

SOFTWARE ENGINEERING: THE STATE OF THE ART IN 2005

Version 5 – February 11, 2005

Abstract

The phrase “state of the art” refers to software development practices that have consistently yielded results that are in the top 15% of approximately 12,000 projects examined. There are hundreds of ways to develop software that lead to overruns and failures. Surprisingly, there are only a few ways to develop software that consistently result in successful outcomes.

The three weakest links in the chain of software development practices are those dealing with planning and estimating before the project starts, with absorbing changing requirements during the project, and with successfully minimizing bugs or defects via excellent quality control. Successful projects using state of the art methods excel in all three of these critical activities: estimating, change control, and quality control. By contrast, projects that run late or fail usually had optimistic estimates, did not anticipate changes, and failed to control quality.

Capers Jones, Founder and Chief Scientist
Software Productivity Research LLC

Email CJones@spr.com
Web <http://www.spr.com>

Copyright © 2001-2005 by Capers Jones.
All Rights Reserved.

SOFTWARE DEVELOPMENT: THE STATE OF THE ART IN 2005

The author's company performs software assessments and also collects benchmark data on productivity and quality. Approximately 12,000 projects have been examined since 1984 as discussed in the author's book Software Assessments Benchmarks, and Best Practices, Addison Wesley Longman, 2000. This large quantity of data on software projects is useful for research studies on practices that tend to cause projects to be either better or worse than average.

It is difficult to pick out successful or unsuccessful techniques from projects that are more or less average. However when projects in the top 15% of productivity rates are compared to projects in the bottom 15% of productivity, some very interesting differences stand out.

Software projects are influenced by more than 100 different factors. (Our software assessment method includes more than 100 questions on processes, tools, and other pertinent factors.) However when similar projects are examined where one is successful (i.e. on time with high productivity and good quality) and one is a failure (i.e. cancelled, delayed, or inoperable) about a dozen key factors tend to distinguish success from failure.

Following are major factors associated with both success and failure first published by the author in his book Patterns of Software System Failure and Success (International Thomson Computer Press 1995). The factors that the author has observed as the most significant include:

Successful Projects

Effective project planning
Effective project cost estimating
Effective project measurements
Effective project milestone tracking
Effective project quality control
Effective project change management
Effective development processes
Effective communications
Capable project managers
Capable technical personnel
Significant use of specialists
Substantial volume of reusable materials

Failing Projects

Inadequate project planning
Inadequate cost estimating
Inadequate measurements
Inadequate milestone tracking
Inadequate quality control
Ineffective change control
Ineffective development processes
Ineffective communications
Ineffective project managers
Inexperienced technical personnel
Generalists rather than specialists
Little or no reuse of technical material

In addition to the factors that the author and his colleagues at Software Productivity Research have studied, James Johnson and his colleagues at the Standish Group have also studied failing projects. Some of the Standish findings published in their famous "Chaos Report" overlap our findings (Chaos Report, 2000). However, the three top factors discussed in the Chaos report developed by James Johnson of the Standish group include:

Successful Projects

Effective user involvement
Executive management support
Clear requirements

Failing Projects

Insufficient user involvement
Lack of management support
Ambiguous, changing requirements

The 12 factors we have noted are associated with the development team that builds the software. The three top factors that the Standish report emphasizes focus on the clients of the software. This is not too surprising since the Standish report emphasizes the views of project managers rather than client executives. The Standish studies and the SPR studies are complimentary, because both sets of factors are important.

Both clients and project managers can have either positive or negative impacts. However, the author has never seen a failing project or one whose delivery was missed by more than about 30% that did not have serious flaws in project management methods. Project management errors tend to be associated with 100% of the cancelled and seriously delayed software projects noted over many years.

Good project management can prevent problems from occurring or correct them early before they become serious. Bad project management lacks the foresight to prevent problems and is not successful in correcting them. Indeed, bad project management causes many of the problems in the first place.

It is useful to consider all 15 of these factors to ascertain the most probable reasons for software failures, and to look at the opposite extreme or the current software state of the art circa 2005 that consistently leads to successful outcomes.

Because large software systems are far more hazardous than small, this study concentrates on the state of the art tools and methods used to develop large systems that are in the approximate range of 10,000 function points in size. (A size of 10,000 function points is roughly equal to about 1,250,000 statements in the C programming language.)

Project Planning

Project planning for large projects in large corporations often involves both planning specialists and automated planning tools. The state of the art for planning software projects circa 2005 for large projects in the nominal 10,000 function point range involves:

- Development of complete work breakdown structures
- Consideration to staff hiring and turnover during the project
- Usage of automated planning tools such as Artemis Views or Microsoft Project
- Factoring in time for requirements gathering and analysis
- Factoring in time for handling changing requirements
- Consideration given to multiple releases if requirements creep is extreme
- Factoring in time for a full suite of quality control activities

Successful projects do planning very well indeed. Delayed or cancelled projects, however, almost always have planning failures. The most common planning failures include 1) Not dealing effectively with changing requirements; 2) Not anticipating staff hiring and turnover during the project; 3) Not allotting time for detailed requirements analysis; 4) Not allotting sufficient time for inspections, testing, and defect repairs.

Project Estimating

The state of the art for estimating software projects in the nominal 10,000 function point range is a critical activity. The current state of the art for estimating large systems involves the use of:

- Formal sizing approaches for major deliverables based on function points
- Comparison of estimates to historical data from similar projects
- Trained estimating specialists
- Software estimating tools (CHECKPOINT, COCOMO, KNOWLEDGEPLAN, PRICE-S, SEER, SLIM, SOFTCOST etc.)
- Inclusion of new and changing requirements in the estimate
- Quality estimation as well as schedule and cost estimation

Failing projects often understate the size of the work to be accomplished. Failing projects often omit to perform quality estimates at all. Overestimating productivity rates is another common reason for cost and schedule overruns.

Because estimating is complex trained estimating specialists are the best, although such specialists are few in number. These specialists always utilize one or more of the leading commercial software estimating tools or sometimes proprietary estimating tools. About half of our leading clients utilize two commercial software estimating tools frequently, and may own as many as half a dozen. Manual estimates are never adequate for major systems in the 10,000 function point range.

Manual estimates using standard templates are difficult to modify when assumptions change. As a result, they often fall behind the reality of ongoing projects with substantial rates of change. My observations of the overall results of using manual estimates for projects larger than about 1000 function points in size is that they tend to be incomplete and err on the side of excessive optimism.

Project Measurements

Leading companies always have software measurement programs for capturing productivity and quality historical data. The state of the art of software measurements for projects in the nominal 10,000 function point domain includes:

- Measures of accumulated effort
- Measures of accumulated costs
- Measures of development productivity
- Measures of the volume of requirements changes
- Measures of defects by origin
- Measures of defect removal efficiency

The measures of effort are often granular and support work-breakdown structures. Cost measures are complete, and include development costs, contract costs, and costs associated with purchasing or leasing packages. There is one area of ambiguity even for top companies: the overhead or burden rates associated with software costs vary widely and can distort comparisons between companies, industries, and countries.

Development productivity circa 2005 normally uses function points in two fashions: function points per staff month and/or work hours per function point.

Measures of quality are powerful indicators of top-ranked software producers. Laggards almost never measure quality while top software companies always do. Quality measures include data on defect volumes by origin (i.e. requirements, design, code, bad fixes) and severity level.

Really sophisticated companies also measure defect removal efficiency. This requires accumulating all defects found during development and also after release to customers for a pre-determined time period. For example if a company finds 900 defects during development and the clients find 100 defects in the first three months of use, then the company achieved a 90% defect removal efficiency level. Top companies are usually better than 95% in defect removal efficiency, which is about 10% better than U.S. average of 85%.

Milestone Tracking

The phrase "milestone tracking" refers to having formal closure on the development of key deliverables. Normally the closure milestone is the direct result of some kind of review or inspection of the deliverable. A milestone is not an arbitrary calendar date.

Project management is responsible for establishing milestones, monitoring their completion, and reporting truthfully on whether the milestones were successfully completed or encountered problems. When serious problems are encountered, it is necessary to correct the problems before reporting that the milestone has been completed.

A typical set of project milestones for software applications in the nominal 10,000 function point size range would include:

- Completion of requirements review
- Completion of project plan review
- Completion of cost and quality estimate review
- Completion of external design reviews
- Completion of data base design reviews
- Completion of internal design reviews
- Completion of quality plan and test plan reviews
- Completion of documentation plan review
- Completion of deployment plan review
- Completion of training plan review
- Completion of code inspections
- Completion of each development test stage
- Completion of customer acceptance test

Failing or delayed projects usually lack of serious milestone tracking. Activities might be reported as finished while work was still on going. Milestones might be simple dates on a calendar rather than completion and review of actual deliverables. Some kinds of reviews may be so skimpy as to be ineffective. Other topics such as training may be omitted by accident.

It is important that the meaning of "milestone" is ambiguous in the software industry. Rather than milestones being the result of a formal review of a deliverable,

the term "milestone" often refers to arbitrary calendar dates or to the delivery of unreviewed and untested materials.

Delivering documents or code segments that are incomplete, contain errors, and cannot support downstream development work is not the way milestones are used by industry leaders.

Another aspect of milestone tracking among industry leaders is what happens when problems are reported or delays occur. The reaction is strong and immediate: corrective actions are planned, task forces assigned, and correction begins to occur. Among laggards, on the other hand, problem reports may be ignored and very seldom do corrective actions occur.

Successful and Unsuccessful Quality Control

Effective software quality control is the most important single factor that separates successful projects from delays and disasters. The reason for this is because finding and fixing bugs is the most expensive cost element for large systems, and takes more time than any other activity.

Successful quality control involves defect prevention, defect removal, and defect measurement activities. The phrase "*defect prevention*" includes all activities that minimize the probability of creating an error or defect in the first place. Examples of defect prevention activities include the Six-Sigma approach, joint application design (JAD) for gathering requirements, usage formal design methods, usage of structured coding techniques, and usage of libraries of proven reusable material.

The phrase "*defect removal*" includes all activities that can find errors or defects in any kind of deliverable. Examples of defect removal activities include requirements inspections, design inspections, document inspections, code inspections, and all kinds of testing.

The phrase "*defect measurement*" includes measures of defects found during development and also defects reported by customers after release. These two key measures allow leading companies to calculate their defect removal efficiency rates, or the percentages of defects found prior to release of software applications. Supplemental measures such as severity levels, code complexity, and defect repair rates are also useful and important. Statistical analysis of defect origins and root-cause analysis are also beneficial. Of course the costs of defect repairs are also key measurements.

Some activities benefit both defect prevention and defect removal simultaneously. For example participation in design and code inspections is very effective in terms of defect removal, and also benefits defect prevention. The reason why defect prevention is aided is because inspection participants learn to avoid the kinds or errors that inspections detect. Successful quality control activities for 10,000 function point projects include the following:

Defect Prevention

- Joint application design (JAD) for gathering requirements
- Formal design methods
- Structured coding methods

- Formal test plans
- Formal test case construction
- Six-Sigma approaches

Defect Removal

- Requirements inspections
- Design inspections
- Document inspections
- Code inspections
- Test plan and test case inspection
- Defect repair inspection
- Software quality assurance reviews
- Unit testing
- Component testing
- New function testing
- Regression testing
- Performance testing
- System testing
- Acceptance testing

The combination of defect prevention and defect removal activities leads to some very significant differences in the overall numbers of software defects compared between successful and unsuccessful projects. For projects in the 10,000 function point range, the successful ones accumulate development totals of around 4.0 defects per function point and remove about 95% of them before delivery to customers. In other words, the number of delivered defects is about 0.2 defects per function point or 2,000 total latent defects. Of these about 10% or 200 would be fairly serious defects. The rest would be minor or cosmetic defects.

By contrast, the unsuccessful projects accumulate development totals of around 7.0 defects per function point and remove only about 80% of them before delivery. The number of delivered defects is about 1.4 defects per function point or 14,000 total latent defects. Of these about 20% or 2,800 would be fairly serious defects. This large number of latent defects after delivery is very troubling for users.

One of the reasons why successful projects have such high defect removal efficiency compared to unsuccessful projects is the usage of design and code inspections (Radice 2002; Wieggers 2002). Formal design and code inspections average about 65% efficient in finding defects. They also improve testing efficiency by providing better source materials for constructing test cases.

Unsuccessful projects typically omit design and code inspections and depend purely on testing. The omission of up-front inspections causes three serious problems: 1) The large number of defects still present when testing begins slows down the project to a standstill; 2) The "bad fix" injection rate for projects without inspections is alarmingly high; 3) The overall defect removal efficiency associated with only testing is not sufficient to achieve defect removal rates higher than about 80%.

(Note: The term "bad fixes" refers to secondary defects accidentally injected by means of a patch or defect repair that is itself flawed. The industry average is about 7% but for unsuccessful projects, the number of bad fixes can approach 20%; i.e. 1

out of every 5 defect repairs introduced a fresh defects (Jones 1997). Successful projects, on the other hand, can have bad-fix injection rates of only 2% or less.)

Defect Measurements

Among the quality measurements noted among industry leaders, the following are the most common and also the most important:

Customer satisfaction: Leaders perform annual or semi-annual customer satisfaction surveys to find out what their clients think about their products. Leaders also have sophisticated defect reporting and customer support information available via the web. Many leaders in the commercial software world have active user groups and forums. These groups often produce independent surveys on quality and satisfaction topics. There are also Focus groups, and some large software companies even have formal "usability labs" where new versions of products are tried out by customers under controlled conditions.

Defect quantities and origins: Industry leaders keep accurate records of the bugs or defects found in all major deliverables, and they start early during requirements or design. At least five categories of defects are measured: requirements defects, design defects, code defects, documentation defects, and bad fixes or secondary bugs introduced accidentally while fixing another bug. Accurate defect reporting is one of the keys to improving quality. In fact analysis of defect data to search for root causes had led to some very innovative defect prevention and defect removal operations in many SPR client companies. Overall, careful measurement of defects and subsequent analysis of the data is one of the most cost-effective activities a company can perform.

Defect removal efficiency: Industry leaders know the average and maximum efficiency of every major kind of review, inspection, and test and they select optimum series of removal steps for projects of various kinds and sizes. The use of pre-test reviews and inspections is normal among Baldrige winners and organizations with ultra-high quality, since testing alone is not efficient enough. Leaders remove from 95% to more than 99% of all defects prior to delivery of software to customers. Laggards seldom exceed 80% in terms of defect removal efficiency, and may drop below 50%.

Delivered defects by application: Industry leaders begin to accumulate statistics on errors reported by users as soon as the software is delivered. Monthly reports are prepared and given to executives, which show the defect trends against all products. These reports are also summarized on an annual basis. Supplemental statistics such as defect reports by country, state, industry, client, etc. are also included.

Defect severity levels: All of the industry leaders, without exception, use some kind of a severity scale for evaluating in-coming bugs or defects reported from the field. The number of plateaus vary from one to five. In general, "Severity 1" are problems which cause the system to fail completely, and the severity scale then descends in seriousness.

Complexity of software: It has been known for many years that complex code is difficult to maintain and has higher than average defect rates. A variety of complexity analysis tools are commercially available that support standard

complexity measures such as cyclomatic and essential complexity. It is interesting that the systems software community is much more likely to measure complexity than the information technology (IT) community.

Test case coverage: Software testing may or may not cover every branch and pathway through applications. A variety of commercial tools are available that monitor the results of software testing, and help to identify portions of applications where testing is sparse or nonexistent. Here too the systems software domain is much more likely to measure test coverage than the information technology (IT) domain.

Cost of quality control and defect repairs: One significant aspect of quality measurement is to keep accurate records of the costs and resources associated with various forms of defect prevention and defect removal. For software, these measures include: 1) The costs of software assessments; 2) the costs of quality baseline studies; 3) the costs reviews, inspections, and testing; 4) the costs of warranty repairs and post-release maintenance; 5) the costs of quality tools; 6) the costs of quality education; 7) the costs of your software quality assurance organization; 8) the costs of user satisfaction surveys; 9) the costs of any litigation involving poor quality or customer losses attributed to poor quality. In general the principles of Crosby's "Cost of Quality" topic apply to software, but most companies extend the basic concept and track additional factors relevant to software projects.

Change Management

Applications in the nominal 10,000 function point size range run from 1% to 3% per month in new or changed requirements during the analysis and design phases. The total accumulated volume of changing requirements can top 50% of the initial requirements when function point totals at the requirements phase are compared to function point totals at deployment. Therefore successful software projects in the nominal 10,000 function point size range must use state of the art methods and tools to ensure that changes do not get out of control.

The state of the art of change control for applications in the 10,000 function point size range include the following:

- A joint client/development change control board or designated domain experts
- Use of joint application design (JAD) to minimize downstream changes
- Use of formal prototypes to minimize downstream changes
- Planned usage of iterative development to accommodate changes
- Formal review of all change requests
- Revised cost and schedule estimates for all changes > 10 function points
- Prioritization of change requests in terms of business impact
- Formal assignment of change requests to specific releases
- Use of automated change control tools with cross-reference capabilities

One of the observed byproducts of the usage of formal JAD sessions is a reduction in downstream requirements changes. Rather than having unplanned requirements surface at a rate of 1% to 3% per month, studies of JAD by IBM and other companies have indicated that unplanned requirements changes often drop below 1% per month due to the effectiveness of the JAD technique.

Prototypes are also helpful in reducing the rates of downstream requirements changes. Normally key screens, inputs, and outputs are prototyped so users have some "hands on" experience with what the completed application will look like.

However, changes will always occur for large systems. It is not possible to freeze the requirements of any real-world application and it is naïve to think this can occur. Therefore leading companies are ready and able to deal with changes, and do not let them become impediments to progress. Therefore some form of iterative development is a logical necessity.

Development Processes

The phrase "development process" refers to a standard set of activities that are performed in order to build a software application. For conventional software development projects, about 25 activities and perhaps 150 tasks are usually included in the work breakdown structure (WBS).

The work-breakdown structure of large systems will vary based on whether the application is to be developed from scratch, or involves modifying a package of a legacy application. In today's world circa 2002 projects that are modifications are actually more numerous than complete new development projects.

An effective development process for projects in the nominal 10,000 function point range that include acquisition and modification of commercial software packages would resemble the following:

- Requirements gathering
- Requirements analysis
- Requirements inspections
- Package analysis
- Requirements/Package mapping
- Contacting Package user association
- Package licensing and acquisition
- Training of development team in selected package
- Design of package modifications
- Development of package modifications
- Review and inspection of package modifications
- Documentation of package modifications
- Testing of package modifications
- Training of user personnel in package and modifications
- Deployment of package modifications

These high-level activities are usually decomposed into a full work breakdown structure with between 150 and more than 1000 tasks and lower level activities. Doing a full work breakdown structure is too difficult for manual approaches on large applications. Therefore project management tools such as Artemis Views, Microsoft Project, Primavera, or similar tools are always used in leading companies.

Because of the fact that requirements change at about 2% per calendar month, each of these activities must be performed in such a manner that changes are easy to accommodate during development; i.e. some form of iterative development is necessary for each major deliverable.

However, due to fixed delivery schedules that may be contractually set, it is also mandatory that large applications be developed with multiple releases in mind. At a certain point, all features for the initial release must be frozen, and changes occurring after that point must be added to follow-on releases. This expands the concept of iterative development to a multi-year, multi-release philosophy.

Effective Communications

Large software applications in the nominal 10,000 function point domain are always developed by teams that number from at least 50 personnel to more than 100 personnel. In addition, large applications are always built for dozens or even hundreds of users many of whom will be using the application in specialized ways.

It is not possible to build any large and complex product where dozens of personnel and dozens of users need to share information unless communication channels are very well planned and utilized. Communication needs are even greater when projects involve multiple subcontractors.

The state of the art for communication on nominal 10,000 function point project includes all of the following:

- Monthly status reports to corporate executives from project management
- Weekly progress reports to clients by vendor project management
- Daily communication between clients and the prime contractor
- Daily communication between the prime contractor and all sub contractors
- Daily communication between developers and development management
- Full email support among all participants
- Full voice support among all participants
- Video conference communication among remote locations
- Automated distribution of documents and source code among developers
- Automated distribution of change requests to developers
- Automated distribution of defect reports to developers
- Emergency or "red flag" communication for problems with a material impacts

For failing projects many of these communication channels were either not fully available or had gaps that tended to interfere with both communication and progress. For example cross-vendor communications may be inadequate to highlight problems. In addition, the status reports to executives may gloss over problems and conceal them rather than highlight causes for projected cost and schedule delays.

The fundamental purpose of good communications was encapsulated in a single phrase by Harold Geneen, the former Chairman of the ITT Corporation: "NO SURPRISES."

Capable Project Managers

When there are major delays or cost overruns for projects in the nominal 10,000 function point size range, project management problems are always present. Conversely, when projects have high productivity and quality levels, good project management is always observed. The state of the art for project management on large projects includes:

- Knowledge of sizing techniques such as function points

- Knowledge of formal estimating tools and techniques
- Knowledge of project planning tools and techniques
- Knowledge of risk analysis methods
- Knowledge of value analysis methods
- Knowledge of project measurement techniques
- Knowledge of milestone and cost tracking techniques
- Knowledge of change management techniques
- Knowledge of all forms of software quality control
- Knowledge of personnel management techniques
- Knowledge of the domain of the applications being developed

For the global software industry it appears that project management was a weak link and possibly the weakest link of all. For example for failing projects sizing by means of function points are seldom utilized. Formal estimating tools are not utilized. Although project planning tools may be used, projects often run late and over budget anyway. This indicates that the plans were deficient and omitted key assumptions such as the normal rate of requirements change, staff turnover, and delays due to high defect volumes found during testing.

The roles of management in outsource projects are more complex than the roles of management for projects developed internally. It is important to understand the differences between client management and vendor project management.

The active work of managing the project is that of the vendor project managers. It is their job to create plans and schedules, to create cost estimates, to track costs, to produce milestone reports, and to alert the client directors and senior client executives to the existence of potential problems.

The work of the client director or senior client executive centers around facilitation, funding, and approval or rejection of plans and estimates produced by the vendor's project manager.

The word "facilitation" means that the client director will provide access for the vendor to business and technical personnel for answering questions and gathering requirements. The client director may also provide to the vendor technical documents, office space, and sometimes tools and computer time.

The word "funding" means that the client director, after approval by corporate executives, will provide the money to pay for the project.

The word "approval" means that the client director will consider proposals, plans, and estimates created by the vendor and either accept them, reject them, or request that they be modified and resubmitted.

The main problems with failing projects seem to center around the approval role. Unfortunately, clients may be presented with a stream of optimistic estimates and schedule commitments by vendor project management and asked to approve them. Unfortunately this tends to lead to cumulative overruns and the reason for this deserves comment.

Once a project is underway, the money already spent on it will have no value unless the project is completed. Thus if a project is supposed to cost \$1,000,000 and but has a cost overrun that needs an additional \$100,000 for completion, the client is

faced with a dilemma. Either cancel the project and risk losing the accrued cost of a million dollars, or provide an additional 10% and bring the project to completion so that it returns positive value and results in a working application.

If this scenario is repeated several times, the choices become more difficult. If a project has accrued \$5,000,000 in costs and seems to need another 10%, both sides of the dilemma are more expensive. This is a key problem with projects that fail. Each time a revised estimate is presented, the vendor asserts that the project is nearing completion and needs only a small amount of time and some additional funds to bring it to full completion. This can happen repeatedly.

All corporations have funding criteria for major investments. Projects are supposed to return positive value in order to be funded. The value can consist of either revenue increases, cost reductions, or competitive advantage. A typical return on investment (ROI) for a software project in the United States would be about 3 to 1. That is, the project should return \$3.00 in positive value for every \$1.00 that is spent on the project.

During the course of development the accrued costs are monitored. If the costs begin to exceed planned budgets, then the ROI for the project will be diminished. Unfortunately for failing projects, the ability of client executives to predict the ROI can be damaged by inaccurate vendor estimating methods.

The root problem, of course, is that poor estimating methods are never realistic nor are the schedules: they are always optimistic. Unfortunately it can take several iterations before the reality of this pattern emerges. Each time a vendor presents revised estimates and schedules there may be no disclosure to clients of internal problems and risks that the vendor is aware of. Sometimes this kind of problem does not surface until litigation occurs, when all of the vendor records have to be disclosed and vendor personnel are deposed.

Capable Development Personnel

The software development domain is characterized by workers who usually have a fairly strong work ethic and reasonable competence in core activities such as detail design, programming, and unit testing. Many software personnel put in long hours and are fairly good in basic programming tasks. However to be successful on specific large 10,000 function point applications some additional skill sets are needed:

- Knowledge of application domains
- Knowledge of the data base packages, forms, tools, and products
- Knowledge of the skill sets of the subcontract companies
- Knowledge of joint application design (JAD) principles
- Knowledge of formal design inspections
- Knowledge of all programming languages utilized
- Knowledge of formal code inspections
- Knowledge of change control methods and tools
- Knowledge of performance measurement and optimization techniques
- Knowledge of testing methods and tools

When software technical problems occur they are more often related to the lack of specialized knowledge about the application domain or specific technical topics such

as performance optimization rather than to lack of knowledge of basic software development approaches.

There may also be lack of knowledge of key quality control activities such as inspections, JAD, and specialized testing approaches. In general, common programming tasks are not usually problems. The problems occur in areas where special knowledge may be needed, which brings up the next topic.

Use of Specialists

In many human activities specialization is a sign of technological maturity. For example, the practice of medicine, law, and engineering all encompass dozens of specialists. Software is not yet as sophisticated as the more mature professions, but specialization is now occurring in increasing numbers. From analyzing the demographics of large software production companies, from 20 to more than 90 specialized occupations now exist in the software industry.

What is significant about specialization in the context of 10,000 function point projects is that projects with a full complement of a dozen or more specialists have a better chance of success than those relying upon generalists.

The state of the art of specialization for nominal 10,000 function point projects would include the following specialist occupation groups:

- CASE and tool Specialists
- Configuration Control Specialists
- Cost Estimating Specialists
- Customer Liaison Specialists
- Customer Support Specialists
- Data Base Administration Specialists
- Data quality Specialists
- Decision Support Specialists
- Development specialists
- Domain Knowledge Specialists
- Education Specialists
- Function Point Specialists (certified)
- Graphical User Interface (GUI) Specialists
- Human Factors Specialists
- Integration Specialists
- Joint Application Design (JAD) Specialists
- Measurement Specialists
- Outsource Evaluation Specialists
- Package Evaluation Specialists
- Performance Specialists
- Project Cost Estimating Specialists
- Project Planning Specialists
- Quality Assurance Specialists
- Quality Measurement Specialists
- Reusability Specialists
- Risk Management Specialists
- Standards Specialists
- Systems Analysis Specialists
- Systems Support Specialists

- Technical Writing Specialists
- Testing Specialists

Senior project managers need to know what specialists are required, and should take active and energetic steps to find them. The domains where specialists usually outperform generalists include technical writing, testing, quality assurance, data base design, and performance optimization. For some tasks such as function point analysis, certification examinations are a prerequisite to doing the work at all. Really large projects also benefit from using planning and estimating specialists.

Both software development and software project management are now too large and complex for generalists to know everything needed in sufficient depth. The increasing use of specialists is a sign that the body of knowledge of software engineering and software management is expanding, which is a beneficial situation.

For the past 10 years U.S. and European companies have been outsourcing software development, maintenance, and help-desk activities to countries with lower labor costs such as India, China, Russia, and a number of others. In general it is important that outsource vendors utilize the same kinds of methods as in-house development, and in particular achieve excellence in quality control.

Reusable Material

There are at least 12 different software artifacts that lend themselves to reusability. Unfortunately, much of the literature on software reuse has concentrated only on reusing source code, with a few sparse and intermittent articles devoted to other topics such as reusable design.

The state of the art of developing nominal 10,000 function point projects includes substantial volumes of reusable materials. Following are the 12 artifacts that are potentially reusable for software projects:

- Reusable source code
- Reusable designs
- Reusable test cases
- Reusable data
- Reusable requirements
- Reusable cost estimates
- Reusable screens
- Reusable project plans
- Reusable test plans
- Reusable architecture
- Reusable user documents
- Reusable human interfaces

One of the common advantages of using an outsource vendor is that these vendors are often very sophisticated in reuse and have many reusable artifacts available. However, reuse is most often encountered in areas where the outsource vendor is a recognized expert. For example, an outsource vendor that specializes in insurance applications and has worked with a dozen property and casualty insurance companies probably has accumulated enough reusable materials to build any arbitrary insurance application with at least 50% reusable components.

Software reuse is a key factor in reducing costs and schedules and improving quality. However, reuse is a two-edged sword. If the quality levels of the reusable materials are good, then reusability has one of the highest returns on investment of any known software technology. But if the reused materials are filled with bugs or error, the ROI can become very negative. In fact, reuse of high quality or poor quality materials tends to produce the greatest extreme range of ROI of any known technology: plus or minus 300% have been observed.

Software reusability is often cited as a panacea that will solve software's sluggish schedules and high costs. This may be true theoretically, but reuse will have no practical value unless the quality levels of the reusable materials approach zero defects.

User Involvement (cited from the Standish Chaos Report)

It is not possible to design and build nominal 10,000 function point business applications without understanding the requirements of the users. Further, when the application is under development users normally participate in reviews and also assist in trials of specific deliverables such as screens and documents. User may also review or even participate in the development of prototypes for key inputs, outputs, and functions. For any major application the state of the art of user involvement includes:

- Participation in Joint Application Design (JAD) sessions
- Participation in requirements reviews
- Participation in change control boards
- Participation in reviewing documents produced by the contractors
- Participation in design reviews
- Participation in using prototypes and sample screens produced by the contractors
- Participation in training classes to learn the new application
- Participation in defect reporting from design through testing
- Participation in acceptance testing

User involvement is time consuming but valuable. On average, user effort totals to about 20% of the effort put in by the software technical team. The range of user involvement can top 50% at the high end, and be less than 5% at the low end. However, for large and complex projects if the user involvement totals to less than about 10% of the effort expended by the development team the project will be at some risk of having poor user satisfaction when it is finally finished.

Executive Management Support (cited from the Standish Chaos Report)

The topic of executive management support of new applications varies with the overall size of the application. For small projects below about 500 function points executive involvement may hover around zero because these projects are so low in cost and low in risk as to be well below the level of executive interest.

However, for large applications in the 10,000 function point range executive scrutiny is the norm. It is an interesting phenomenon that the frequent failure of large software projects has caused a great deal of distrust of software managers by corporate executives.

In the software industry overall, the state of the art of executive management support indicates the following:

- Approving the return on investment (ROI) calculations for software projects
- Providing funding for software development projects
- Assigning key executives to oversight and project director roles
- Reviewing milestone, cost, and risk status reports
- Determining if overruns or delays have reduced the ROI below corporate targets

Even if executives perform all of the roles that normally occur there can still be problems and failures. A key failing of software projects is that executives cannot reach good business decisions if they are provided with disinformation rather than accurate status reports. If software project status reports and risk assessments gloss over problems and technical issues then executives cannot control the project with the precision that they would like. Thus inadequate reporting and less than candid risk assessments will delay the eventual and prudent executive decision to try and limit further expenses by terminating projects that are out of control.

It is a normal corporate executive responsibility to ascertain why projects are running out of control. One of the reasons why executives at many large corporations distrust software is because software projects have a tendency to run out of control and often fail to provide accurate status reports. As a class, the corporate executives that I have met are more distrustful of software organizations than almost any other corporate group under their management control. Unfortunately corporate executives appear to have many reasons for being distrustful of software managers after so many delays and cost overruns.

Clear Requirements (cited from the Standish Chaos Report)

Although clear requirements are a laudable goal, they almost never occur for nominal 10,000 function point software applications. The only projects I have observed where the initial requirements were both clear and unchanging were for specialized small applications below 500 function points in size

Businesses are too dynamic for requirements to be completely unchanged for large applications. Many external events such as changes in tax laws, changes in corporate structure, business process reengineering, or mergers and acquisitions can trigger changes in software application requirements. The situation is compounded by the fact that large applications take several years to develop. It is unrealistic to expect that a corporation can freeze all of its business rules for several years merely to accommodate the needs of a software project.

The most typical scenario for dealing with the requirements of a nominal 10,000 function point application would be to spend several months in gathering and analyzing the initial requirements. Then as design proceeds, new and changed requirements will arrive at a rate of roughly 2% per calendar month. The total volume of requirements surfacing after the initial requirements exercise will probably approach or even exceed 50%. These new and changing requirements will eventually need to be stopped for the first release of the application, and requirements surfacing after about 9 to 12 months will be aimed at follow-on releases of the application.

The state of the art for gathering and analyzing the requirements for 10,000 function point projects include the following:

- Utilization of Joint Application Design (JAD) for initial requirements gathering
- Utilization of prototypes for key features of new applications
- Ensuring that requirements are clearly expressed and can be understood
- Utilization of formal requirement inspections with both users and vendors
- Creation of a joint client/vendor change control board
- Selection of domain experts for changes to specific features
- Ensuring that requirements traceability is present
- Multi-release segmentation of requirements changes
- Utilization of automated requirements analysis tools
- Careful analysis of the features of packages that will be part of the application

The lowest rates of requirements changes observed on 10,000 function point projects are a little below one half of one percent a month, with an accumulated total of less than 10% compared to the initial requirements. However, the maximum amount of growth has topped 200%.

The concurrent use of JAD sessions, careful analysis of requirements, requirements inspections, and prototypes can go far to bring the requirements process under technical and management control.

SUMMARY AND CONCLUSIONS

There are hundreds of ways to make large software systems fail. There are only a few ways of making them successful.

Among the most important software development practices are those dealing with planning and estimating before the project starts, with absorbing changing requirements during the project, and with successfully handling bugs or defects. Successful projects using state of the art methods are always excellent in all three of these critical activities: estimating, change control, and quality control. By contrast, projects that run late or fail usually had optimistic estimates, did not anticipate changes, and failed to control quality.

REFERENCES AND READINGS

Ewusi-Mensah, Kweku; Software Development Failures; MIT Press, Cambridge, MA; 2003; ISBN 0-26205072-2/276 pages.

Garmus, David and Herron, David; Function Point Analysis – Measurement Practices for Successful Software Projects; Addison Wesley Longman, Boston, MA; 2001; ISBN 0-201-69944-3; 363 pages.

Glass, R.L.; Software Runaways: Lessons Learned from Massive Software Project Failures; Prentice Hall, Englewood Cliffs; 1998.

International Function Point Users Group (IFPUG); IT Measurement – Practical Advice from the Experts; Addison Wesley Longman, Boston, MA; 2002; ISBN 0-201-74158-X; 759 pages.

- Johnson, James et al; The Chaos Report; The Standish Group, West Yarmouth, MA; 2000.
- Jones, Capers; Applied Software Measurement; McGraw Hill, 2nd edition 1996; ISBN 0-07-032826-9; 618 pages.
- Jones, Capers; Assessment and Control of Software Risks; Prentice Hall, 1994; ISBN 0-13-741406-4; 711 pages.
- Jones, Capers; Patterns of Software System Failure and Success; International Thomson Computer Press, Boston, MA; December 1995; 250 pages; ISBN 1-850-32804-8; 292 pages.
- Jones, Capers; Software Quality – Analysis and Guidelines for Success; International Thomson Computer Press, Boston, MA; ISBN 1-85032-876-6; 1997; 492 pages.
- Jones, Capers; Estimating Software Costs; McGraw Hill, New York; ISBN 0-07-9130941; 1998; 725 pages.
- Jones, Capers; Software Assessments, Benchmarks, and Best Practices; Addison Wesley Longman, Boston, MA; ISBN 0-201-48542-7; 2000; 657 pages.
- Jones, Capers: "Sizing Up Software;" Scientific American Magazine, Volume 279, No. 6, December 1998; pages 104-111.
- Jones, Capers; Conflict and Litigation Between Software Clients and Developers; Software Productivity Research, Inc.; Burlington, MA; 2001; 40 pages.
- Kan, Stephen H.; Metrics and Models in Software Quality Engineering, 2nd edition; Addison Wesley Longman, Boston, MA; ISBN 0-201-72915-6; 2003; 528 pages.
- Radice, Ronald A.; High Quality Low Cost Software Inspections; Paradoxicon Publishing Andover, MA; ISBN 0-9645913-1-6; 2002; 479 pages.
- Wieggers, Karl E.; Peer Reviews in Software – A Practical Guide; Addison Wesley Longman, Boston, MA; ISBN 0-201-73485-0; 2002; 232 pages.
- Yourdon, Ed; Death March - The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects; Prentice Hall PTR, Upper Saddle River, NJ; ISBN 0-13-748310-4; 1997; 218 pages.
- Yourdon, Ed; Outsource: Competing in the Global Productivity Race; Prentice Hall PTR, Upper Saddle River, NJ; ISBN 0-13-147571-1; 2005; 251 pages.