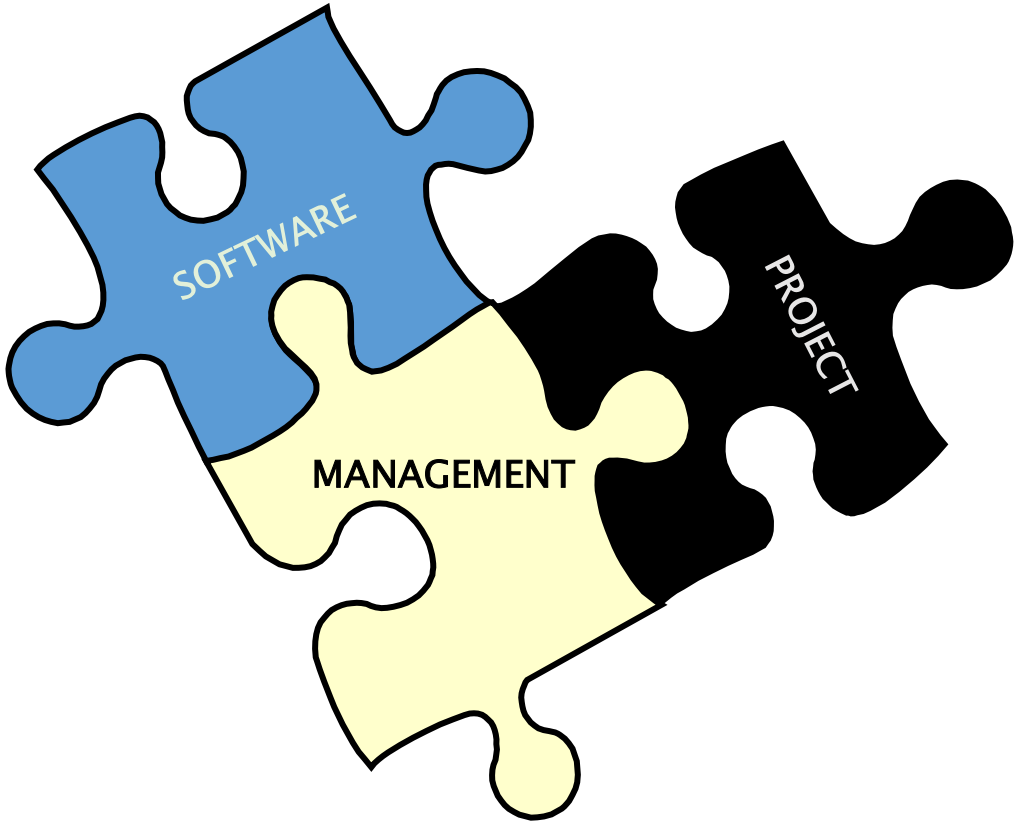
Software Project Management



Unit 4. Modern Approach to Software Project and Economics

Prepared By:

BIJAY MISHRA

(बिजय मिश्र)

biizay@gmail.com

@jijibisha

UNIT 4. Modern Approach to Software Project and Economics - 4 Hrs.

4.1 Elements of Modern Software Projects and Management Principles

4.2 Next-Generation Software Economics and Cost Models

4.3 Modern Process Transition – Paradigm Shifts

4.1 Elements of Modern Software Projects and Management Principles

Modern Project Profiles

Continuous Integration

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%

Modern Project Profiles

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.

Modern Project Profiles

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.

Modern Project Profiles

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 life-cycle progresses and the requirements understanding matures.

Modern Project Profiles

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.
- The process with more-effective working relationships between stakeholders requires that customers, users and monitors have both applications and software expertise, remain focused on the delivery of a usable system
- 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.

Modern Project Profiles

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***

Modern Project Profiles

Top 10 Software Management Principles

6. Capture design artifacts in rigorous, ***model-based notation***

7. Instrument the process for ***objective quality control*** and progress assessment

8. Use a ***demonstration-based approach*** to asses intermediate artifacts

9. Plan intermediate releases in groups of usage scenarios with ***evolving levels of detail***

10. Establish a ***configurable process*** that is economically scalable

Modern Project Profiles

Software Management Best Practices

- There is nine best practices:

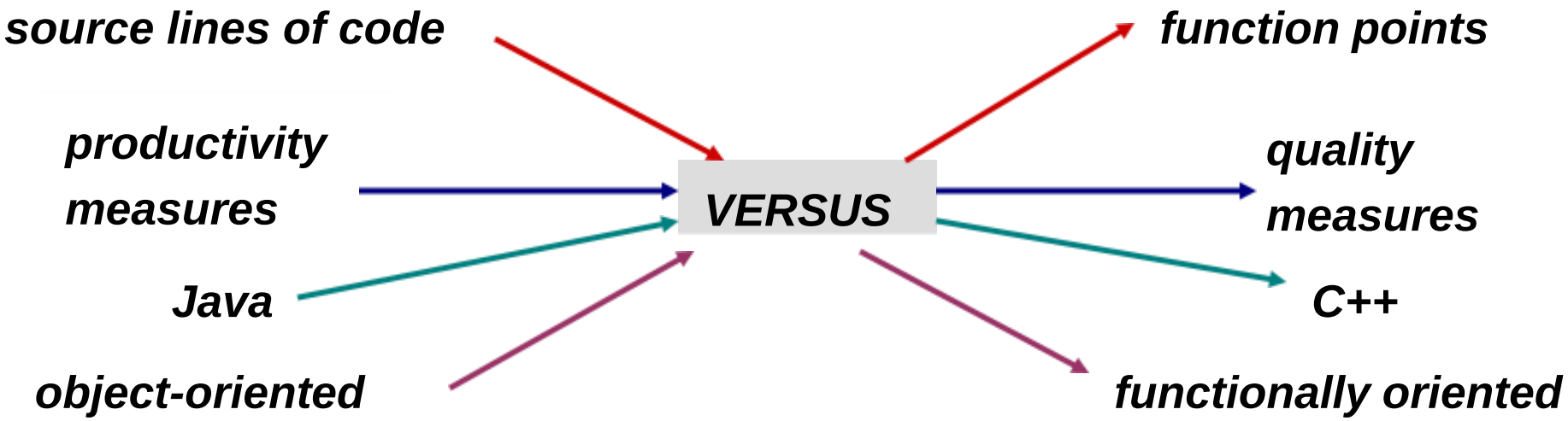
1. ***Formal risk management***
2. ***Agreement on interfaces***
3. ***Formal inspections***
4. ***Metric-based scheduling and management***
5. ***Binary quality gates at the inch-pebble level***
6. ***Program-wide visibility of progress versus plan.***
7. ***Defect tracking against quality targets***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

4.2 Next-Generation Software Economics and Cost Models

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.

Next-Generation Software Economics

Next-Generation Cost Models

- 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.**

Next-Generation Software Economics

Next-Generation Cost Models

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

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

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)

Next-Generation Software Economics

Next-Generation Cost Models

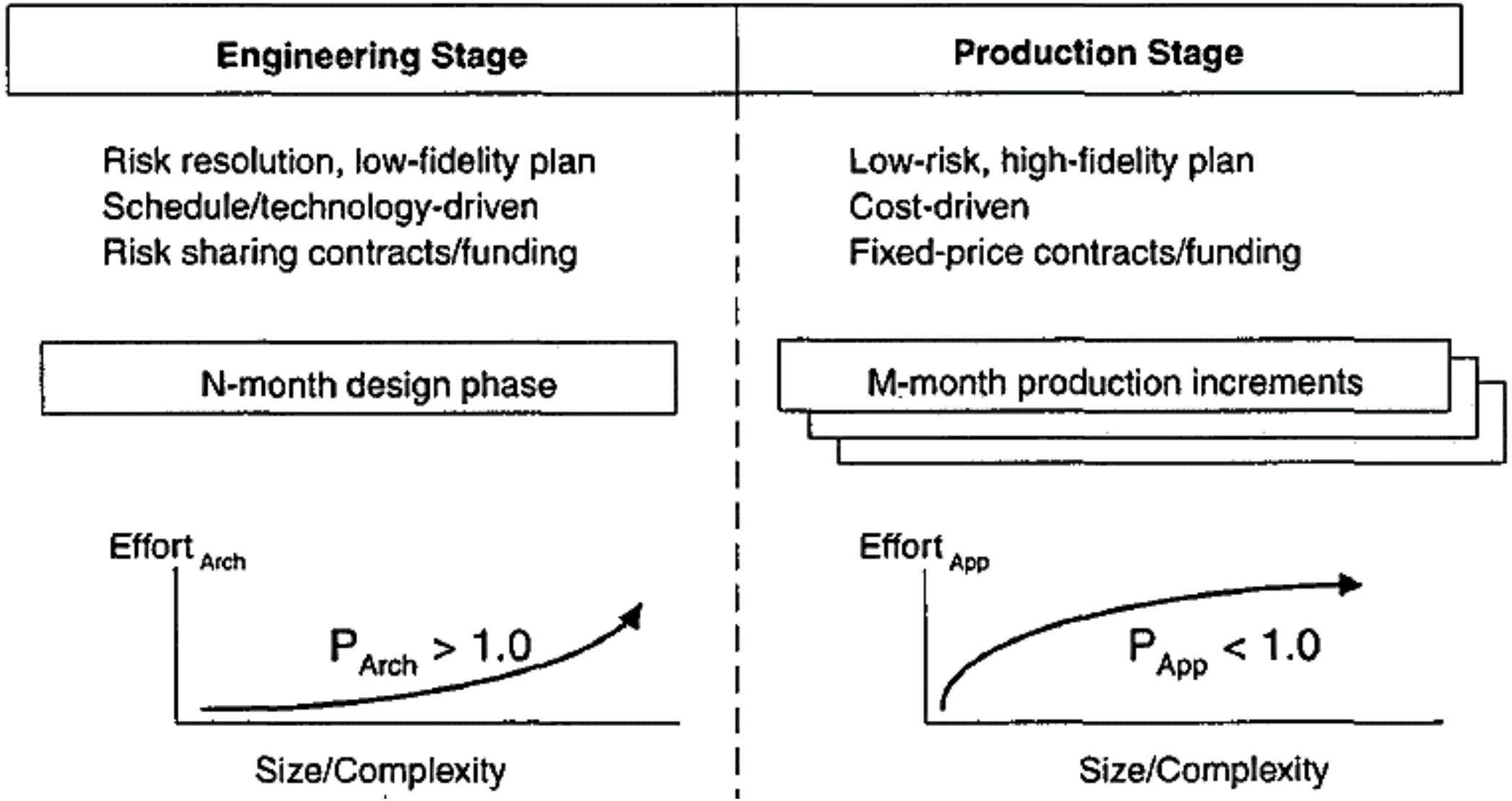


Figure: Next-Generation Cost Models

Next-Generation Software Economics

Next-Generation Cost Models

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: Next-Generation Cost Models

Next-Generation Software Economics

Next-Generation Cost Models

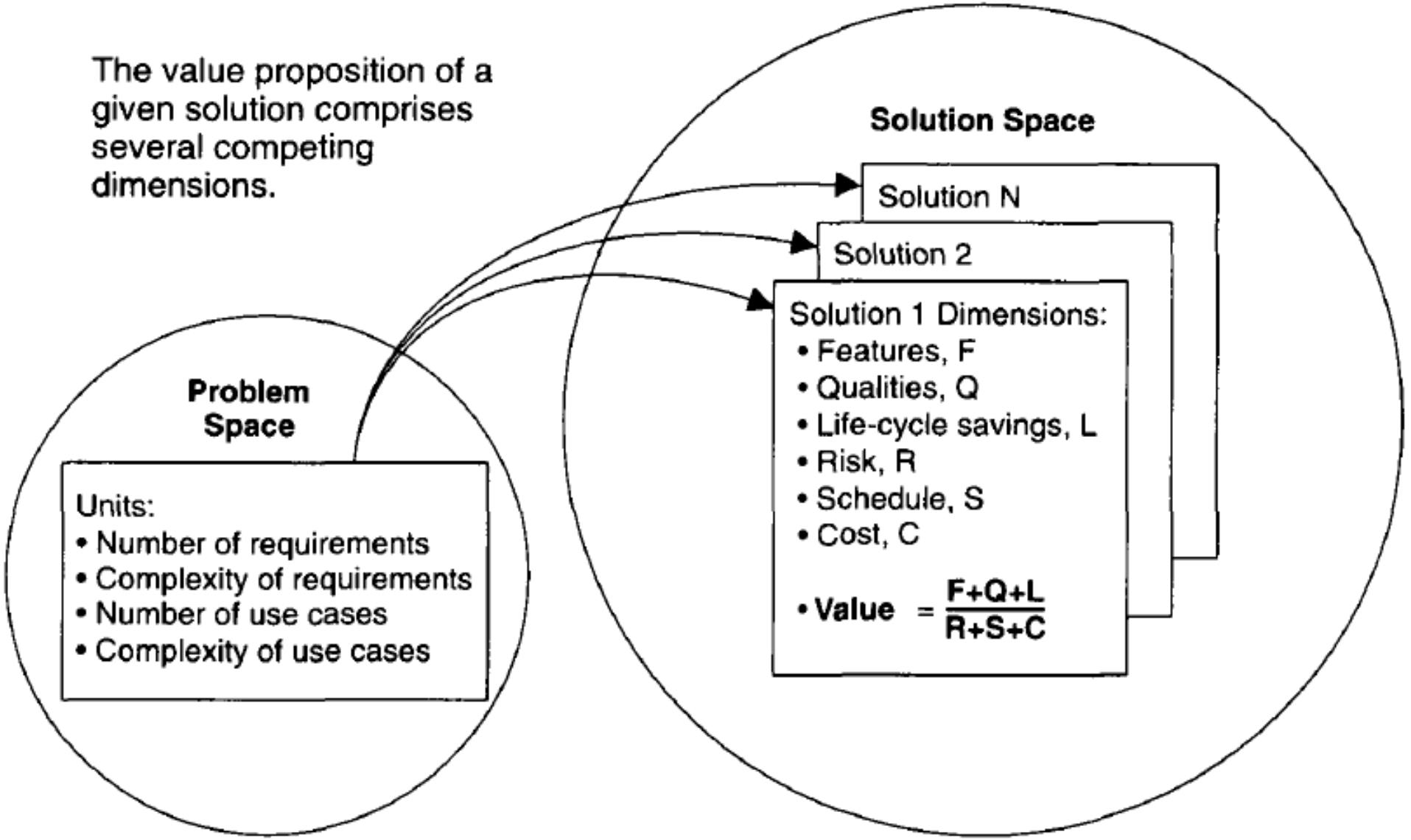


Figure: Differentiating Potential Solutions through Cost Estimation

Next-Generation Software Economics

Next-Generation Cost Models

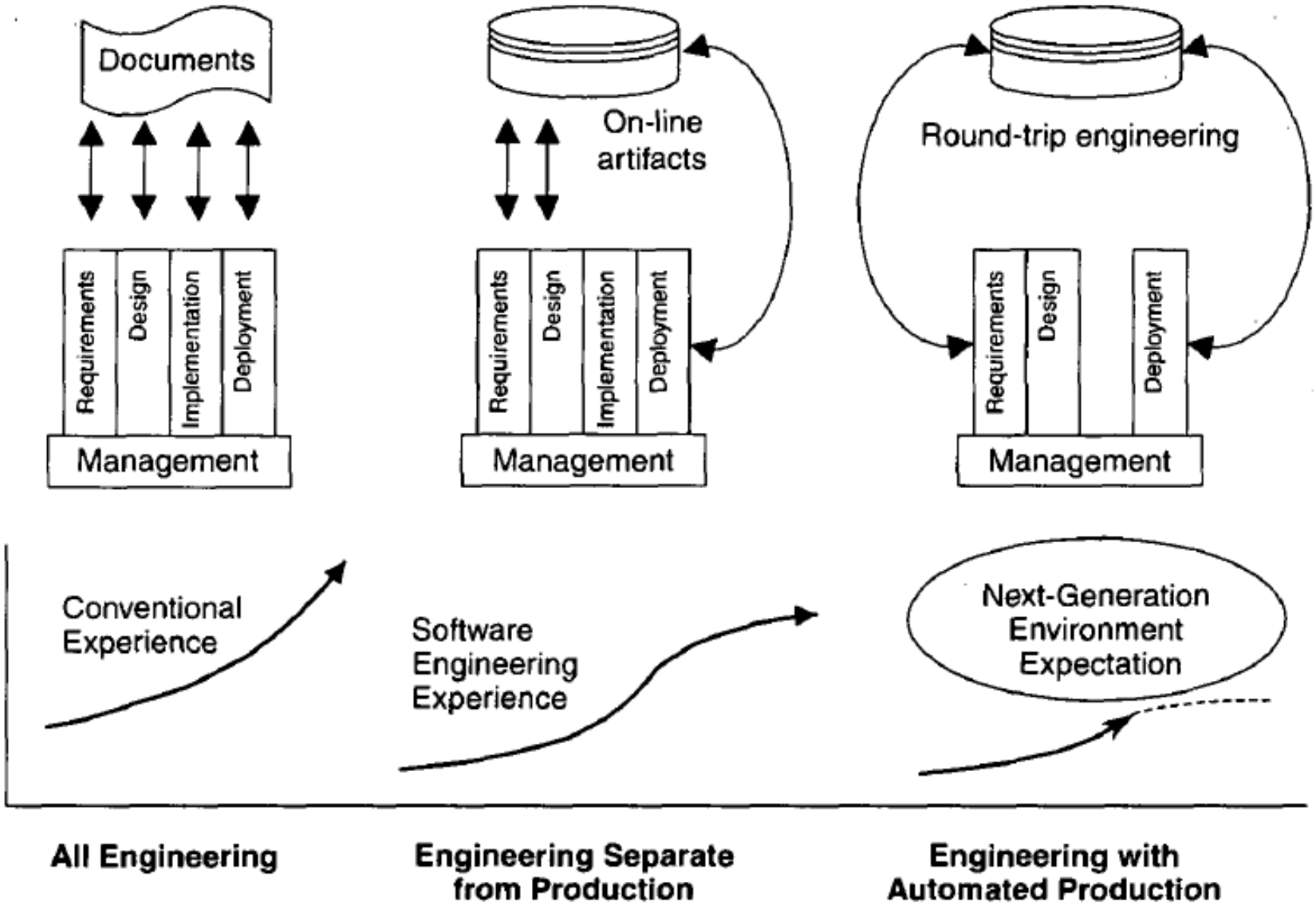


Figure: Automation of Construction Process in next-generation environments

Next-Generation Software Economics

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:
  - ✓ Finding and fixing a software problem after delivery costs 100 times more than fixing the problem in early design phases
  - ✓ You can compress software development schedules 25% of nominal, but no more.
  - ✓ For every $1 you spend on development, you will spend $2 on maintenance.
  - ✓ Software development and maintenance costs are primarily a function of the number of source lines of code.

Next-Generation Software Economics

Modern Software Economics

- ✓ Variations among people account for the biggest differences in software productivity.
- ✓ The overall ratio of software to hardware costs is still growing – in 1955 it was 15:85; in 1985 85:15.
- ✓ Only about 15% of software development effort is devoted to programming.
- ✓ Software systems and products typically cost 3 times as much per SLOC as individual software programs.
- ✓ Walkthroughs catch 60% of the errors.
- ✓ 80% of the contribution comes from 20% of the contributors.

4.3 Modern Process Transition – Paradigm Shifts

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.

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

Modern Process Transitions

Denouement

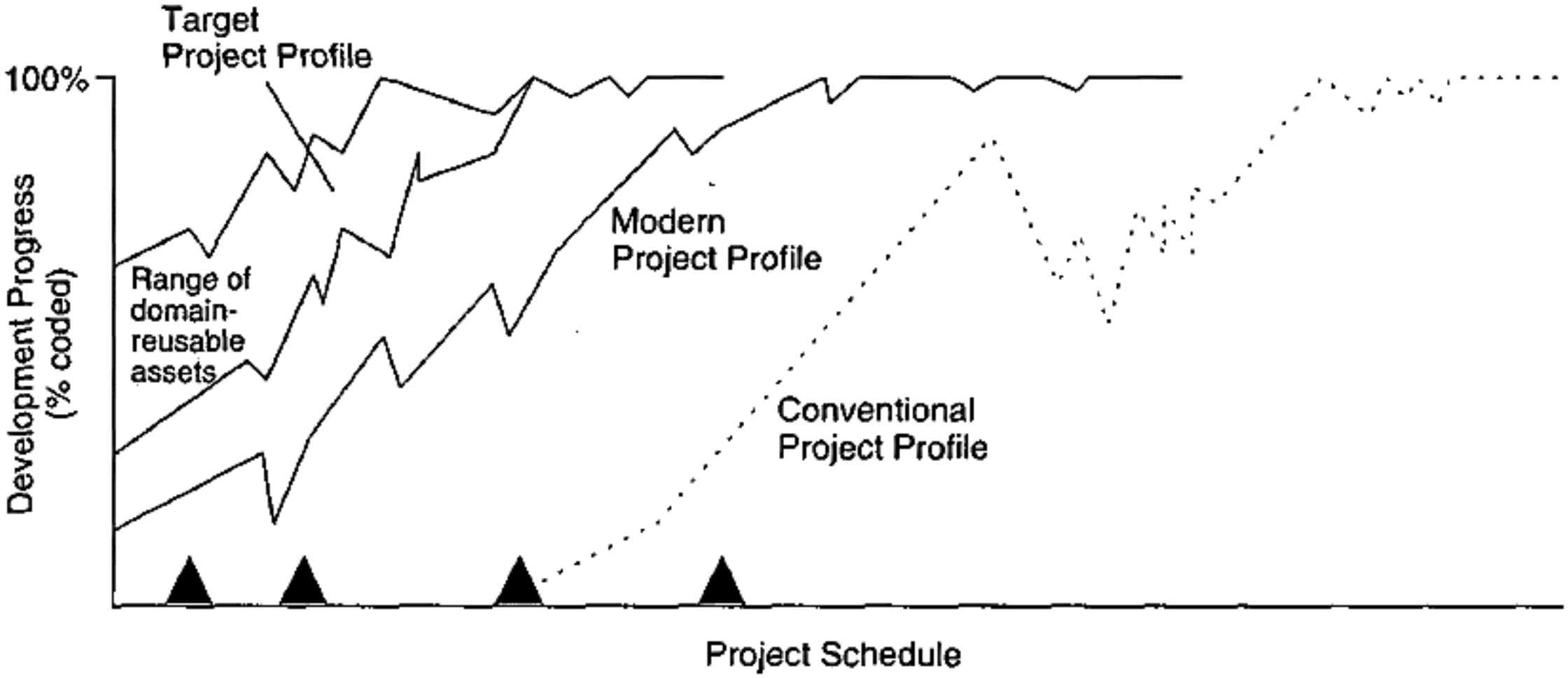


Figure: Next-Generation Project Performance

Modern Process Transitions

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.


Any Questions
?

Book References:

1. Software Project Management – A Unifies Framework, Walker Royce, 1998, Addison Wesley
2. Software Project Management – From Concept to Deployment, Conway, K., 2001.
3. Software Project Management, Bob Hughes and Mike Cotterell, Latest Publication
4. Software Project Management, Rajeev Chopra, 2009
5. Software Engineering – A Practitioner's approach, Roger S. Pressman Latest Plublication
6. Managing Global Software Projects, Ramesh, 2001, TMH


Thank You