**World Class Software Organizations**

**by**

**David S Burris**

***Ph.D. in Computing Science***

***Certified Computer Programmer***

***Certified Systems Professional***

***Certified Computing Professional***

csc\_dsb@shsu.edu

This material summarizes the ideas of multiple authors. Permission is granted to students in my classes to use any or all of the material in this document for their personnel educational in conjunction with course textbooks. All other rights reserved. Materials may not be transmitted to individuals not registered in my classes.

Updated: Nov 29, 2007

***World-Class Software Organizations***

**Key issues:**

**1) Recognition that *enterprise productivity* is much more important than *project productivity or programmer productivity*. This will require measuring aspects of productivity ignored by most organizations today:**

**turnover,**

**training (just-in-time) ,**

**idle programmer time between projects,**

**systems that are scrapped before (or after) they are developed,**

**and so forth.**

**2) A *capital investment* will be required notoriously lacking in organizations that budget DP activities on a year-to-year, quarter-to-quarter basis(1%-3% of enterprise budget is insufficient):**

**for programming tools,**

**development environments,**

**CASE tools,**

**re-engineering tools,**

**maintenance tools,**

**and so forth.**

***Fundamental questions must be raised and answered:***

**What do world-class software organizations do differently from my organization?**

**What tools do they use?**

**What procedures, methods, and techniques do they use?**

**How long term is their investment?**

**Does your company *care about software quality*, enough for example to delay release of a new product that production does not yet feel meets quality standards when the customer says they are happy, willing to pay, and want it now? Does your company even employ software reliability models to determine what is acceptable?**

**Does your company *care about its people*. Does it invest time and money every year for staff training, to educate managers to do a better job of hiring *quality* people? *Does it assume that its software people are replaceable commodities, not worth any investment?* Does it communicate organizational goals and employ techniques to ensure that corporate goals are aligned with the personal consequences of these goals?**

**Does your company *employ modern analysis, design, and program development techniques and tools?***

**Does your company *measure everything* in the software arena, not just the *product* but the *process* as well? Is there a separate software metrics group. Are its metrics available to everyone and are they used in a positive manner?**

**Does your company *support the concept of software reusability?* *More importantly, does it provide incentives for reuse to its engineers to encourage creation of reusable components as well as for their reuse?* Does it base project estimates on reuse rather than checking after completion?**

**Does your company provide adequate hardware and software for all its employees?**

**Is it running scared, as if its very existence is at stake?**

***If the answer to a majority of these question is not "YES," then consider changing the company, or updating your resume and "voting with your feet." It may be time to move on.***

**Edward Yourdon, "Decline & Fall of the American Programmer," Prentice Hall, 1992, pp. 18-20, ISBN 0-13-203670-3.**

**A SOFTWARE PROCESS MATURITY FRAMEWORK**

**A mature Software Process must be *predictable*, cost estimates and schedule commitments must be met with reasonable consistency, and the resulting product should generally meet the user's functional and quality expectations.**

**The *Software Process* is a set of *tools*, *methods* and *practices* we use to produce a software product. The *objectives or goals of software process management* is to *produce products according to plan while simultaneously improving the organization's capability to produce better products.***

***The basic principles are those of statistical process control.* These techniques have been used successfully applied in many fields.**

**A process is said to be stable or under statistical control if its *future performance is predictable within established statistical limits.***

***If a process is not under statistical control, sustained progress is not possible until it is.***

**Many of the required techniques were first applied by Dr. W. E. Deming in Japan after World War II. While there are important differences, these concepts are just as important to the production of software as they are to the production of consumer goods like televisions and cameras.**

**Deming, W. E., *Quality, Productivity, and Competitive Position*, Cambridge, MA: Massachusetts Institute of Technology Center for Advanced Engineering Study, 1982.**

**The basic principle behind statistical control is measurement. More than a century ago Lord Kelvin stated: *"When you can measure what you are speaking about, and express it in numbers, you know something about it*; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science."**

**Dunham, J. R., and E. Kruesi, "The measurement task area," IEEE Computer, Vol. 16, no. 11, November 1983.**

**FAILURE RATES**

**Fifteen percent of all software projects never deliver anything; that is they fail utterly to achieve their established goals.**

**Overruns of one hundred to two hundred percent are common in software projects.**

**The manager of an effort that fails to deliver any result at all, or that overruns budget and schedule by a hundred percent or more, appears in retrospect *to never to have been in control*. But, almost certainly, when those projects were at the stage that your project is at now, *the managers believed they were in control*. What surprises did the future hold for them, and might it have similar surprises in store for you? *When you find yourself walking a battle field that is littered with corpses, you have to wonder, "What did they learn at the end that I still don't know?"***

**Failed projects frequently cannot be blamed on:**

**lack of strong motivation from project staff members,**

**lack of clear understanding of the issues,**

**adequate grasp of relevant technologies,**

**evident capability in the political sphere,**

**and yet they failed.**

**Most Common Reasons for failure:**

***Inflated and unreasonable expectations.***

***Success or staying in control means making sure that results match up to expectations. That requires two things:***

**1) You have to manage the project so that performance stays at or above some reasonable standard.**

**2) You have to make sure that original expectations are not allowed to exceed what's possible for a project performing at that standard.**

**This leads to MEASUREMENT. *You cannot control what you cannot measure.***

**The extent of control is a function of the precision of measurement.**

**Anything you don't measure at all is out of control.**

**The only unforgivable failure is the failure to learn from past failure.**

**Paradox**

**The worth of a project manager is largely a function of how well that manager stays in control. This leads to the following paradox:**

**“If Manager A plans mediocrity from the beginning to end of his/her project and delivers exactly that, while Manager B predicts great wonders and delivers wonders of a slightly lesser order, then A is a better manager than B.**

**I am not deliriously happy with about this conclusion, but it seems correct. Smooth operation of any but the smallest enterprise depends more on control than on the occasional wonder.”**

**Tom DeMarco, "Controlling Software Projects Management Measurement & Estimation," Yourdon Press Computing Series, 1982, Prentice-Hall, Inc., ISBN 0-13-171711-1.**

**Software Process Improvement**

**The first step an organization must perform to improve the quality of the software it produces is to *treat the entire software development process as a task that can be controlled, measured, and improved.***

**For this purpose *we define a process as that set of tasks that, when properly performed, produces the desire result.* Clearly, a fully effective software process must consider the relationships of all the required tasks, the tools and methods used, and the skill, training, and motivation of the people involved.**

**To improve their software capabilities, organizations must take six steps:**

**1) Understand the current status of their development process or processes.**

**2) Develop a vision of the desired process.**

**3) Establish a list of required process improvement actions in order of priority.**

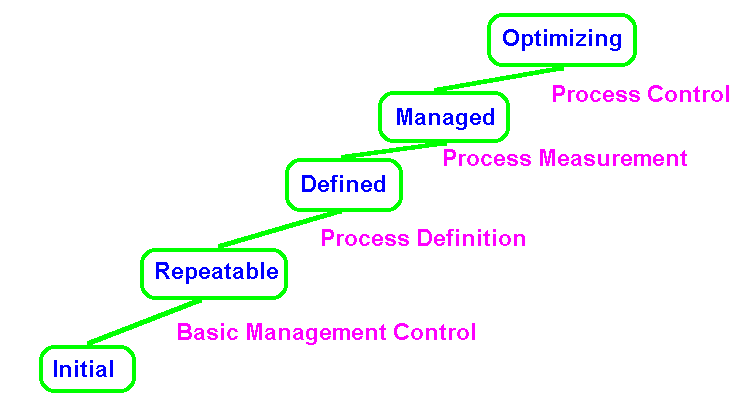
**4) Produce a plan to accomplish the required actions.**

**5) Commit the resources to execute the plan.**

**6) Start over at step 1.**

**For an organization to improve it is helpful to have a clear picture of the ultimate goal and some way to gauge progress along the way.**

**Watts S. Humphrey, "Managing the Software Process," Addison Wesely, ISBN 0-201-18095-2, 1990, p. 4.**

****

**PROCESS MATURITY LEVELS**

**1) Initial: Until the process is under statistical control, orderly progress in process improvement is not possible. While there are many degrees of statistical control, the first step is to achieve rudimentary predictability of schedules and costs.**

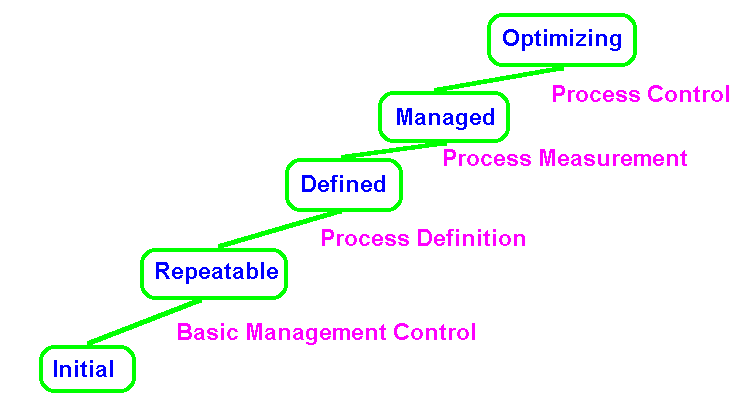
**2) Repeatable: The organization has achieved a stable process with a repeatable level of statistical control by initiating rigorous project management of commitments, costs, schedules, and changes.**

**3) Defined: The organization has defined the process as a basis for consistent implementation and better understanding. At this point advanced technology can usefully be introduced.**

**4) Managed: The organization has initiated comprehensive process measurements and analysis. This is when the most significant quality improvements begin.**

**5) Optimizing: The organization now has a foundation for continuing improvement and optimization of the process.**

**Watts S. Humphrey, "Managing the Software Process," Addison Wesely, ISBN 0-201-18095-2, 1990, pp 4 - 7.**

****

**Humphrey selected these levels because in his opinion they:**

**Reasonably represent the actual historical phases of evolutionary improvement of real software organizations,**

**Represent a measure of improvement that is reasonable to achieve from the prior level,**

**Suggest interim improvement goals and progress measures,**

**Make obvious a set of immediate improvement priorities, once an organization's status in this framework is known.**

**This process maturity structure is intended for use with an assessment methodology and a management system. Assessment helps an organization establish its maturity status. A management system establishes a structure for implementing priority improvement actions. Once an organizations maturity structure is defined, the organization can concentrate on those items that will help it advance to the next level.**

**Humphry, W. S. "The IBM large-systems software development process: objectives and direction. " *IBM Systems Journal*, vol. 24, no. 2, 1985.**

**Humphry, W. S. "Managing for Innovation-Leading Technical People. Englewood Cliffs, NJ: Prentice-Hall, 1987.**

**Humphrey, W. S., and D. H. Kitson. "Preliminary Report on Conducting SEI-Assisted Assessments of Software Engineering Capability," SEI Technical Report SEI-87-TR-16, July, 1987.**

**Radice, R. S., J. T. Harding, P. E. Munnis, and R. W. Phillips. "A programming process study," IBM Systems Journal, vol. 24, no. 2, 1985.**

**The Initial Process (level 1)**

**The initial level could be described as *ad hoc, chaotic, or one in which anarchy prevails*. The success of a project does not depend on the process, there is none. Management makes no significant contribution to success either. *Success depends entirely on the skill of the individuals on the project*.**

**Developers consider themselves to be artist not subject to rules or procedures. There may be standards or officially endorsed methodologies but they are *routinely ignored* or practiced according to the whims of the developers.**

**At this stage the organization typically *operates without formalized procedures, cost estimates, and project plans. Change control is lax or non-existent, there is little or no senior management exposure or understanding of the problems and issues involved.* Software installation and maintenance often present serious problems as *many problems have been deferred or even forgotten*.**

***There is no management mechanism to ensure that procedures for tracking and planning are used*. The best test to recognize a Level 1 organization is to observe them in a crisis. If the organization abandons established procedures and essentially reverts to coding and testing, then they are probably Level 1.**

**Level 1 organizations subscribe to the Mongolian Horde approach, if the current group of 1000 developers is insufficient, simply hire another 1000 and the problem will be solved.**

**Software coding and testing seems like progress but frequently reflects a Level 1 organization spinning its wheels.**

****

***The sooner you start coding before the design is complete the longer it will take to finish and the more problem will exist in the final product.***

**Level 1 organizations can only improve their performance by instituting basic project controls. *The most important are project management, management oversight, quality assurance, and change control*.**

**PROJECT MANAGEMENT: The fundamental role for project management is to ensure effective control of commitments, the organization does not accept work it cannot accomplish or complete. A plan is developed to determine the magnitude of the project, the resources required, and the best schedule for completion.**

**MANAGEMENT OVERSIGHT: Senior management must review and approve all major development plans prior to their official commitment. A quarterly audit should be made to determine facility-wide process compliance, schedule tracking, cost trends, computing services, and quality and productivity goals by project. Organizations that do not commit themselves to the review process typically suffer from inadequate implementation of the process as well as unpleasant and frequent over commitments of resources and cost surprises.**

**QUALITY ASSURANCE: A quality assurance organization is created to assure management that work is accomplished in the desired manner. To be effective, the quality assurance team must have an independent reporting line to senior management and sufficient resources to monitor all key planning, implementation, and verification activities. These functions are not free, they require 3 to 6 percent of the size of the organization personnel and budget.**

**CHANGE CONTROL: Changes in requirements, design, and code must be documented and controlled to allow reasonable stability throughout the development cycle. Though change is often required, historical evidence demonstrates that many changes can be deferred and incorporated at a later time. Changes must be controlled in an orderly fashion in order for design, implementation, and testing to be conducted in an effective manner.**

**Problems for companies wishing to go from Level 1 to Level 2 (based on surveys from several hundred US companies):**

**66% Estimating software size. If you do not know the problem size, the tools and methodologies you utilize do not matter.**

**64% Tracking software size changes. All changes must lead to adjustment of schedules, budgets, and personnel assignments prior to their acceptance.**

**58% Tracking code and test errors. Sophisticated organizations track errors from design and analysis, not just during coding and testing.**

**49% Scheduling and estimating. Mostly done seat-of-the-pants. Packaged software to produce Gantt and Pert charts is seldom used. They have never mastered critical path, topological sorting, and related topics.**

**45% Software commitment review. The organization takes on projects that exceed their resources or abilities and agree to unreasonable schedules and budgets under political pressure.**

***Emphasis on the "down stream" activities of coding and debugging with no consideration or only "lip service" for "up stream" activities such as specification and design.***

**Humphrey, Watts. "Software Process Maturity," Proceeding of the CASE WORLD Conference. Andover, MA: Digital Consulting, October 1990.**

**Yourdon, Edward, "Decline & Fall of the American Programmer," Prentice Hall, 1992, ISBN 0-13-203670-3.**

**The Repeatable Process (Level 2)**

**The *"Repeatable Process"* provides control over the way the organization establishes its plans and commitments. Because they tend to meet their commitments they tend to believe they have mastered the software problem.**

**If the Level 1 organization can be characterized as *"anarchy"*, the Level 2 organization is characterized by *"tribal folklore."* *There is a feeling that "we should all be doing things the same way."***

**The Level 2 organizations success has come from classical project management, not through advanced software engineering methodologies or CASE tools.**

**What the Level 2 organization has normally accomplished:**

**1) SOFTWARE COMMITMENT MANAGEMENT: A rational, dispassionate "process" has been established that allows for evaluating task and the organization to simply say "NO" when asked to accept a commitment that cannot be reasonably accomplished.**

**2) SOFTWARE PLANNING AND COST ESTIMATION: Formal methods for planning and estimation have been instituted.**

**3) CONFIGURATION MANAGEMENT AND CHANGE CONTROL: Modifications to the system during development as well as maintenance are tracked and made in an orderly fashion.**

**4) ESTABLISHMENT OF A SOFTWARE QUALITY ASSURANCE ORGANIZATION: This group focuses on the *quality* of the delivered system as well as adherence to budgets and schedules.**

**According to Software Engineering Institute (SEI) surveys, the most common weaknesses are:**

***88% Software Engineering Training:* The great majority of Level 2 groups require training for their professional staffs in formal methods. Ironically some of this training may have been done years earlier but must be repeated as the techniques were never utilized.**

***77% Regression Testing:* A formal regression testing process is required to ensure that modifications to a system perform as intended without damaging previously implemented functionality. This must be done in addition to configuration management.**

***50% Design Error Data:* Level 2 teams collect *down-stream* data about software defects introduced during coding and testing. They need to begin capturing information about defects introduced during the *up-stream* activities of requirements analysis, and design. These defects are typically more serious and more costly to repair if not discovered very early.**

***31% Software Process Group:* It is important to have a group formally charged with developing and disseminating information on the software engineering process. Some organizations may have already assigned this task to a training or standards department.**

***Major risks at this level include:***

**1) *New tools and methods will affect the process* and must be introduced with great care so that they do not destroy the intuitive historical base on which the organization depends.**

**2) The organization must break new ground (enter new territory) every time it is asked to develop a new product. For example, it is difficult to make the transition from banking applications on a single machine to a telecommunications environment without having problems with design, scheduling, and estimating. These changes destroy the intuitive basis for the organization's process.**

**3) *Major organizational changes are frequently disruptive.* For example, success in a Level 2 organization typically depends on the skill of the project manager. A level two company can survive the loss on an individual team member but not the manager. A new manager will not understand the process, there is no orderly basis for understanding the teams operation. New team members and managers must learn the ropes by word of mouth.**

**To advance to Level 3, the DEFINED PROCESS, the key actions are to *establish a process group, establish a development process architecture, and introduce a family of software engineering methods and technologies.***

**1) ESTABLISH A PROCESS GROUP: A technical group that focuses exclusively on improving the software process. Normally the staff is focused exclusively on developing the product. The process will not be improved unless someone is assign to its development independent of the product. The size of the group should be about 1 to 3 percent of the size of the organization but not less than four professionals.**

**2) ESTABLISH A SOFTWARE DEVELOPMENT PROCESS ARCHITECTURE: It must describe the technical and management activities required for proper execution of the development process. The architecture is a structural decomposition of the development cycle into tasks, each of which has a set of prerequisites, functional descriptions, verification procedures, and task completion specifications.**

**3) INTRODUCE A FAMILY OF SOFTWARE ENGINEERING METHODS AND TECHNOLOGIES: These should include formal design methods; analysis, design, code, and test inspections; library control systems; and comprehensive testing methods. Prototyping and modern implementation languages should be considered.**

**Humphrey, Watts. "Software Process Maturity," Proceeding of the CASE WORLD Conference. Andover, MA: Digital Consulting, October 1990.**

**Yourdon, Edward, "Decline & Fall of the American Programmer," Prentice Hall, 1992, ISBN 0-13-203670-3.**

**Level 3 - The Defined Process**

**The organization has *defined the process* and it forms a foundation for continuing progress. When faced with a crisis, the team will likely continue to use the process in favor of coding and testing.**

**The *process is written* and everyone in the organization can point to *"the bible"* and say *"This is the way we do things here!"***

**At the end of each project managers and software engineers look for opportunities to change the process to institute improvement. A key Software Engineering Institute premise is that the organization cannot make improvement unless the process is rigorously followed by everyone.**

***Level 3 organizations have usually accomplished the following:***

**INTRODUCTION OF FORMAL STANDARDS - versus the informal standards of level 2.**

**INSPECTIONS - as a formal quality assurance mechanism to insure the process is being followed.**

**MORE FORMAL TESTING POLICIES.**

**MORE ADVANCED SOFTWARE CONFIGURATION MANAGEMENT (SCM): - Configuration management is applied to upstream as well as downstream activities. Automated tools have been installed to aid the formal SCM, including configuration accounting and audits.**

**ESTABLISHMENT OF A SOFTWARE ENGINEERING PROCESS GROUP - to document the evolution and dissemination of the process throughout the organization.**

***With a "Defined Process" we can focus measurements on specific tasks.***

***To progress to Level 4, the "Managed Process" the key steps are:***

**1) *Identify and collect a minimum set of process measurements* to establish the cost parameters of each process step. Cost versus benefit must be established for each process step, e.g., the cost and yield of error detection and correction methods.**

***2) Establish a process database with the necessary resources to maintain the data.* Cost and yield data should be centrally maintained to prevent loss, insure availability, and to facilitate process quality and productivity analysis.**

***3) Skilled professional should analyze new data prior to its entry in the database and provide guidance for analysis methods and interpretation of the data.***

***4) An independent quality assurance group should assess the quality actions of each project and track its progress against its quality plan.* The relative quality of the product should be assessed and reported to management. Progress should be compared to historical data on similar projects in the database.**

***LEVEL 4 - THE MANAGED LEVEL***

**Humphreys characterizes the level 4 organization as "The organization has initiated comprehensive process measurements, beyond those of cost and schedule performance. This is when the most significant quality improvements begin."**

**The Level 4 organization is *not only measuring the product but the process as well.***

**Process data should not be used to compare projects or individuals. The variations in task complexity and cost of modifications make direct comparisons almost impossible. Its purpose should be restricted to illuminating our understanding of the product being developed and to provide an informed basis for improving the process.**

**The two fundamental requirements for advancing to the next level, the Optimized Process are:**

***1) Automated support for gathering process data.* All data is subject to error and omission. Every effort should be made to prevent data gathering from influencing the data collected (as in quantum mechanics, Heisenberg's Uncertainty Principle).**

***2) Use process data both to analyze and to modify the process to prevent problems and improve efficiency.***

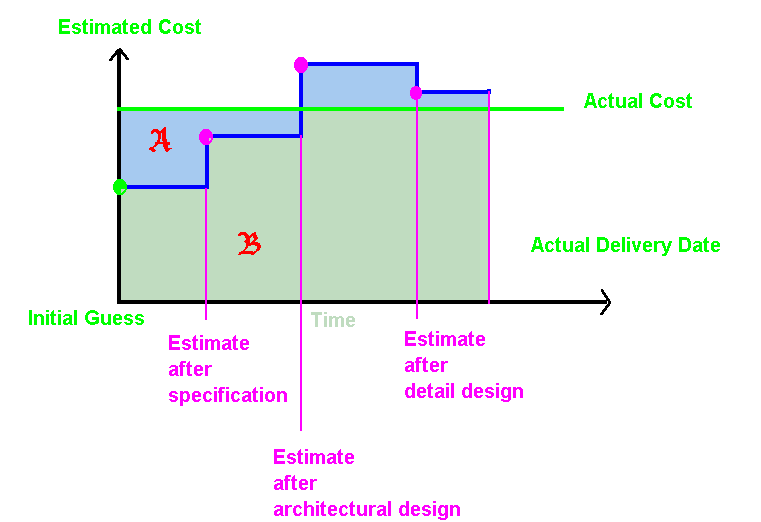
**If you intend to improve software quality, then your organization must live by four basic quality principles:**

**1) Nothing will change unless you establish aggressive quality goals.**

**2) *The goals must be numeric and measured* or the program will remain just talk.**

**3) Without quality plans, only you are committed to quality.**

**4) Quality plans are just talk unless you track and review the results.**

****

***Measuring the measurer:***

**The performance of the group charged with giving accurate predictions of when projects will be completed may be measured as follows:**

***Actual Value of Estimated Quantity x Duration of Unknown***

**Simple EQF = --------------------------------------------------------------**

***Integral( Estimated Value (t) - Actual Value) ) dt***

**or more simply, the Estimation Quality Factor is defined as the reciprocal of the average deviation between estimate and actual cost, e.g., B / A.**

**As long as the EQF gets larger from one project to the next, the estimating group is actually improving as their estimate deviates less from the actual.**

***The actual formula used is not as important as the fact that it exists, is measured, and is consistently applied.***

DeMarco, Tom. "Controlling Software Projects Management Measurement & Estimation," Prentice Hall, 1982, ISBN 0-13-171711-1.

**LEVEL 5 - THE OPTIMIZING LEVEL**

**In level 5 there is a paradigm shift. Up to this point management has focused primarily on the product and has mostly gathered and analyzed data that directly relates to improving the product. Now the data is used to tune and optimize the process itself. With practice, management should be able to produce major quality and productivity improvements by optimizing the process.**

**For example, the role of functional and system testing should be extended to one of gathering quality data on the programs. Studying individual bugs will allow a determination as to whether it is an isolated problem or indicates more profound design problems.**

**The formal emphasis is on continuous, ongoing process improvement based on the metrics captured in step four.**

**The major danger encountered in Level 4 is the cost of gathering the data.**

**YOU CAN NOT SKIP LEVELS**

**Transitions from Level 1 to Level 2 or from Level 2 to Level 3 *take from one to three years*, even with a dedicated management commitment to process improvement.**

***Much of the transition from one level to the next is Cultural in nature.* It cannot be rushed. Large software organizations have a lot of natural inertia that must be modified (social imperative).**

**To date, I have observed only a few software teams at level 4 or 5 and no complete organizations (Humphrey - circa 1990).**

**As of late 1991 the SEI has conducted *59 in-depth on-site assessments of 27 sites and 296 projects, together with some 167 workshop assessments.* Most of these have been in the military or aerospace industry including Gunther AFB, U.S. Space Command, Magnavox, Hughes Aircraft, GTE, McDonnell Douglas, Northrop, Medtronics, TRW, and the Strategic Air Command.**

**Approximately 81 percent of U.S. sites are at Level 1; approximately 12% at Level 2; about 7% at Level 3; and no sites surveyed were at Level 4 or 5. At the project level, 88% were at Level 1, 5% at level 2, 5% at Level 3, and *2% at Level 5*.**

**Advanced technology frequently cannot be successfully introduced until Level 3.**

**From Watts S. Humphrey, "Managing the Software Process," Addison Wesely, ISBN 0-201-18095-2, 1990, p. 14. This information was produced to give guidance to the military in selecting capable software contractors at the Software Engineering Institute, Carnegie Mellon University.**

**The original paper was Humphrey, W. S., and W. L. Sweet, "A method for assessing the software engineering capability of contractors," SEI Technical Report SEI-87-TR-23, September 1987.**

**Bannert, John. "New SEI Maturity Model Targets Key Pracitces," *IEEE Software*, November 1991.**

|  |
| --- |
| **SEI Maturity Level** |
| **Characteristics** |
| **Actions required to advance to next level.** |

|  |
| --- |
| **Initial** |
| **Ad hoc**  **Little formalization.**  **Tools informally applied to process.** |
| **Initiate rigorous project management oversight, and quality control.** |

|  |
| --- |
| **Repeatable** |
| **Achieved a stable process with a repeatable level of statistical control.** |
| **Establish a process group.**  **Establish a software development process architecture.**  **Introduce software engineering methods and technologies.** |

|  |
| --- |
| **Defined** |
| **Achieved a foundation for major and continuing progress** |
| **Establish a basic set of process metrics to identify the quality and cost parameters.**  **Establish a process database.**  **Gather and maintain process data.**  **Assess relative the relative quality of each product and inform management.** |

|  |
| --- |
| **Managed** |
| **Substantial quality improvements.**  **Comprehensive process measurement.** |
| **Support automated gathering of process data.**  **Use data to analyze and modify the process.** |

|  |
| --- |
| **Optimized** |
| **Major quality and quantity improvements gained through process modification.** |
| **Continue improvement and optimization of the process.** |

***ISO 9000***

***ANSI/ASQC Q90 to Q94***

**ISO 9000: Road map for implementing the rest of the series.**

**ISO 9001: Certify quality systems in the development cycle.**

**ISO 9002: For companies whose focus is production and installation.**

**ISO 9003: For companies such as commodity suppliers in which comprehensive quality systems may not be important or necessary.**

**ISO 9004: Standards 9000 through 9003 are contractual documents written in *“shall”* language. ISO 9004 is recommended quality guidelines in *“should”* language.**

|  |  |  |
| --- | --- | --- |
|  | **System Requirements** |  |
| ***ISO 9003***  ***12 Requirements*** | **ISO 9002**  **18 Requirements** | **ISO 9001**  **20 Requirements** |
| **Management Responsibility** |  |  |
| **Quality System** |  |  |
| **Product Identification and Tractability** |  |  |
| **Inspection Status** |  |  |
| **Inspection and Testing** |  |  |
| **Inspection, Measuring, and Test Equipment** |  |  |
| **Control of Nonconforming Product** |  |  |
| **Handling, Storage, Packaging and Delivery** |  |  |
| **Document Control** |  |  |
| **Quality Records** |  |  |
| **Training** |  |  |
| **Statistical Techniques** |  |  |
|  | **Internal Auditing** |  |
|  | **Contract Review** |  |
|  | **Purchasing** |  |
|  | **Process Control** |  |
|  | **Purchaser Supplied Product** |  |
|  | **Corrective Action** |  |
|  |  | **Design Control** |
|  |  | **Servicing** |

**Conformity Assessment**

**1) Self-certification: Also called “manufacturer’s declaration.”**

**2) Product assessment: A sample and/or representative products from a production run are tested and inspected to determine conformance to customers’ requirements. Engineering drawings and calculations may also be checked.**

**3) Quality system registration: The supplier is audited by an independent third party called a “registrar.” The registrar provides two basic services: they audit against one of three ISO standards (ISO 9001, 9002, or 9003), and they place the company on a list of suppliers that have been certified by their auditors.**

***Registrars* are approved and certified by *accreditors.* In the U.S., the *Registrar Accreditation Board (RAB).***

**Cost**

1. **assume a single sight.**
2. **a single produce line.**
3. **two to three hundred employees**
4. **registration to ISO 9002**
5. **first attempt is successful**
6. **no corrective action or post-assessments**

**The direct cost averages (1990 informal survey of RAB) $10,000 to $15,000 plus expenses such as motels and travel cost. Dual registration domestic and European adds about $4,000. This assumes no consulting fees are required and does not cover internal costs.**

**Similarities and Differences Between the Malcolm Baldridge National Quality Award and ISO 9000**

|  |  |
| --- | --- |
| **U.S. based** | **Internationally applicable** |
| **Highest level of quality** | **Highest common-denominator quality criteria** |
| **“World-class” quality** | **Doable and attainable quality** |
| **Advanced TQM award** | **First step in TQM journey** |
| **System-oriented** | **System -oriented** |
| **Broad quality criteria covered** | **ISO 9001 generically covers MBNQA criteria** |
| **Focus on control, participation, and improvement** | **Focus on control** |
| **Exclusive, only two winners per category** | **Inclusive, all can become registered** |
| **Quality criteria higher and more demanding, stress customer satisfaction, quantifiable results, and continuous improvement** | **Quality criteria generic; customer satisfaction improvement not emphasized** |

**“ISO 9000 A Comprehensive Guide to Registration, Audit Guidelines and Successful Certification” by Hutchins, ISBN 0-939246-31-7.**

**“ISO 9000 International Standards for Quality Management,” 4th Edition, from ANSI, FAX 212-302-1286, ph 212-642-4900, approximate cost $240.00.**

**J. D. Powers Quality Award**

**Usage of Popular Methodologies**

**"The Second Annual Report on CASE," Bellevue, WA: Case Research Corporation, 1990, Gregory H. Boone, Vaughan P. Merlyn, and Roger E. Dobratz.**

**Note that OOP's is in the noise level.**

**The most significant item however is the predominance of "in-house" methodologies: most companies pick the best features of several different textbook methodologies and then adapt them to their own needs.**

**Learning Time (% of respondents versus months)**

**The typical Software engineer takes 1 to 2 years to become familiar and comfortable with a methodology. Methodologies are more complex to learn than programming languages.**

Edward Yourdon "Decline & Fall of the American Programmer," Prentice-Hall, 1992, pp. 105-106, ISBN 0-13-203670-3.

***Edward Yourdon's Silver Bullets***

***For Quality and Productivity***

**1) Better programming languages.**

**2) Better people.**

**3) Automated tools.**

**4) Joint Application Design (JAD).**

**5) Rapid Application Development (RAD).**

**6) Prototyping.**

**7) Agile**

**8) Patterns**

**9) Structured techniques.**

**10) Information engineering.**

**11) Object-oriented methodologies.**

**12) Software reusability.**

**13) Software reengineering.**

**14) Others**

**Yourdon, Edward, "Decline & Fall of the American Programmer," Yourdon Press, Prentice Hall, 1992, ISBN 0-13-203670-3, p25-37.**

**Brooks, Fred. "No Silver Bullets," *IEEE Computer*, April 1987.**

***1) Better Programming Languages:***

**a. The perennial favorite of the tech weenies and propeller heads in the organization.**

**b. Ignores the fact that 50% to 80% of MIS budget is spent in maintaining old systems, e.g., bug repair, enhancement, responding to environmental changes.**

**c. Ignores the problem of retraining existing programmers who have spent the last 5 to 10+ years in their old language.**

**d. "The real problem with programming languages as the silver bullet solution to software problems is that it puts emphasis at the wrong level: *better coding techniques may do nothing more than help you arrive at a disaster sooner than before*. Better programming languages used without anything else may be just what your programmers need to develop a *brilliant solution to the wrong problem*.**

**e. Certainly one major lesson we have learned from the past 20 years of software engineering experience is that *there is more to be gained from attention to design, analysis, and business strategy issues than from attention to programming-level issues.*"**

**f. CASE tools that generate all or at least most of the code will reduce or eliminate the importance of programming languages eventually.**

**2) Better People:**

**a. Good people have been consistently shown to be an *order of magnitude better* than the lowest performers in an organization and *twice as productive as median performers*. Even with poor tools and mediocre languages they turn out good software.**

**b. DeMarco and Lister point out in "Peopleware," Dorset House, 1987, that the *major problems of our work are not so much technological as sociological in nature*.**

**Most managers will even concede the idea that they have more people worries than technical worries, yet *they seldom manage that way.***

**Peopleware efforts should be directed at:**

**Hiring the best people.**

**Engaging in the ongoing training and education of existing staff (four weeks per year to prevent Technical Obsolescence).**

**Motivating people for higher levels of performance (positive).**

**Developing performance management ideas to align personal goals with corporate goals.**

**Offering an *adequate working environment, with particular emphasis on adequate facilities rather than the pigsties* in which many software engineers find themselves squatting for 8 or more hours per day.**

**Placing more emphasis on creating and maintaining effective teams of people who can work together to create high-quality software products.**

**3) Automated Tools:**

**a. CASE tools are expected to be mainstream technology by the mid to late 1990's with approximately 50% of software engineers using them.**

**b. This technology will separate the world-class players from the wish-washy mediocre shops.**

**One reason is cost, a reasonably equipped CASE environment in the United States normally costs from *$30,000 to $50,000 per person*.**

**This is an expensive item even if justified by 20% to 30% improvements in productivity. It will not help with existing maintenance and modification problems, frequently 80% or more of existing costs.**

**Short term problems:**

**CASE users frequently experience a *productivity decrease* for the first 3 to 6 months.**

**Extensive training is required for management as well as technical personnel.**

**Expect to use the tool for 12 to 18 months before productivity gains are visible.**

**Automated software environments (CASE) are likely to be the largest capital investment in the software business.**

**Few shops are immediately ready for the technology or will pick the right tool the first time.**

**4) Joint Application Development (JAD):**

**JAD was originally developed by IBM Canada in the 1970's as a means for *bringing users and systems analysis's together for intensive, highly productive mediated sessions to elicit the requirements* for a new system. CASE tools have revived the concept.**

**5) Rapid Application Development (RAD)**

**A modern variation of JAD. Usually described as a *combination of JAD sessions to determine user requirements quickly, as well as CASE tools, Prototyping techniques, Rambo-style SWAT teams, and a formal software development methodology* to implement those requirements quickly.**

**As a combination of tools and techniques, RAD has a lot of potential. The individual techniques used alone should not be expected to perform adequately on a regular basis.**

**Some people just see this as prototyping/agile development. It is *particularly useful when the user is unsure of his or her real requirements, the user interface, algorithms required, or the desired inputs and outputs*.**

**It is expected that the use of CASE tools in conjunction with JAD will result in the evolution of *"groupware"* tools to facilitate cooperative work activities in teams.**

**7) Structured Techniques:**

**By the mid 1990's, structured techniques became the *most widely practiced form of systems analysis in North America and one of the two or three most widely popular techniques in the world* (Yourdon, Edward. "Japan Revisited," *American Programmer*, February 1990).**

**What formal methodology dominates the market today?**

**Major improvement for organizations with no existing methodology.**

**Most widely supported methodology by the CASE products. Unfortunately *many CASE tools only support older dialects* of structured techniques and only support the modeling of functions. *The tools have not kept up!***

***Data modeling* was not incorporated until the mid 1980's. *Entity-relationship diagrams* for modeling data and *state-transition diagrams* for real time systems have also been added.**

**McMenamin and Palmer added the crucial concept of *events* and *event partitioning* as well as a *critical emphasis on modeling the "essence" of a system*.**

**Another concept to be considered is Martin's *User-Centered Requirements Analysis* which shift the perspective of systems analysis noticeably toward the end user.**

**Yourdon, Edward. Modern *Structured Analysis*. Englewood Cliffs, NJ: Yourdon Press / Prentice Hall, 1989.**

**Ward, Paul, and Stephen J. Mellor. *Structured Development of Real-Time Systems*. Englewood Cliffs, NJ: Yourdon Press / Prentice Hall, 1985.**

**Martin, Charles. *User-Centered Requirements Analysis*. Englewood Cliff, NJ: Prentice Hall, 1988.**

**Martin, James. *Information Engineering*, Vols. 1-3. Englewood Cliffs, NJ: Prentice Hall, 1990.**

**8) Information Engineering:**

**Introduced by James Martin and others at the beginning of the 1990's and used by approximately 10% as many DP organizations as using structured techniques. It was the fastest growing technique in the early 1990's.**

**Information engineering emphasizes "data" as a corporate asset and as the basis for system analysis and design.**

**Though it can be used for the analysis and design phases of individual projects, information engineering is most often perceived as a methodology for enterprise wide modeling activities. By contrast, structured analysis is generally perceived as a methodology for modeling individual systems within an organization.**

**9) Object Oriented:**

**a. Most widely discussed technology for the 1990's and through 2010.**

**b. Most papers address which language is better, object oriented programming, and low level design issues.**

**Word-class organizations looking to object-oriented technology to improve their productivity and software quality dramatically are strongly advised to focus their attention on object oriented analysis, object-oriented design, and object-oriented databases, not object-oriented implementation.**

**10) Software Reusability:**

**a. World-class organizations report reuse of *60-70%*.**

**b. Average organizations report reuse of *20-30%*.**

**c. *The key issues for success in reusability initiatives are managerial and cultural in nature.* Software engineers must see a motivational incentive for reusing components from a library.**

**If management assesses the level of reuse only at the end of the project (passive) rather than building it into the project plan from the beginning (active), it's likely the extent of reuse will be random and accidental.**

**11) Software Reengineering:**

**Most organizations spend the majority of their software dollar in fixing, upgrading, or enhancing existing systems. In fact this often exceeds 80% of the software budget. Hence the greatest productivity gains are potentially in this area.**

***Unfortunately the ability to perform these tasks is frequently determined by the quality of the original implementation.***

**Restructuring is the transformation of old unstructured code to functionally equivalent structured code. Only about 10% of organizations that could benefit utilize this technique and existing software to do it. While helpful, *the benefits of these techniques appear limited in practice.***

**Reengineering tools provide a means for maintenance programmers to understand more easily "alien" code written years earlier by people frequently no longer with the organization.**

**Reverse engineering attempts to reconstruct design models and specification-level models directly from the source code.**

**Re-Use Failure and Performance Management**

**To promote reuse, management generally *issues memos, has training sessions, holds revival meetings for people to take their vows of support, and has the revered standards group publish the ideal characteristics of a reusable module*. At the end of a year or two the library is still empty, why?**

**Reuse as seen by management is almost never the same as reuse as seen by the software engineer in terms of personnel consequences:**

**It took three weeks of my time to make this component ready for submission to the library.**

**The group in charge of the library rejected the submission. They didn't feel it would receive the required level of use.**

**After two weeks, I discovered that this really was not a candidate for reuse, do I still get my reward?**

**The software engineer's manager appreciates the effort, thanks him/her publicly, and remembers to include the effort in the next performance review.**

**While building for reuse, other more enjoyable tasks such as coding more modules must be postponed.**

**The software engineer encounters problems with several modules being late due to time spent in generalizing one or more modules for the library, does anyone understand?**

**The time to perform the generalization requires that the software engineer put in unpaid overtime to stay on schedule.**

**That which is of consequence to the individual is of no consequence to the organization at all and vice versa.**

**Consider one company developing a library as part of a project for the military. Is it to our advantage to provide library access to competitors (Lowery AFB - Ada)?**

**Consequences can be characterized as:**

**Positive (P) or negative (N)**

**Immediate (I) or future (F)**

**Certain (C) or uncertain (U).**

**People tend to respond to situations that are *positive, immediate, and certain* - PICs.**

**They avoid situations that are *negative, immediate, and certain* - NICs.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Consequences** | **P/N** | **I/F** | **C/U** |
| **Creating a reusable module.** | **P** | **I** | **U** |
| **Unable to create a reusable module.** | **N** | **I** | **U** |
| **Rejection of the module.** | **N** | **F** | **U** |
| **Gaining manager recognition.** | **P** | **F** | **U** |
| **Unable to accomplish more enjoyable task.** | **N** | **I** | **C** |
| **Late delivery due to lost time.** | **N** | **F** | **U** |
| **Working overtime.** | **N** | **I** | **C** |

***In this analysis there are two NICs and "no" PICs.* There is very little motivation on a personal level to produce the reusable modules.**

**To effectively populate the reuse library, *management must provide immediate job satisfaction and minimization of penalties to the personnel in the trenches.***

**They must not feel that they have *fallen behind or spent their budget to give others an advantage over them.***

**Edward Yourdon "Decline & Fall of the American Programmer," Prentice-Hall, pp. 64 - 67, ISBN 0-13-203670-3.**