database information, and any other information necessary to build, release, verify, and/or validate the product.
4. The CM process should be organizationally centered in a project library. This library will be the repository (current and historical) of all controlled products. The ACQUIRER and the DEVELOPER will implement an organizationally specific library. The library(s) will be partitioned according to the level of control of the information.

5. All information managed by CM is subject to change control. Change control consists of:
    a. Identification
    b. Reporting
    c. Analysis
    d. Implementation
6. The change control process needs to be implemented through an appropriate change mechanism tied to who owns the information:
    a. Change control boards, which manage formal baseline products.
    b. Interface boards, which manage jointly owned information
    c. Engineering review boards, which manage DEVELOPER-controlled information.
7. Any information released from the CM library should be described by a Version Description Document (Software Version Description under 498). The version description should consist of any inventory of all components by version identifier, an identification of open problems, closed problems, differences between versions, notes and assumptions, and build instructions. Additionally, each library partition should be described by a current version description that contains the same information.
8. Tools are an important component in the development and control processes of software development. Great care should be exercised in the selection, use, maintenance, and configuration control of automated tools. Automated tools have the potential to solve project problem more efficiently than manual techniques
9. The CM process will include supplier/developer-controlled developed baselines as well as acquirer-controlled baselines. It will also include release procedures for all classes of products under control, means for identification, change control procedures, status of products, and reviews and audits of information under CM control. The CM plan will be consistent with other plans and procedures used by the project.
10. There are two different transitions that must be managed by CM: Engineering Transition — transitions or release of informal, non-base lined information between activities or organizations; and Baseline Transition — transitions of formal documentation or products characterized by a formal approval or acceptance and product transfer.

CM identification segregates information to ensure that the proper level of approval is applied to each piece of information (in-process information not yet shared or approved, configuration management information, controlled program information, test information, pre-released information internally approved for release, customer-approved or base lined information

11. The written CM plan should include:
    - Identification of the configuration items in delivered-product and developmental baselines
    - Details of the change-control process for each of the delivered-product and developmental baselines
    - Identification of external interfaces and ICWGs
    - Details of configuration status reporting
    - Details of version control processes
    - Details of the configuration audit processes
    - Details of the use of automated tools by CM
12. The developmental baseline will include:
    - All task output that is input to other tasks
    - Output files and databases of automated tools used for analysis, design, requirements traceability, and document production
    - Test input data, test drivers, and successful test results

- Deficiency (problem, trouble) reports
- Automated tools used for development and COTS products, such as database management systems (DBMSs), that are part of the delivered software
- Operating systems and middleware

13. CM should collect and report the CM churn metric as an indicator of potentially excessive rework.
14. Develop CM Plan
15. Development Baseline & Formal Baseline
16. All info under CM meet exit criteria, uniquely identified and controlled
17. Single Organization Project library which supports the project. The library should support multiple control levels, provide for storage and retrieval of configuration items, provide for sharing and transfer of CI's between groups and control levels, help in the use of product standards, provide for storage and recovery of archive versions of CI's, help ensure the correct creation of correct products from the library, provide for storage and retrieval of SCM records, support production of SCM reports, and provide for the maintenance of library structure and contents (e.g. backup/recovery of files).
18. Develop and monitor execution of a formal Change Control process
19. Identification. Products are identified for CM control following established/documented criteria. Once identified for CM, they are assigned unique identifiers, the characteristics of each CI are specified, the baseline to which the CI belongs is specified, as well as the point when the item will come under CM control and the person responsible for the item is identified.
20. Reporting. Standard reports include SCCB minutes, change request summary, trouble report summary, baseline change summary, CI revision history, baseline status, audit results. As tools improve, much of this information becomes more readily available online. If processes can be adapted to make this information more readily available via on-line access, this capability should be exploited. Other reports/information releases may be needed to accomplish CM's objective of keeping all shared information consistent.
21. Change Mechanism. Changes should be made to baselines according to an organized, planned, and documented procedure. Reviews/tests should be preformed to ensure that changes have not caused unexpected effects on formal baselines, only CI's approved by the SCCB are entered into the formal baseline, and CI's are checked in and out in a manner that maintains the correctness and integrity of the library (e.g. verifying that revisions are authorized, creating a change log, maintaining a copy of the change, archiving the original version)
22. Change Control Boards (formal baseline)
23. Three types of automated tools: narrow spectrum – single task (structured design tool), broad spectrum – across tasks (traceability tool), integrating technology – integrates tools (information management tools)
24. The CM process together with automated tools used by CM will maintain version and semantic consistency between CI’s.
25. The CM process will include supplier/developer-controlled developed baselines as well as acquirer-controlled baselines. The developmental baseline will include: all task output that is input to other tasks; output files and databases of automated tools used for analysis, design, requirements traceability, and document production; test input data, test drivers, and successful test results; deficiency (problem, trouble) reports; automated tools used for development and COTS products such as database management systems (DBMS’s), that are part of the delivered software; and operating systems and middleware.
26. Test information is all test documentation and versions of software released for test, which includes: test scenarios, data, and results, test configurations, test cases, software configurations under test and configuration documentation, patches and unqualified source modifications, special test data, records, and audit trails.
27. The working area for CM includes: workspace for staging documentation and software releases; tools and files used by the project or needed to build or control project information and/or deliverables; project plans, procedures, and reports; schedules, budgets, and CM records; and audit trails, records, and lessons-learned information essential for certification, approval, or acceptance

28. The two types of baselines managed by CM are developmental and formal. Developmental baselines include all software, artifacts, documentation, tools, and other products not yet approved for delivery to the acquirer but essential for successful production. Formal baselines are information/products (software, artifacts, or documentation) delivered and accepted by the acquirer. The supplier/developer owns developmental baselines while the acquirer owns formal baselines.
29. All information managed by CM will be subject to change control. Change control consists of: identification, reporting, analysis, and implementation.
30. All information that is used by more than one person should be under CM control.
31. A CM manager should be specifically assigned
32. Any information released from the CM library will be described by a Version Description Document (VDD) (Software Version Description under MIL-STD-498 and EIA/IEEE J-STD-016). The version description will consist of any inventory of all components by version identifier, an identification of open problems, closed problems, differences between versions, notes and assumptions, and build instructions. Additionally each library partition will be described by a current version description that contains the same information.

---

8. Manage and Trace Requirements

Practice Essentials:

1. Before any design is initiated, requirements for that segment of the software need to be agreed to.
2. Requirements tracing should be a continuous process providing the means to trace from the user requirement to the lowest level software component.
3. Tracing shall exist not only to user requirements but also between products and the test cases used to verify their successful implementation.
4. All products that are used as part of the trace need to be under configuration control.
5. Requirements tracing should use a tool and be kept current as products are approved and placed under CM.
6. Requirements tracing should address system, hardware, and software and the process should be defined in the system engineering management plan and the software development plan.
7. System requirements should
   - include scenarios (sequences of events) that could occur in
   - specify system input that will cause the system to operate under highest stress.
   - include business rules.
   - include specification of all external systems to which there will be an electronic interface.
8. Design will not begin for any design layer until all requirements have been allocated and traced to that layer, including performance, reliability, safety, external interface, and security requirements.
9. If development will be done with incremental releases, a release build plan should be developed at project inception where all existing system requirements are traced into releases.
10. If the incremental release model is to result in evolutionary definition of system requirements, whenever new system requirements are defined, they should be traced into future releases defined by the release build plan.

Implementation Guidelines:

1. The program needs to define and implement a requirements management plan that addresses system, hardware, and software requirements. This plan should be linked to the SDP.
2. All requirements need to be documented, reviewed, and entered into a requirements management tool and put under CM. This requirements information should be kept current.
3. The CM plan should describe the process for keeping requirements data internally consistent and consistent with other project data.
4. Requirements traceability needs to be maintained through specification, design, code, and testing.
5. Requirements should be visible to all project participants.
6. An overall requirement on a successful project is the definition and maintenance of good processes. Process requirements definition starts at the system requirements
   - Management should demonstrate their commitment to software engineering practices by requiring via written policy an SEPG that coordinates development processes and improvement activities.
   - An SEPG should be established and staffed by experienced professionals, with time to devote. All software and systems engineering disciplines should be represented.
   - The SEPG should develop and maintain the standard development process (or coordinate this activity) and all process assets.
   - The organization's standard process is maintained according to a documented process.
   - The process is documented according to organizational standards.
   - Guidelines/criteria for tailoring the standard process are defined & maintained.
   - A software process database is established and maintained.
   - A process library is maintained.
7. Individual requirements should be linked to one or more critical requirements categories such as safety, security, and performance.
8. Requirements volatility metrics should be computed from the requirements traceability database.
9. All requirements documented are reviewed & put in RM tool under CM
10. CM plan describe maintaining Requirements (consistent)
11. Traceability through specification, design, code & testing
12. Develop Requirements Management Plan (linked to SDP)
13. Requirements visible to all
14. Requirements must trace down from system requirements through all derived requirements and layers of design to the lowest-level software and hardware configuration items.
15. Requirements tracing should be done using an automated requirements traceability tool.
16. Trace system requirements down through all derived requirements and layers of design to the lowest level and to individual test cases
17. All requirements will be documented, reviewed, and entered into a requirements management tool and put under CM. This requirements information will be kept current.
18. The program will define and implement a requirements management plan that addresses system, hardware, and software requirements. This plan will be linked to the SDP.
19. Senior management should establish and communicate a formal policy that establishes their commitment to good software engineering practices and project success. Items to be covered include: software requirements will be derived from and

always retain a systems perspective, that all requirements and their allocations will be documented, formally reviewed and approved before they are used as the basis for future work, and that changes to requirements will consistently incorporated into all project plans, products and activities. Further the policy should indicate that project activities will be designed to efficiently and effectively implement the system and software requirements and that formal plans will be used to document program activities. When objectives are not met immediate action will be taken. Plans will be coordinated and approved by participants. Policy should emphasize training - making sure training necessary to perform SDP tasks is provided.

20. Requirements will be reviewed to make sure they are "good" (i.e. testable, clear, properly stated, consistent, and testable)

21. Requirements should be the basis for all program plans and activities; SQA should review/audit the products and activities for all activities to ensure process compliance and product quality - problems identified are assigned to the responsible party for correction and consistent incorporation with other project activities - the process that allowed the failure is reviewed and corrected. Metrics on problems are used to track project wide trends. Both trends and specific problems should be reported to the PM. Corrective actions should be tacked to closure.

22. Senior management should establish and communicate a formal policy that establishes their commitment to good software engineering practices and project success. The policy should also call for coordination and agreement between team members for project activities - when negotiation cannot reach agreement, then issues should be elevated for immediate resolution. W.R.T CM, the policy should specify that responsibility for CM is explicitly assigned, that SCM shall be implemented throughout the project life cycle and integrated with CM at the system level, that CM is to be used for all products, shared information and tools, that projects will have a repository for CM units and records and that SCM baselines and activities should be audited periodically. W.R.T. SQA, the policy should specify that SQA will be used for all projects, provide a reporting avenue for SQA to senior management, and that senior management will regularly review SQA activities.

23. Senior management should establish and communicate a formal policy that establishes their commitment to good software engineering practices and project success. W.R.T. Subcontract mgmt, policy should state that documented standards and procedures will be used to select subcontractors and manage subcontracts, that contracts are the basis for managing the sub, and contractual changes are made by mutual agreement of all parties. SQA should pay specific attention to the following activities: Planning (schedule estimating/planning, activities for reviewing and making project commitments, activities/standards for preparing plans (e.g. SDP, CM, QA, Risk), and content of plans). SQA should share their results with the customer's SQA organization. Recall that SQA activities are activities as well and should be independently reviewed.

24. Management should demonstrate their commitment to software engineering practices by requiring via written policy an SEPG that coordinates development processes and improvement activities. This would include maintaining an organizational focus, periodic assessments of development processes, establishing a consistent process that can be tailored for each project, and proliferation of experiences, tools, and lessons between projects. Mgmt should reflect their commitment via their actions, focus, priorities, promotions, compensation plans and commitment of resources. Mgmt should ensure that improvement objectives support strategic objectives and guide improvement activities. Mgmt policy should also call for a standard software process, tailoring to each project, maintenance of process assets (tools, guides), and organized collection of project results to use in improving the standard process

25. Mgmt should demonstrate their commitment to software engineering practices by requiring via written policy an SEPG that coordinates development processes and improvement activities. This would include maintaining an organizational focus, periodic assessments of development processes, establishing a consistent process that can be tailored for each project, and proliferation of experiences, tools, and lessons between projects. Mgmt should reflect their commitment via their actions, focus, priorities, promotions, compensation plans and commitment of resources. Mgmt should ensure that improvement objectives support strategic objectives and guide improvement activities. Mgmt policy should also call for a standard software process, tailoring to each project, maintenance of process assets (tools, guides), and organized collection of project results to use in improving the standard process

26. An SEPG should be established and staffed by experienced professionals, with time to devote. All software and systems engineering disciplines should be represented. The SEPG should develop an improvement plan that uses information from assessments, defines improvement activities/schedules, defines responsibilities, identifies resources, undergoes peer review/update and is agreed to by the organization's managers. Process improvement activities should be managed as a project, with corresponding discipline. The objective for each activity should be clearly stated (experiment, adoption, etc). The SEPG should develop/improve the organization's standard development process and each project's tailored processes.

Info on development processes & improvement activities should be kept in a database easily accessible by all team members. Tools/processes that prove effective should be transferred to the organization in an organized manner IAW the plan. SEPG activities should be advertised & regularly seek feedback

27. The SEPG should develop and maintain the standard development process (or coordinate this activity) and all process assets.
28. The organization's standard process is maintained according to a documented process which ensures that: the process satisfies all organizational policies, standards, etc., satisfies the policies, standards, etc. normally imposed by customers, state of the art tools and methods are used when appropriate, internal process interfaces are described, interfaces with systems engineering/contract mgmt/documentation support are identified, a process exists for proposing/reviewing/agreeing to/implementing process changes exists, the process undergoes peer review, and is placed under CM
29. The process is documented according to organizational standards and is decomposed into understandable segments, each process is well defined, and process inter-relationships are defined. Approved software life cycle models are identified, The models must be compatible with the organization's systems approach and standard software processes, changes to the models are coordinated/approved/documented in an organized manner
30. Guidelines/criteria for tailoring the standard process are defined & maintained. The guidelines/criteria cover selecting and tailoring the life cycle, the process, and standards for documenting the project's process. Tailoring criteria/guideline are placed under CM control and follow that process for review and update. The SEPG approves the changes.
31. A software process database is established and maintained. The db collects and makes available info on software work products. Data entered is reviewed to ensure integrity. The DB is controlled/managed. User access is controlled to ensure completeness, accuracy and integrity. The data is available to all who need it, however.
32. A process library is maintained. Candidate documents are reviewed prior to inclusion. Documents are catalogued for easy access. The library is kept current (revisions posted). Contents are available to all organizational members/team members. Use of documents is monitored to determine if document is still relevant. The library contents are managed/controlled.

---

9. Use System-Based Software Design

Practice Essentials:

1. All methods used to define system architecture and software design should be documented in the system engineering management plan and software development plan and be frequently and regularly evaluated through audits conducted by an independent program organization.
2. Software engineering needs to participate in the definition of system architectures and should provide an acceptance gate before software requirements are defined.
3. The allocation of system architecture to hardware, software, or operational procedures needs to be the result of a predefined engineering process and be tracked through traceability and frequent quality evaluations.
4. All agreed to system architectures, software requirements, and software design decisions should be placed under CM control when they are approved for program implementation.
5. All architecture and design components need to be approved through an inspection prior to release to CM. This inspection should evaluate the process used to develop the product, the form and structure of the product, the technical integrity, and the adequacy to support future applications of the product to program needs.
6. All system architecture decisions should be based on a predefined engineering process and trade studies conducted to evaluate alternatives.
7. System and software architectures should be developed from the same partitioning perspective to minimize the complexity of requirements allocation/traceability: Information, Objects, Functions, States.

8. Software engineers should participate in system architecture design and design of the client/server network on which the software will execute.
9. The system architecture design should support security, reliability, performance, safety, and interoperability requirements.
10. The system architecture design should support reuse of legacy software and integration of COTS/GOTS software in a way that minimizes modifications and additions to the reuse software.
11. System and software architecture design should give views of static, dynamic, and physical structures.
    - Static views show components that can exist. They contain no temporal information.
    - Static views also include threads of collaboration.
    - Dynamic views show temporal, concurrency, and synchronization behavior. They include interactions in time sequences and sequences of states.
    - The physical view describes the mapping of the software and databases onto the hardware and reflects its distributed aspect.
12. The system architecture should be modeled and simulated before the start of software architecture design to verify support for security, reliability, performance, and safety requirements.

Implementation Guidelines:

1. The DEVELOPER should ensure that the system and software architectures are developed and maintained consistent with standards, methodologies, and external interfaces specified in the system and software development plans.
2. Software engineers need to be an integral part of the team performing systems engineering tasks that influence software.
3. Systems engineering requirements trade studies should include efforts to mitigate software risks.
4. System architecture specifications need to be maintained under CM.
5. The system and software architecture and architecture methods need to be consistent with each other.
6. System requirements, including derived requirements, need to be documented and allocated to hardware components and software components.
7. The requirements for each software component in the system architecture and derived requirements need to be allocated among all components and interfaces of the software component in the system architecture.
8. Software engineers will be an integral part of the team performing systems engineering tasks that influence software. Software engineering personnel should be integral team members in all preliminary project phases. This includes concept exploration, feasibility studies and proposal preparation. This is especially important during the negotiations phases when project commitments are being made
9. When an incremental development life cycle model is followed, the system architecture should be developed as part of the first incremental release to minimize the risk of architecture rework causing major ripple-effect rework.
10. The structured methods for system and software architecture design should be selected first, and then automated tools selected that implement the selected methods.
11. CSCI Architectures is consistent, specified in System & Software Development Plans
12. SW Engineers are part of Systems Engineering
13. Use System Engineering trade studies to mitigate risks
14. System and Software architecture consistent
15. System Requirements & Derived Requirements are allocated to HW & SW
16. Software engineers should participate in system architecture design and design of the client/server network on which the software will execute.

17. Automated CASE tools should be used for system and software architecture design. CASE tools automatically find errors in completeness and consistency. A drawing tool is not a design CASE tool.
18. Planning for software activities should begin at the same time as system planning activities. The CM process should be exploited to keep these parallel efforts consistent. This coordinated planning activity should continue throughout the project life. As part of the coordinated planning a software lifecycle that supports the system objective and schedule will be agreed to and documented. Planning should include direct and support activities (e.g. SQA). All plans should be reviewed and agreed to by all participants and impacted parties.
19. Automated CASE tools should be used for system and software architecture design.
    - CASE tools automatically find errors in completeness and consistency.
    - A drawing tool is not a design CASE tool.

---

10. Ensure Data and Database Interoperability

Practice Essentials:

1. All data and database implementation decisions should consider interoperability issues and, as interoperability factors change, these decisions should be revisited.
2. Program standards should exist for database implementation and for the data elements that are included. These standards should include process standards for defining the database and entering information into it and product standards that define the structure, elements, and other essential database factors.
3. All data and databases should be structured in accordance with program requirements, such as the DII COE, to provide interoperability with other systems.
4. All databases shared with the program need to be under CM control and managed through the program change process.
5. Databases and data should be integrated across the program with data redundancy kept to a minimum.
6. When using multiple COTS packages, compatibility of the data/referential integrity mechanisms need to be considered to ensure consistency between databases.
7. Download and read documents from DoD data administration home page. Not to check off compliance with DoD policy, but to get guidance on how to lower the cost of data interoperability with your externals
8. Information systems should be designed with very loose coupling between hardware, persistent data, and application software.
9. No change to an application software program accessing a database should force a change to any other application software program accessing the same database.
10. Data elements should be identified using several processes.
    - Analysis of business or mission processes using a structured method
    - Definition of persistent data transactions and business rules
    - Development of user interface prototype with participation of operational users
    - Analysis of persistent data needs of all application software to interface the database
    - Reverse engineering of legacy application software may be required
    - A complete inventory of all external interfaces
11. Relationships between data items should be defined based on queries to be made on the database.

12. Data element names, definitions, minimum accuracy, data type, units of measure, and range of values should be defined to minimize the amount of translation required to share data with external systems.
13. Business rules and high-volume transactions on the database should be specified before database physical design begins.
14. Data security requirements, including security level of aggregated data, should be specified before database design begins.

Implementation Guidelines:

1. The DEVELOPER needs to ensure that data files and databases are developed with standards and methodologies.
2. The DEVELOPER needs to ensure that data entities and data elements are consistent with the DoD data model.
3. All data and databases should be structured in compliance with DII COE to provide interoperability with other systems.
4. Data integrity and referential integrity should be maintained automatically by COTS DBMSs or other COTS software packages. The DEVELOPER should avoid developing its package, if at all possible. Before selecting multiple COTS software packages, the DEVELOPER should study the compatibility of the data/referential integrity mechanisms of these COTS packages and obtain assurance from the COTS vendors first.
5. Unnecessary data redundancy should be reduced to minimum.
6. Data and databases should be integrated as much as possible. Except data for temporary use or for analysis/report purposes, each data item should be updated only once, and the changes should propagate automatically everywhere.
7. Quick response to high-volume transactions should be a major consideration in the physical database design of on-line transaction processing (OLTP) databases that support operations.
8. Decision-support databases should be designed to support ad hoc queries and should be separate from OLTP databases.
9. Database design should be completed in the first, or a very early, release in an incremental-release development.
10. Capabilities of the COTS DBMS used to implement the database should be selected with a detailed understanding of which capabilities are proprietary.
11. The ease of linking database queries to the database vendor's product to widgets and fields in screens developed with a GUI-builder tool should be a major factor in the selection of a GUI-builder tool.
12. If databases managed by COTS DBMS from different vendors are to be integrated with a vendor gateway product, carefully determine the proprietary features of the gateway and how it limits database access, particularly to DBMS, from vendors other than the gateway vendor.
13. Design of database physical distribution, replication, and levels of aggregation on a network should be a trade-off between providing current data with quick response to users and reducing the bandwidth needs from the network.
14. When legacy data will be used in a new database structure, plan a task with adequate time and budget to convert legacy data to the new format.
15. This task should include analyzing the legacy data for errors and fixing them
16. All data and databases should be structured in compliance with Defense Information Infrastructure Common Operating Environment (DII COE) to provide interoperability with other systems.
17. The ease of linking database queries to the database vendors product to widgets and fields in screens developed with a GUI-builder tool should be a major factor in the selection of a GUI-builder tool.

---

11. Define and Control Interfaces

Practice Essentials:

1. Before completion of system-level requirements, a complete inventory of all external interfaces needs to be completed.

2. All external interfaces need to be described as to source, format, structure, content, and method of support and this definition, or interface profile, needs to be placed under CM control.
3. Any changes to this interface profile should require concurrence by the interface owners prior to being made.
4. Internal software interfaces should be defined as part of the design process and managed through CM.
5. Interfaces should be inspected as part of the software inspection process.
6. Each software or system interface needs to be tested individually and a test of interface support should be conducted in a stressed and anomalous test environment.
7. The requirements for each external electronic interface should include the items listed in paragraph 3 of the MIL-STD-498 data item description DI-IPSC-81434.
8. The design specification for each external electronic interface should include the items listed in paragraph 3 of the MIL-STD-498 data item description DI-IPSC-81436.
9. Future user participation in the development of a user interface prototype should be a primary structured method for users to define system requirements.
10. Interface Change Concurrence (by owners)
11. Milestones in Project Activity Network (see below for detail)
12. Control Subsystem Interface at Program Level
13. Application external interfaces, interfaces with middleware and operating systems, and interfaces of COTS products integrated into the application should comply with applicable public, open API standards and data interoperability standards unless there is a compelling reason to do otherwise. Comply with JTA interface standards
14. Do not assume that if two interfacing applications comply with the JTA and DII COE TAFIM interface standards that they will interoperate.
15. A standardized API should be used for accessing security mechanisms.
16. APIs are defined primarily in support of application portability.

Implementation Guidelines:

1. All internal and external interfaces need to be documented and maintained under CM control.
2. Changes to interfaces require concurrence by the interface owners prior to being made.
3. Milestones related to external interfaces should be tracked in the project activity network. [Keep these milestones off your critical path.]
4. Subsystem interfaces should be controlled at the program level.
5. Develop user interface prototype used in system requirements definition with a GUI-builder tool in a way that the prototype, with minimal modification, will be the user interface software of the delivered application.
6. The user interface prototype used for system requirements definition should include screens and navigation between screens.
7. Milestones related to external interfaces--such as completion of specifications and completion of system interface-should be included in the project activity network.
8. Compliance with open, public standards for APIs should be a major factor in selecting COTS operating system and middleware products and COTS application and service/utility products.
9. If you plan significant reuse of legacy software, design the software architecture to minimize changes to legacy module interfaces.
10. With a GUI-builder tool, develop the user interface prototype specified in the system requirements definition in a way that the prototype, with minimal modification, will be the user interface software of the delivered application. This user interface prototype should include screens and navigation between screens.

11. Gain a good understanding of middleware, client/server network, and Internet operating system API options and pitfalls.
    - Electronic interfaces connect modules in the architecture
    - Application Programming Interface (API) is defined as the interface between the application software and the application platform
    - External Environment Interface contains the external entities with which the application platform exchanges information.
        - The External Environment Interface (EEI) is the interface between the application platform and the external environment across which information is exchanged.
        - EEI may be divided into these groups: Human/Computer Interaction Services EEI, Information Services EEI, and Communications Services EEI.

---

12. Design Twice, Code Once

Practice Essentials:

1. All design processes should follow methods documented in the software development plan.
2. All designs need to be subject to verification of characteristics, which are included as part of the design standards for the product produced.
3. Completed Architectures in First Release
4. Verify System & Software Architecture per SDP
5. All designs should be evaluated through a structured inspection prior to release to CM. This inspection should consider reuse, performance, interoperability, security, safety, reliability, and limitations.
6. Traceability needs to be maintained through the design and verified as part of the inspection process.
7. Critical components should be evaluated through a specific white-box test level step.
8. Design can be incrementally specified when an incremental release or evolution life cycle model is used provided the CM process is adequate to support control of incremental designs and the inspection process is adapted to this requirement.
9. The software detailed design methods should be consistent with the design methods used for architecture design.
10. End-user functionality should be visible in the software design description.
11. The execution process should be visible in the software design, including performance, availability, concurrency, fault-tolerance, threads of control, and processes (groups of tasks that form an executable unit).
12. Physical software components and their interfaces should be visible in a design description.
13. The mapping of physical software components onto individual hardware components should be visible.
14. States and state transition should be visible in the design description if the system has more than one state.
15. Each of the above views of the software should be documented in a graphical notation.
16. Operational scenarios should be defined that show how the different views of design work together.

Implementation Guidelines:

1. When reuse of existing software is planned, the system and software architectures should be designed to facilitate this reuse.

2. When an incremental release life cycle model is planned, the system and software architectures need to be completed in the first release or, at most, extended in releases after the first without changes to the architecture of previous releases.
3. The system and software architectures will be verified using methods specified in the SDP. This verification will be conducted during a structured inspection of the software architecture and will include corroboration that the architecture will support all reuse, performance, interoperability, security, safety, and reliability requirements. The architecture will be under CM.
4. The structured design method should be selected first, then select an automated tool that enforces the selected method.
5. Each of the design views defined in the universal markup language (UML) notation guide should be considered:
    - Static structure diagrams
    - Use CASE diagrams
    - Sequence diagrams
    - Collaboration diagrams
    - State chart diagrams
    - Activity diagrams
    - Implementation diagrams
6. Design should not be represented in a program design language (pseudocode) because pseudocode affects the partitioning and complexity management reasons for design.
7. Design should be done to minimize complexity and maximize understandability.
8. Design diagrams should be developed with a CASE tool that enforces notation and semantic rules, not with a drawing tool.
9. The approach to design should never be to write code before design and then reverse engineer the design from this code.
10. A programming language should be selected that was developed to implement the design method.

---

13. Assess Reuse Risks and Costs

Practice Essentials;

1. The use of reuse components, COTS, GOTS, or any other non-developmental items (NDI) should be treated as a risk and managed through risk management.
2. Application of reuse components, COTS, GOTS, or any other NDI will be made only after successful completion of a NDI acceptance inspection. This inspection needs to consider the process used to develop it, how it was document, number of users, user experience, and compliance with essential program considerations such as safety or security.
3. Before a decision is made to reuse a product or to acquire COTS, GOTS, or NDI, a complete cost trade-off should be made considering the full life cycle costs, update requirements, maintenance costs, warranty and licensing costs, and any other considerations which impact use of the product throughout its life cycle. [Avoid need for costly development of "wrappers" to translate reuse software external interfaces]
4. All reuse products, COTS, GOTS, or NDI decisions should be based on architectural and design definitions and be traceable back to an approved user requirement.
5. All reuse components, COTS, and COTS need to be tested individually first against program requirements and in an integrated software and system configuration prior to release for testing according to the program test plan.
6. Reuse, COTS, GOTS, and NDI decisions will be continuously revisited as program conditions change
7. Establish quantified selection criteria and acceptability thresholds

- Fit of functionality to current application
- Compliance of external and system software interfaces with external engineering interface, API, and data-interoperability standards
- Interfaces with your operating systems and middleware
- Reuse architecture to which it was designed
- Quality
- Performance under stress of current application
- Range of values for input variables
- Security
- Quality of documentation

8. It is almost certain that there will be proprietary features even when the COTS are marketed as "open." If your application will operate on a client/server network, analyze the ease of distributing the COTS product and its output data on a network
9. Develop code for future reuse to integrate into an architecture framework that is well defined and is supported by many software companies
    - There are two COTS architectural frameworks that dominate in commercial markets:
        - CORBA/JavaBeans
        - ActiveX/DCOM
    - Consider more stringent development process for code developed for future reuse than for code developed only for the current application
    - Estimate the additional cost of development for future reuse for input to the decision to develop for reuse
10. Reverse engineer legacy software when developing new system to replace legacy software
    - Requirements
    - Algorithms
    - Functions
    - Business rules
    - Documentation likely not to be consistent with code
    - Much internal to software likely invisible to users
11. Reverse engineer with an automated tool used by a person who can read source code and who understands the domain
12. Establish Reuse Plan
13. Reuse plan in SDP, Evaluate Reuse vs. System Requirements
14. Per Plan, System Engineering Process identifies software to reuse
15. Identify Reuse in Test Plan
16. Cost Estimate of Integration and licenses over Life Cycle

Implementation Guidelines:

1. The DEVELOPER will establish a reuse plan for the integration of COTS, GOTS, and in-house software. This plan needs to include discussion and allocation of whom and by what process reused software code is tested, verified, modified, and maintained.

2. The reuse plan should be in the SDP and document an approach for evaluating and enforcing reused functionality against system requirements.
3. The reuse plan should suggest a system engineering process that identifies software requirements by taking existing, reusable software components into account.
4. The test plan should identify the testing of the integrated reused code.
5. When integrating COTS, GOTS, and in-house software, ensure accurate cost estimation of integrating the reused code into the system. The cost of integrating unmodified reused code is approximately one-third the cost of developing code without reuse.
6. The DEVELOPER and the ACQUIRER need to be able to plan for the estimated costs of obtaining the necessary development and run-time licenses over the system's life cycle and the maintenance/support critical to the product, including source code availability.
7. Define your reuse software selection criteria in writing.
8. Talk with users of the COTS or GOTS product before selection.
9. Ask the ones the vendor referenced.
10. Find your own references independent of vendor-provided references.
11. Talk to users with similar percentage fit of the COTS product to their application and similar operational environment.
12. Try and get hard data on the productivity achieved using tools provided by the COTS vendors to modify their COTS products.
13. It will cost organizations that normally develop world-class quality software about 30 percent more to develop code for future reuse.
14. It can cost the average shop more than by a factor of 4 to develop code for future reuse.
15. Reverse engineer legacy software when developing a new system to replace legacy software (requirements, algorithms, functions, business rules, documentation likely to not be consistent with code, much internal to software likely invisible to users, reverse engineer with an automated tool used by a person who can read source code and who understands the domain)

PRODUCT STABILITY AND INTEGRITY

14. Inspect Requirements and Design

Practice Essentials:

1. All products that are placed under CM and are used as a basis for subsequent development need to be subjected to successful completion of a formal inspection prior to its release to CM.
2. Goal of 80% defect detection
3. No CM until satisfaction of Peer Review
4. Metrics on defects
5. Exit criteria for non-LOE earned value as gates for increasing CM
6. Structured architecture inspection
7. The inspection needs to follow a rigorous process defined in the software development plan and should be based on agreed-to entry and exit criteria for that specific product.

8. At the inspection, specific metrics should be collected and tracked which will describe defects, defect removal efficiency, and efficiency of the inspection process.
9. All products to be placed under CM should be inspected as close to their production as feasible.
10. Inspections should be conducted beginning with concept definition and ending with completion of the engineering process.
11. The program needs to fund inspections and track rework savings
12. System and software architectures will, in addition to formal inspections, be modeled and simulated to verify that the architecture supports performance, reliability, security, and safety requirements.
13. Formal inspections and architecture modeling/simulation will be conducted in accordance with detailed instructions contained in the SDP.
14. Start now and get the team trained, then go and do it.
15. There is no reason to wait for the right time to employ inspections in a product.
16. Begin immediately. Even if the project is halfway through the testing stage, the team can begin inspecting fixes to defects discovered during test.
17. The payoff is immediate.
18. No sooner will inspections be started than defects will begin to be removed.
19. These would only have been found later at a higher cost.
20. Modeling and simulation of architecture should be done with automatic tools, verify the architecture will support performance requirements, and use input that represents stress loads that might occur during operation.

Implementation Guidelines:

1. The DEVELOPER will implement a formal, structured inspection/peer review process that begins with the first system requirements products and continue through architecture, design, code, integration, testing, and documentation products and plans. The plan needs to be documented and controlled as per the SDP.
2. The project should set a goal of finding at least 80 percent of the defects in every product undergoing a structured peer review or other formal inspection.
3. Products should not be accepted into a CM baseline until they have satisfactorily completed a structured peer review.
4. The DEVELOPER needs to collect and report metrics concerning the number of defects found in each structured peer review, the time between creating and finding each defect, where and when the defect was identified, and the efficiency of defect removal.
5. Successful completion of inspections should act as the task exit criteria for non-Level-of-Effort earned value metrics (and other metrics used to capture effectiveness of the formal inspection process) and as gates to place items under increasing levels of CM control.
6. The DEVELOPER should use a structured architecture inspection technique to verify correctness and related system performance characteristics.
7. Implement six well-defined inspection steps:
   - Planning
   - Defining and verifying inspection entry criteria
   - Arranging the availability of the right participants
   - Overview
   - Education of participants in what is to be inspected
   - Assignment of roles to participants

  - Preparation--participants learn the material
  - Inspection--find defects but do not look for fixes
  - Rework--product author fixes defects
  - Follow up--verification fixes made and no new defects

8. Participation in inspection should be limited to 2-hour periods with no more than two 2-hour inspection periods per day.
9. Inspection training should be comprehensive
   - Three 3-hour sessions
   - Video tape of actual inspection
   - Students observe actual inspection
10. Architecture modeling and simulation should be done using COTS modeling and simulation products.
11. Inspections should be conducted on small products that can be completely understood by an individual.
12. Modeling and simulation of architecture should:
    - Be done with automatic tools
    - Verify the architecture will support performance requirements
    - Use input that represents stress loads that might occur during operation.

---

15. Manage Testing as a Continuous Process

Practice Essentials:

1. All testing should follow a preplanned process, which is agreed to and funded.
2. Every product that is placed under CM should be tested by a corresponding testing activity.
3. All tests should consider not only a nominal system condition but also address anomalous and recovery aspects of the system.
4. Prior to delivery, the system needs to be tested in a stressed environment, nominally in excess of 150 percent of its rated capacities.
5. All test products (test cases, data, tools, configuration, and criteria) should be released through CM and be documented in a software version description document.
6. Every test should be described in traceable procedures and have pass-fail criteria included.
7. Concurrent with test planning: analyze code to identify potential lockups, non-developers conduct unstructured “free-play” test
8. Software development projects will deliver test products to minimize the cost of regression test during sustainment.
9. Deliverable test products will undergo formal inspections.
10. GPO will plan the test process & document this plan with test cases and detailed descriptions. Include use cases re operational mission scenarios
11. Consistent with RFP, award fee incentives

Implementation Guidelines:

1. The testing process must be consistent with the RFP and the contract. The award fee should incentivize implementation of the testing practices described below.
2. The ACQUIRER and DEVELOPER need to plan their portion of the test process and document this plan with test cases and detailed test descriptions. These test cases should use cases based on projected operational mission scenarios.
3. The testing process should also include stress/load testing for stability purpose (i.e., at 95% CPU use, system stability is still guaranteed….)
4. The test plan should include a "justifiable testing stoppage criteria." This gives testers a goal. If your testing satisfies these criteria, then the product is ready for release.
5. The test process should thoroughly test the interfaces between any in-house and COTS functionality. These tests should include timing between COTS functionality and the bespoken functionality. The test plans need to pay serious attention to how to demonstrate that, if the COTS software fails, how to test that the rest of the software can recover adequately. This involves some very serious stress testing using fault injection testing.
6. Software testing should include a traceable white-box and other test process verifying implemented software against CM-controlled design documentation and the requirements traceability matrix.
7. A level of the white-box test coverage should be specified that is appropriate for the software being tested.
8. The white-box and other testing should use automated tools to instrument the software to measure test coverage.
9. All builds for white-box testing need to be done with source code obtained from the CM library.
10. Frequent builds require test automation, since more frequent compiles will force quick turnaround on all tests, especially during regression testing. However, this requires a high degree of test automation.
11. A black-box test of integration builds needs to include functional, interface, error recovery, stress, and out-of-bounds input testing.
12. Reused components and objects require high-level testing consistent with the operational/target environment.
13. Software testing includes a separate black-box test level to validate implemented software. All black-box software tests should trace to controlled requirements and be executed using software built from controlled CM libraries.
14. In addition to static requirements, a black-box test of the fully integrated system will be against scenarios—sequences of events designed to model field operation.
15. Performance testing for systems (e.g., performing 10,000 tests/second still yields response times under 2 seconds) should be tested as an integral part of the black-box test process.
16. Test to design documentation & requirements in CM
17. Use Automated tools
18. CM source code for builds for white box
19. Black box test of integration builds
20. Black box tests to model field operation as well as static requirements
21. Independent audits & reporting to GPO
22. Each test to include pass/fail criteria
23. Ensure Test Planning is accomplished
24. Automated tools should be used to instrument code to measure path coverage for white-box tests.
25. Automated tools should be used to automatically record key strokes and mouse clicks that execute a test and then playback this recording to execute regression tests and compare test results with correct results. All builds for white-box testing will be done with source code obtained from the CM library.

26. An independent QA team should periodically audit selected test cases, test traceability, test execution, and test reports providing the results of this audit to the ACQUIRER. (The results of this or similar audits may be used as a factor in the calculation of Award Fee.)
27. The implementation of test levels to a specific program must be adapted to the specific program strategy used as per DODI 5000.2 and 8120.2i.e., Grand Design, Incremental Development (Preplanned Product Improvement), or Evolutionary Strategies.
28. Deliverable test-related products should include:
    - Test cases
    - Requirements traceability to individual test cases
    - Test descriptions
    - Test drivers
    - Test input data
    - Results from successful tests
    - Test tools
29. An integration-test build plan should be developed that identifies the sequence of integration builds and testing to accomplish one or more of the following:
    - Minimize the amount of test driver software that has to be developed
    - Early test of high-risk code
    - Early test of safety- or security-critical code
    - Minimize duration of integration test

Automated tools should be used to instrument code to measure path coverage for white-box test.

---

16. Compile and Smoke Test Frequently

Practice Essentials:

1. All tests should use systems that are built on a frequent and regular basis (nominally no less than twice a week).
2. All new releases should be regression tested by CM prior to release to the test organization.
3. Smoke testing should qualify new capability or components only after successful regression test completion.
4. All smoke tests should be based on a pre-approved and traceable procedure and run by an independent organization (not the engineers who produced it).
5. All defects identified should be documented and be subject to the program change control process.
6. Smoke test results should be visible and provided to all project personnel.
7. Integration build will be done by CM and will include preparation of a build VDD that identifies the software unit versions in the build and the open and fixed defects against the build.
8. Integration test will execute CM-controlled test descriptions.
9. CM will monitor the status of fixing defects and will track the software versions in which each defect is first fixed.

Implementation Guidelines:

1. From the earliest opportunity to assess the progress of developed code, the DEVELOPER needs to use a process of frequent (one- to two-week intervals) software compile-builds as a means for finding software integration problems early.
2. It is required that a regression facility that incorporates a full functional test suite be applied with the build strategy.
3. Results of testing of each software build should be made available to all project personnel
4. The project must mitigate the risk of losing the benefit of frequent builds.
5. The project must have a firm, enforced standard for what constitutes an acceptable process and product.
6. The project enforced standard needs to set a quality level that's strict enough to keep critical defects out but lenient enough to disregard trivial defects.
7. Inspections should monitor, on a more frequent schedule than the build cycle, product quality.
8. CM, not engineering, should own products approved for the next build.
9. From controlled libraries, CM: compiles all files, libraries, and other components successfully; links all files, libraries, and other components successfully; ensures through inspection and test records that components do not contain any showstopper bugs that prevent the program from being executed or that make it hazardous to operate; ensures that the build passes the smoke test.
10. It doesn't matter how many or few components are replaced.
11. The smoke test should exercise the entire system from end-to-end. It need not be exhaustive.
12. Focus of smoke test is exposing problems and defects, not proving system or software works.
13. The smoke test should be thorough enough that, if the build passes, you can assume that it is stable enough to support subsequent test levels.
14. Smoke tests require adequate time to run and sufficient time to analyze.
15. Process requires substantial automation (simulation, emulation, instrumentation, analysis).
16. Require developers to execute all critical paths through the code
17. Concurrent with test planning: analyze code to identify potential lockups, non-developers conduct unstructured "free-play" test
18. Binary patches will never be made to code during integration test.
19. When any change is made to source code from the developmental baseline during frequent compile and test, the version number of the source code will be changed and the modified code turned in to CM.
20. CM will ensure that every software unit version includes all the defect fixes made to all earlier versions.
21. Automation should be used in the integration build process so that most of the time between successive integration tests is dedicated to integration tests.
22. Integration test should migrate to the same hardware/system software environment that will be used in the field.