AD8001

SOFTWARE DEVELOPMENT PROCESSES

UNIT I SOFTWARE PROCESS

Software Process Maturity Software maturity Framework, Principles of Software Process Change, Software Process Assessment, The Initial Process, The Repeatable Process, The Defined Process, The Managed Process, The Optimizing Process. Process Reference Models Capability Maturity Model (CMM), CMMI, PCMM, PSP, TSP).

UNIT II SOFTWARE ECONOMICS AND LIFECYCLE

Software Project Management Renaissance Conventional Software Management, Evolution of Software Economics, Improving Software Economics, The old way and the new way. Life-Cycle Phases and Process artifacts Engineering and Production stages, inception phase, elaboration phase, construction phase, transition phase, artifact sets, management artifacts, engineering artifacts and pragmatic artifacts, model-based software architectures.

UNIT III SOFTWARE PROCESSES PLANNING 9

Workflows and Checkpoints of process Software process workflows, Iteration workflows, Major milestones, minor milestones, periodic status assessments. Process Planning Work breakdown structures, Planning guidelines, cost and schedule estimating process, iteration planning process, Pragmatic planning.

UNIT IV PROJECT MANAGEMENT AND METRICS

Project Organizations Line-of- business organizations, project organizations, evolution of organizations, process automation. Project Control and process instrumentation The seven-core metrics, management indicators, quality indicators, life-cycle expectations, Pragmatic software metrics, metrics automation.

UNIT V UNIT TITLE 9

CCPDS-R Case Study and Future Software Project Management Practices Modern Project Profiles, Next-Generation software Economics, Modern Process Transitions.

Unit 5 –REAL TIME APPLICATIONS

5.1 - CCPDS-R Case Study and Future Software Project Management Practices

A detailed case study of a successful software project that followed many of the techniques discussed in this topic. Successful here means on budget, on schedule, and satisfactory to the customer. The Command Center Processing and Display Sys-tem-Replacement (CCPDS-R) project was performed for the U.S. Air Force by TRW Space and Defense in Redondo Beach, California. The entire project included systems engineering, hardware procurement, and software development, with each of these three major activities consuming about one-third of the total cost. The schedule spanned 1987 through 1994.

The software effort included the development of three distinct software systems totaling more than one million source lines of code. This case study focuses on the initial software development, called the Common Subsystem, for which about 355,000 source lines were developed. The Common Subsystem effort also produced a reusable architecture, a mature process, and an integrated environment for efficient development of the two software subsystems of roughly similar size that followed. This case study therefore represents about one-sixth of the overall CCPDS-R project effort.

Although this case study does not coincide exactly with the management process presented in this book nor with all of today's modern technologies, it used most of the same techniques and was managed to the same spirit and priorities. TRW delivered

Key Points a An objective case study is a true indicator of a mature organization and a mature project process. The software industry needs more case studies like CGPDS-R.

Unit 5 –REAL TIME APPLICATIONS

The metrics histories were all derived directly from the artifacts of the project's process. These data were used to manage the project and were embraced by practitioners, managers, and stakeholders.

CCPDS-R was one of the pioneering projects that practiced many modern management approaches.

The system on budget and on schedule, and the users got more than they expected. TRW was awarded the Space and Missile Warning Systems Award for Excellence in 1991 for "continued, sustained performance in overall systems engineering and project execution." A project like CCPDS-R could be developed far more efficiently today. By incorporating current technologies and improved processes, environments, and levels of automation, this project could probably be built today with equal quality in half the time and at a quarter of the cost.

Some of today’s popular software cost models are not well matched to an iterative software process focused an architecture-first approach Many cost estimators are still using a conventional process experience base to estimate aq modern project profile A next- generation software cost model should explicitly separate architectural engineering from application production, just as an architecture-first process does. Two major improvements in next- generation software cost estimation models: Separation of the engineering stage from the production stage will force estimators to differentiate between architectural scale and implementation size. Rigorous design notations such as UML will offer an opportunity to define units of measure for scale that are more standardized and therefore can be automated and tracked.

Modern Software Economics: Changes that provide a good description of what an organizational manager should strive for in making the transition to a modern process: 1. Finding and fixing a software problem after delivery costs 100 times more than fixing the problem in early design phases 2.

• Can compress software development schedules 25% of nominal, but no more. 3. For every $1 you spend on development, you will spend $2 on maintenance. 4. Software development and maintenance costs are primarily a function of the number of source lines of code 5. Variations among people account for the biggest differences in software productivity. 6. The overall ratio of software to hardware costs is still growing – in 1955 it was 15:85; in 1985 85:15 7. Only about 15% of software development effort is devoted to programming 8. Software systems and products typically cost 3 times as much per SLOC as individual software programs. 9. Walkthroughs catch 60% of the errors. 10. 80% of the contribution comes from 20% of the contributors.

5.2 Modern Project Profiles

A modern process framework exploits several critical approaches for resolving these issues:

1. Protracted integration and late design breakage are resolved by forcing integration into the engineering stage. This is achieved through continuous integration of an architecture baseline supported by executable demonstrations of the primary scenarios.

2. Late risk resolution is resolved by emphasize Key Points

▲ In modern project profiles, positive and negative trends will be more tangible early in the life cycle.

▲ Integration that occurs early and continuously serves as the verification activity of the design artifacts.

▲ Most of the critical risks will be resolved by the end of the elaboration phase. The construction and transition phases, during which there is generally the most cost risk, should be free of surprises.

▲ Major milestones focus on demonstrated results.

sizing an architecture-first approach, in which the high-leverage elements of the system are elaborated early in the life cycle.

3. The analysis paralysis of a requirements-driven functional decomposition is avoided by organizing lower-level specifications along the content of releases rather than along the product decomposition (by subsystem, by component, etc.).

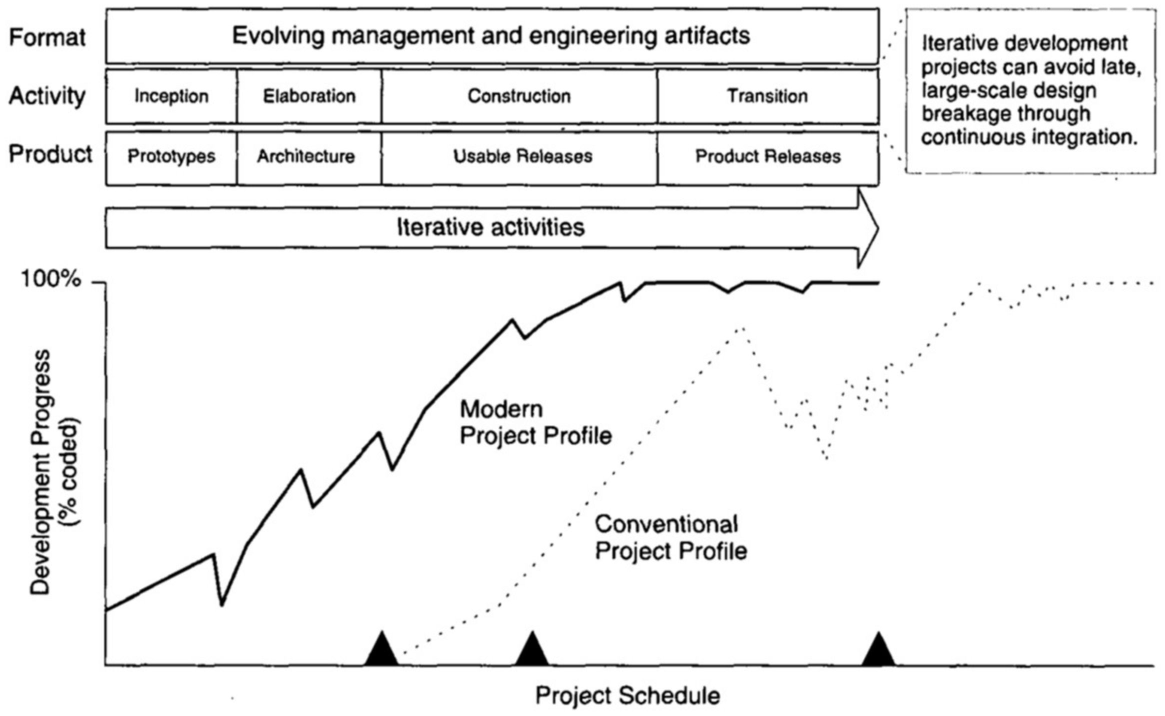
4. Adversarial stakeholder relationships are avoided by providing much more tangible and objective results throughout the life cycle.

5. The conventional focus on documents and review meetings is replaced by a focus on demonstrable results and well-defined sets of artifacts, with more-rigorous notations and extensive automation supporting a paperless environment.

Continuous Integration

Iterative development produces the architecture first, allowing integration to occur as the verification activity of the design phase and enabling design flaws to be detected and resolved earlier in the life cycle. This approach avoids the big-bang integration at the end of a project by stressing continuous integration throughout the project. Figure 15-1 illustrates the differences between the progress profile of a healthy modern project and that of a typical conventional project, which was introduced in Figure 1-2. The architecture-first approach forces integration into the design phase through the construction of demonstrations. The demonstrations do not eliminate the design breakage; instead, they make it happen in the engineering stage, when it can be resolved efficiently in the context of life-cycle goals. The downstream integration nightmare, late patches, and shoe-horned software fixes are avoided. The result is a more robust and maintainable design.

Unit 5 –REAL TIME APPLICATIONS


Format
Evolving management and engineering artifacts
Activity
Inception
Elaboration
Construction
Transition
Product
Prototypes
Architecture
Usable Releases
Product Releases
Iterative development projects can avoid late, large-scale design breakage through continuous integration.
Iterative activities
100%
Development Progress (% coded)
Modern Project Profile
Conventional Project Profile
Project Schedule

Teamwork Among Stakeholders

Many aspects of the classic development process cause stakeholder relationships to degenerate into mutual distrust, making it difficult to balance requirements, product features, and plans. A more iterative process, with more-effective working relationships

Artifacts
Software
Vision
Document
Release Specifications

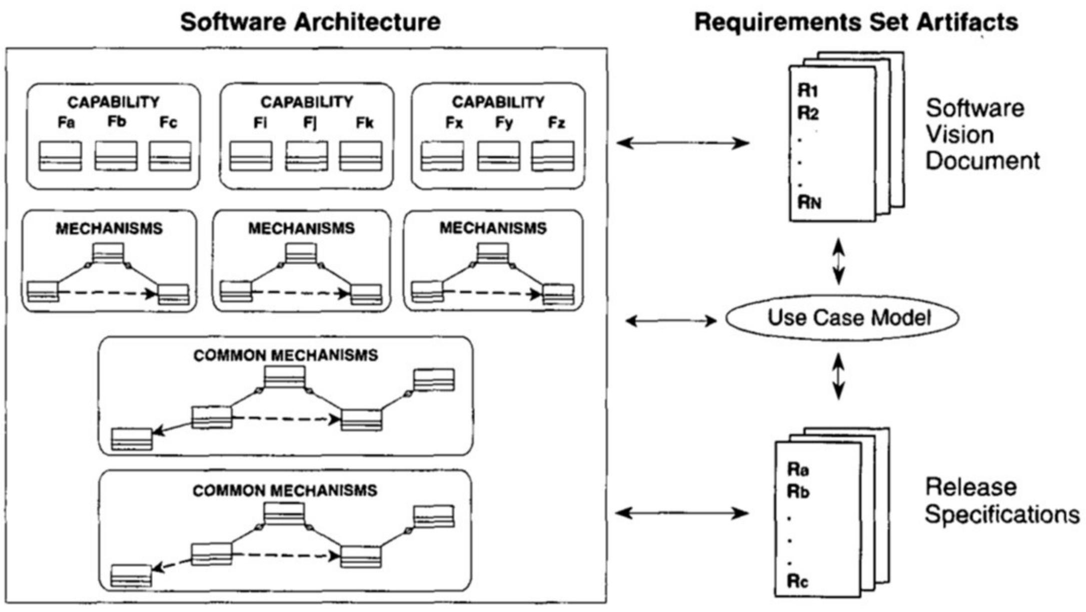
Figure 15-3. Organization of software components resulting from a modern process between stakeholders, allows trade-offs to be based on a more objective understanding by everyone. This process requires that customers, users, and monitors have both applications and software expertise, remain focused on the delivery of a usable system (rather than on blindly enforcing standards and contract terms), and be willing to allow the contractor to make a profit with good performance. It also requires a development organization that is focused on achieving customer satisfaction and high product quality in a profitable manner.

The-transition from the exchange of mostly paper artifacts to demonstration of intermediate results is one of the crucial mechanisms for promoting teamwork among stakeholders. Major milestones provide tangible results and feedback from a usage point of view. As Table 15-2 shows, designs are now guilty until proven innocent: The project does not move forward until the objectives of the demonstration have been achieved. This prerequisite does not preclude the renegotiation of objectives once the demonstration and major milestone results permit further understanding of the tradeoffs inherent in the requirements, design, plans, and technology.

In Table 15-2, the apparent results may still have a negative connotation. A modern iterative process that focuses on demonstrable results (rather than just briefings and paper) requires all stakeholders to be educated in the important distinction between apparently negative results and evidence of real progress. For example, a design flaw discovered early, when the cost to resolve it is tenable, can often be viewed as positive progress rather than as a major issue.

Software Management Best Practices

Many software management best practices have been captured by various authors and industry organizations. One of the most visible efforts has been the Software Acquisition Best Practices Initiative, sponsored by the U.S. Department of Defense to "improve and restructure our software acquisition management process." Brown summarized the initiative [Brown, 1996], which has three components: the Airlie Software Council (composed of software industry gurus), seven different issue panels



The nine best practices are described next, with my commentary on how they resonate with the process framework, management disciplines, and top 10 principles that I have recommended. (Quotations are presented in italics.)

1. Formal risk management.
a Using an iterative process that confronts risk is more or less what this is saying. 2. Agreement on interfaces.

a While we may use different words, this is exactly the same intent as my architecture- first principle. Getting the architecture baselined forces the project to gain agreement on the various external interfaces and the important internal interfaces, all of which are inherent in the architecture.

3. Formal inspections.

a The assessment workflow throughout the life cycle, along with the other engineering workflows, must balance several different defect removal strategies. The least important strategy, in terms of breadth, should be formal inspection, because of its high costs in human resources and its low defect discovery rate for the critical architectural defects that span multiple components and temporal complexity.

4. Metric-based scheduling and management.

a This important principle is directly related to my model-based notation and objective quality control principles. Without rigorous notations for artifacts, the measurement of progress and quality degenerates into subjective estimates.

5. Binary quality gates at the inch-pebble level.

a This practice is easy to misinterpret. Too many projects have taken exactly this approach early in the life cycle and have laid out a highly detailed plan at great expense. Three months later, when some of the requirements change or the architecture changes, a large percentage of the detailed planning must be rebaselined. A better approach would be to maintain fidelity of the plan commensurate with an understanding of the requirements and the architecture. Rather than inch pebbles, I recommend establishing milestones in the engineering stage followed by inch pebbles in the production stage. This is the primary message behind my evolving levels of ' detail principle.

6. Programwide visibility of progress versus plan.

a This practice—namely, open communications among project team members—is obviously necessary. None of my principles traces directly to this practice. It seems so obvious, I let it go without saying.

7. Defect tracking against quality targets.

a This important principle is directly related to my architecture-first and objective quality control principles. The make-or-break defects and quality targets are architectural. Getting a handle on these qualities early and tracking their trends are requirements for success.

8. Configuration management.

a The Airlie Software Council emphasized configuration management as key to controlling complexity and tracking changes to all artifacts. It also recognized that automation is important because of the volume and dynamics of modern; large-scale projects, which make manual methods cost-prohibitive and error-prone. The same reasoning is behind my change management principle.

9. People-aware management accountability.

5.3 Next Generation Software Economics

- ⍰ Software experts hold widely varying opinions about software economics and its manifestation in software cost estimation models:
    - ⍰ Source Lines of code Vs Function Points
    - ⍰ Economy of scale Vs Diseconomy of scale
    - ⍰ Productivity Vs Quality
    - ⍰ Java Vs C++
    - ⍰ Object Oriented Vs Functionally oriented.
    - ⍰ Commercial components Vs Custom development
- ⍰ Some of today’s popular software cost models are not well matched to an iterative software process focused on an architecture-first approach.

- ? This topic provides perspective on how a software cost model should be structured to best support the estimation of a modern software process.

- ? This software cost model is theoretical-based.

- ? No empirical evidence to demonstrate that this approach will be more accurate than today's cost models.

- ? A next generation software cost model should explicitly separate architectural engineering from application production.

Unit 5 –REAL TIME APPLICATIONS

- ? The cost of designing, producing, testing and maintaining the architecture baseline is a function of scale, quality, technology, process and team skill.

- ? There should be some diseconomy of scale (exponent greater than 1.0) in the architecture cost model because it is inherently driven by research and development-oriented concerns.

- ? When an organization achieved a stable architecture, the production costs should be an exponential function of size, quality and complexity, with a much more stable range of process and personnel influence.

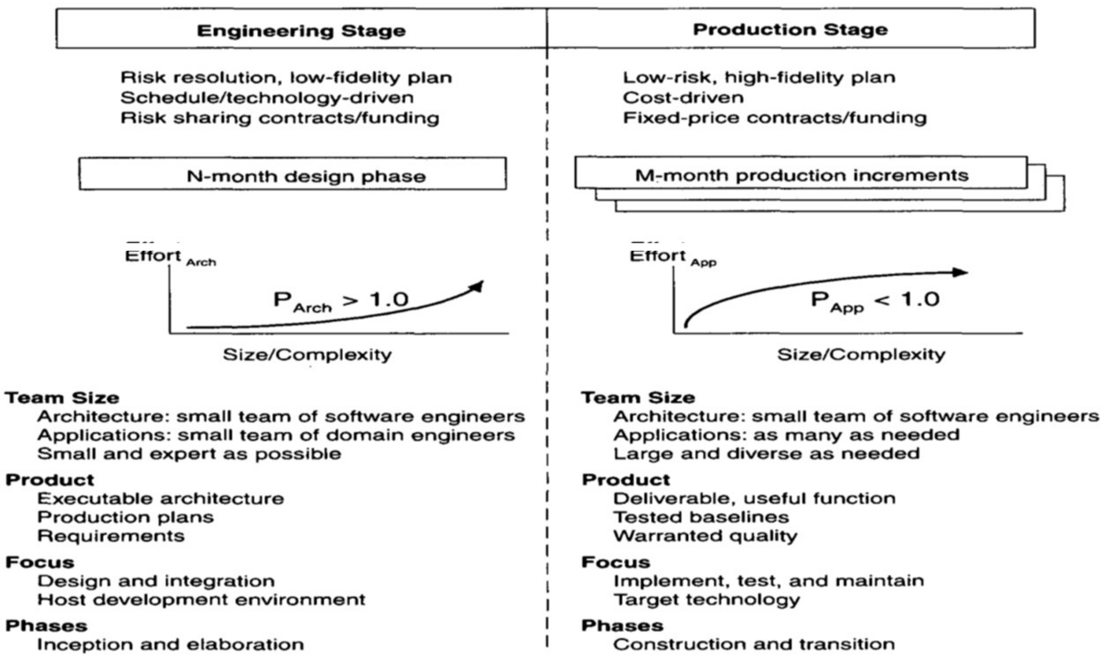
- ? The production stage cost model should reflect an economy of scale (exponent less than 1.0) similar to that of conventional economic models for bulk production of commodities.

• Figure summarizes a hypothesized cost model for an architecture-first development process.

$$\text{Effort} = F(T_{Arch}, S_{Arch}, Q_{Arch}, P_{Arch}) + F(T_{App}, S_{App}, Q_{App}, P_{App})$$

$$\text{Time} = F(P_{Arch}, \text{Effort}_{Arch}) + F(P_{App}, \text{Effort}_{App})$$

where:
- T = technology parameter (environment automation support)
- S = scale parameter (such as use cases, function points, source lines of code)
- Q = quality parameter (such as portability, reliability, performance)
- P = process parameter (such as maturity, domain experience)



Engineering Stage
Production Stage
Risk resolution, low-fidelity plan
Schedule/technology-driven
Risk sharing contracts/funding
Low-risk, high-fidelity plan
Cost-driven
Fixed-price contracts/funding
N-month design phase
M-month production increments
Effort Arch
P Arch > 1.0
Size/Complexity
Effort App
P App < 1.0
Size/Complexity
Team Size
Architecture: small team of software engineers
Applications: small team of domain engineers
Small and expert as possible
Product
Executable architecture
Production plans
Requirements
Focus
Design and integration
Host development environment
Phases
Inception and elaboration
Team Size
Architecture: small team of software engineers
Applications: as many as needed
Large and diverse as needed
Product
Deliverable, useful function
Tested baselines
Warranted quality
Focus
Implement, test, and maintain
Target technology
Phases
Construction and transition

FIGURE 16-1. *Next-generation cost models*

Unit 5 –REAL TIME APPLICATIONS

- Next-Generation software cost models should estimate large-scale architectures with economy scale. This implies that the process exponent during the production stage will be less than 1.0.

- Larger the system, more opportunity is to exploit automation and to reuse common processes, components and architectures.

- ? Next-Generation environments and infrastructures are moving to automate and standardize many of these management activities, thereby requiring a lower percentage of effort for overhead activities as scale increases.

- ? Reusing common processes across multiple iterations of a single project, multiple releases of a single product, or multiple projects in an organization also relieves many of the sources of diseconomy of scale.

- ? Critical sources of scrap and rework are eliminated by applying precedent experience and mature processes.

- ? While most reuse of components results in reducing the size of the production effort, the reuse of processes, tools and experience has a direct impact on the economies of scale.

- ? Another important difference in this cost model is that architectures and applications have different units of mass (scale Vs size) and are representations of the solution space.

- ? Scale might be measured in terms of architecturally significant elements (classes, components, processes, nodes)

- ? Size might be measured in SLOC or megabytes of executable code.

- ? These measures differ from measures of the problem space.

• There are many solutions to given any problem as shown in the figure. 16.2

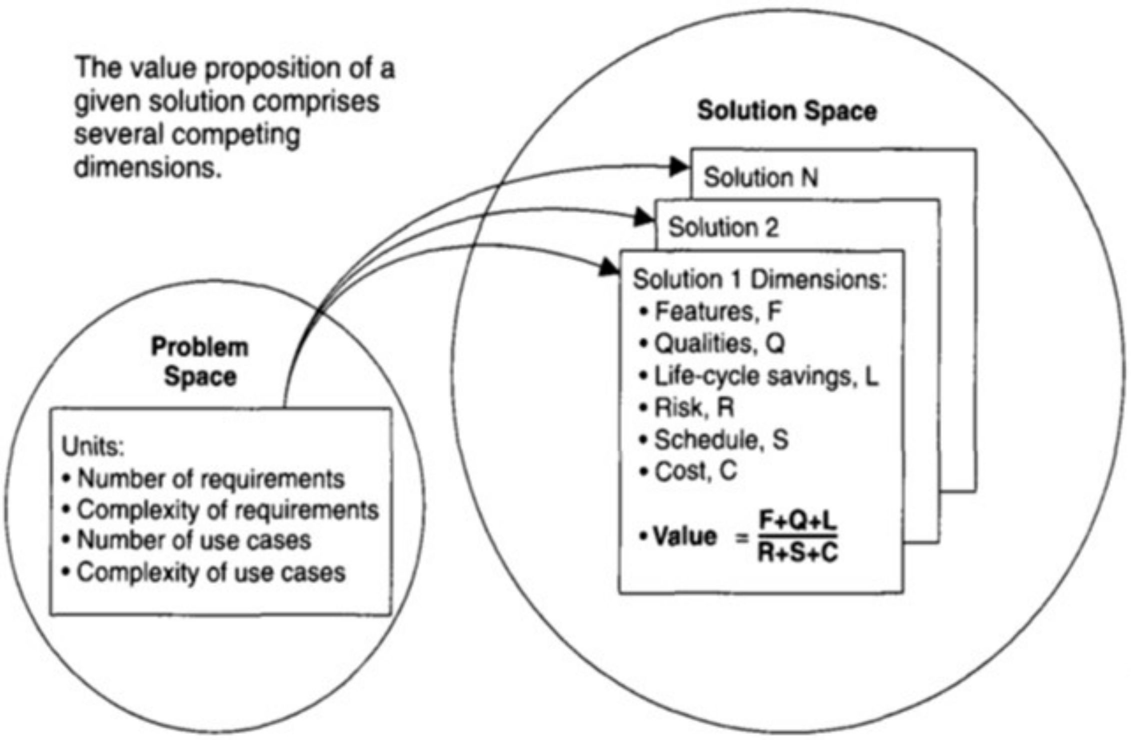


FIGURE 16-2. *Differentiating potential solutions through cost estimation*

- ? Cost is the key discriminator among potential solutions.

- ? Cost estimates that are more accurate and more precise can be derived from specific solutions to problems.

- ? Therefore, the cost estimation model must be governed by the basic parameters of a candidate solution.

- ? Functional points Vs SLOC is a good indicator of the need for measures of both scale and size.

- ? Functional points are more accurate at quantifying the scale of the architecture required, while SLOC more accurately depicts the size of the components that make up the total implementation.

- ? SLOC can be easily be automated and precision can be achieved easily. Many projects use SLOC as a successful measure of size in the later phases of the life cycle.

- ? The value of function points is that they are better at depicting the overall scale of the solution, independently of the actual size and implementation language of the final realization.

- ? Two major improvements in next-generation software cost estimation models:

  - ? Separation of the engineering stage from the production stage will force estimators to differentiate between architectural scale and implementation size. This will permit greater accuracy and more-honest precision in life cycle estimates.

  - ? Rigorous design notations such as UML will offer an opportunity to define units of measure for scale that are more standardized and therefore can be automated and tracked. These measures can also be traced more straight forwardly into the costs of production.

- ? Quantifying the scale of the software architecture in the engineering stage is an area ripe for research.

- ? Two breakthroughs in the software process seem possible, both of them realized through technology advances in the supporting environment.

- ? First breakthrough would be availability of integrated tools that automate the transition of information between requirements, design, implementation and deployment elements. These tools allow more comprehensive round-trip engineering among the engineering artifacts.

- ? The second breakthrough would focus on collapsing today's four sets of fundamental technical artifacts into three sets by automating the activities associated with human-generated source code thereby eliminating the need for a separate implementation set.

As shown in figure 16-3 would allow executable programs to be produced directly from UML representations without human intervention.

5.4 Modern Process Transitions

Successful software management is hard work. Technical breakthroughs, process breakthroughs, and new tools will make it easier, but management discipline will continue to be the crux of software project success. New technological advances will be accompanied by new opportunities for software applications, new dimensions of complexity, new avenues of automation, and new customers with different priorities. Accommodating these changes will perturb many of our ingrained software management values and priorities. However, striking a balance among requirements, designs, and plans will remain the underlying objective of future software management endeavors, just as it is today.

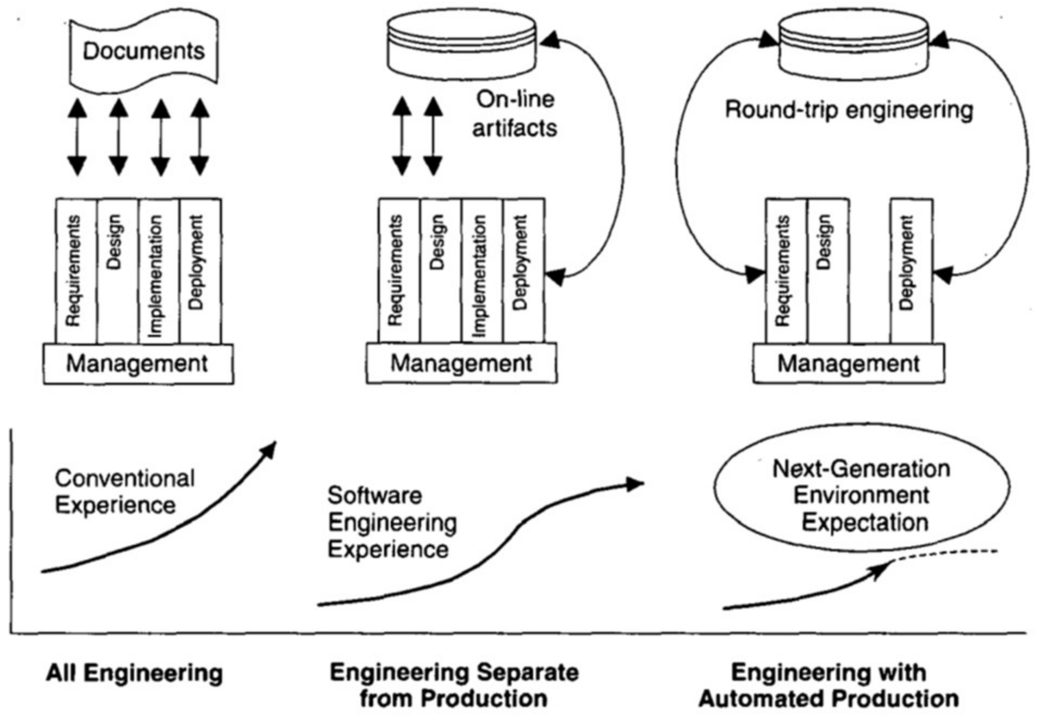

Documents
On-line artifacts
Round-trip engineering
Requirements
Design
Implementation
Deployment
Management
Conventional Experience
Software Engineering Experience
Next-Generation Environment Expectation
All Engineering
Engineering Separate from Production
Engineering with Automated Production

FIGURE 16-3. *Automation of the construction process in next-generation environments*

numerous projects have been practicing some of these disciplines for years. However, many of the techniques and disciplines suggested herein will necessitate a significant paradigm shift. Some of these changes will be resisted by certain stakeholders or by certain factions within a project or organization. It is not always easy to separate cultural resistance from objective resistance. summarizes some of the important culture shifts to be prepared for in order to avoid as many sources of friction as possible in transitioning successfully to a modern process.

Key Points
The transition to modern software

processes and technologies necessitates several culture shifts that will not always be easyto achieve.

Lessons learned in transitioning organizations to a modern process have exposed several recurring themes of success that represent important culture shifts from conventional practice.

A significant transition should be attempted on a significant project. Pilot projects do not generally attract top talent, and top talent is crucial to the success of any significant transition.

PART - A Questions

1. Differentiate Leaders and managers. (CO5/K1) Leaders- set direction, do the right thing Managers- Follow process, do things right.

2. Define charter. (CO5/K1)

A charter is a documentation that formally recognizes the existence of a project.

3. Give some unites for measuring the size of the software. (CO5/K1)

Lines of code (LOC), Function points, feature points, number of bubbles on the data flow diagram, number of entities on entity relationship diagram.

4. Write the any two advantages of LOC. (CO5/K1)

- It is widely used and universally accepted.

- LOC are easily measured upon project completion.
  5. What are dependencies? (CO5/K1)
  Dependencies are one form of constraints for any project. Dependencies are any relationship connections between activities in a project that may impact their scheduling.
  6. Write the special types of relationship in dependencies. (CO5/K1)

- Lag and lead relationship

- Hard versus soft relationship
  7. Define project portfolio? (CO5/K1)
  Project portfolio is group of project carried out under this sponsorship and/or management.
  8. What is modified code? (CO5/K1)
  The code developed for previous application that is suitable for a new application after a modest amount of modification.
  9. Give any two examples for product attributes. (CO5/K1)
  Database size (DATA), Product complexity (CPLX)

PART - A Questions 10. What is modified code? (CO5/K1)

The code developed for previous application that is suitable for a new application after a modest amount of modification.

11. Give any two examples for product attributes. (CO5/K1)

Database size (DATA), Product complexity (CPLX)

12. What is Legacy code? (CO5/K1)

Code developed for a previous application that is believed to be of use for a new application.

13. What is ROI? (CO5/K1)

The Return on Investment is a calculation of the difference between the stream of benefits and the stream of costs over the life of the system, discounted by the applicable interest rate.

14. Write the types of roles. (CO5/K1)

Database designers, Configuration Management experts, Human interface Designers, Web Masters, Network Specialists, System architects, Programming language experts.

15. What are the Managerial activities? (CO5/K1)

Project planning, tracking, control, risk analysis.

Part-B Questions

1	Explain in detail about the next generation	CO5	K2
2	Discuss in detail about modern transition	CO5	K2
3	Explain in detail the Software Project Management techniques used in the	CO5	K2
4	Explain in detail about modern project	CO5	K2