Unit-5

CCPDS-R Case Study

This represents a detailed case study of a successful software project that followed many of the techniques.

- Successful here means on budget, on schedule, and satisfactory to the customer.
- The Command Center Processing and Display System-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.
- 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. a This case study provides a practical context that is relevant to the techniques, disciplines.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.

Modern Project Profiles

Differences in workflow cost allocations between a conventional process and a modern process

SOFTWARE ENGINEERING WORKFLOWS	CONVENTIONAL PROCESS EXPENDITURES	MODERN PROCESS EXPENDITURES
Management	5%	10%
Environment	5%	10%
Requirements	5%	10%
Design	10%	15%
Implementation	30%	25%
Assessment	40%	25%
Deployment	5%	5%
Total	100%	100%

Continuous Integration

The continuous integration inherent in an iterative development process enables better insight into quality trade-offs.

System characteristics that are largely inherent in the architecture (performance, fault tolerance, maintainability) are tangible earlier in the process, when issues are still correctable

Early Risk Resolution

Conventional projects usually do the easy stuff first, modern process attacks the important 20%

of the requirements, use cases, components, and risks.

The effect of the overall life-cycle philosophy on the 80/20 lessons provides a useful risk management

perspective.

80% of the engineering is consumed by 20% of the requirements.

80% of the software cost is consumed by 20% of the components

80% of the errors are caused by 20% of the components

80% of the progress is made by 20% of the people.

Evolutionary Requirements

Conventional approaches decomposed system requirements into subsystem requirements, subsystem requirements into

component requirements, and component requirements into unit requirements

The organization of requirements was structured so traceability was simple

Most modern architectures that use commercial components,

legacy components, distributed resources and object-oriented methods are not trivially traced to the requirements they satisfy.

The artifacts are now intended to evolve along with the process,with more and more fidelity as the progresses life-cycle and the requirements understanding matures

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

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

Top 10 Software Management Principles

1. Base the process on an architecture-first approach – rework rates remain stable over the project life cycle**.**

2.Establish an ***iterative life-cycle process*** that confronts risk early

3.Transition design methods to emphasize ***component-based development***

4.Establish a ***change management environment*** – *the dynamics of iterative development, including concurrent workflows by different teams working on shared artifacts, necessitate highly controlled baselines*

5.Enhance change freedom through tools that support ***round-trip engineering***

Software Management Best Practices:

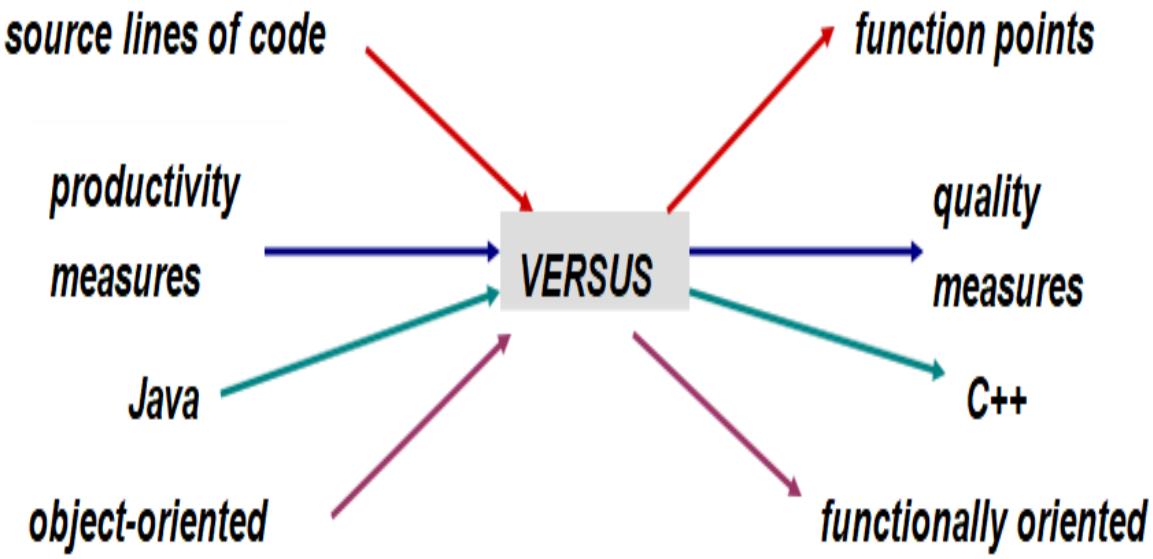
- There is nine best practices:

1. *Formal risk management*
2. *Agreement on interfaces*
3. *Formal inspections*
4. *Metric-based scheduling and management*
5. *Binary quality gates at the inch-pebble level*
6. *Program-wide visibility of progress versus plan.*
7. *Defect tracking against quality targets*
8. *Configuration management*
9. *People-aware management accountability*

Next-Generation Software Economics

Next-Generation Cost Models

- Software experts hold widely varying opinions about software economics and its manifestation in software cost estimation models:



- It will be difficult to improve empirical estimation models while the project data going into these models are noisy and highly uncorrelated, and are based on differing process and technology foundations.

"Software Project Management" Walker 90/112

- 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 a 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. You 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.

Modern Process Transitions

Culture Shifts

- Several culture shifts must be overcome to transition successfully to a modern software management process:
  - Lower level and mid-level managers are performers
  - Requirements and designs are fluid and tangible
  - Good and bad project performance is much more obvious earlier in the life cycle
  - Artifacts are less important early, more important later
  - Real issues are surfaced and resolved systematically
  - Quality assurance is everyone's job, not a separate discipline
  - Performance issues arise early in the life cycle
  - Investments in automation is necessary
  - Good software organization should be more profitable

Denouement

- Good way to transition to a more mature iterative development process that supports automation technologies

and modern architectures is to take the following shot:

- ***Ready.***
  Do your homework. Analyze modern approaches and technologies. Define your process. Support it with mature environments, tools, and components. Plan thoroughly.
- ***Aim.***
  Select a critical project. Staff it with the right team of complementary resources and demand improved results.
- ***Fire.***
  Execute the organizational and project-level plans with vigor and follow-through.