Supplementary Slides for
*Software Engineering:
A Practitioner's Approach, 6/e*
Part 4

Software Engineering: A Practitioner's Approach, 6/e

Chapter 21
Project Management Concepts

The 4 P's

- **People** — the most important element of a successful project
- **Product** — the software to be built
- **Process** — the set of framework activities and software engineering tasks to get the job done
- **Project** — all work required to make the product a reality

3

Stakeholders

- *Senior managers* who define the business issues that often have significant influence on the project.
- *Project (technical) managers* who must plan, motivate, organize, and control the practitioners who do software work.
- *Practitioners* who deliver the technical skills that are necessary to engineer a product or application.
- *Customers* who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
- *End-users* who interact with the software once it is released for production use.

4

Software Teams

How to lead?

How to organize?

How to collaborate?

How to motivate?

How to create good ideas?
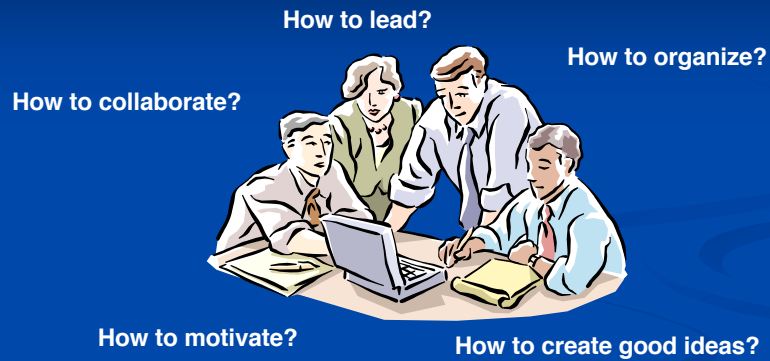
Team Leader

- The MOI Model
  - **Motivation.** The ability to encourage (by "push or pull") technical people to produce to their best ability.
  - **Organization.** The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
  - **Ideas or innovation.** The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

Software Teams

The following factors must be considered when selecting a software project team structure ...

- the difficulty of the problem to be solved
- the size of the resultant program(s) in lines of code or function points
- the time that the team will stay together (team lifetime)
- the degree to which the problem can be modularized
- the required quality and reliability of the system to be built
- the rigidity of the delivery date
- the degree of sociability (communication) required for the project

Organizational Paradigms

- closed paradigm—structures a team along a traditional hierarchy of authority
- random paradigm—structures a team loosely and depends on individual initiative of the team members
- open paradigm—attempts to structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradigm
- synchronous paradigm—relies on the natural compartmentalization of a problem and organizes team members to work on pieces of the problem with little active communication among themselves

suggested by Constantine [CON93]

Avoid Team "Toxicity"

- A frenzied work atmosphere in which team members waste energy and lose focus on the objectives of the work to be performed.
- High frustration caused by personal, business, or technological factors that cause friction among team members.
- "Fragmented or poorly coordinated procedures" or a poorly defined or improperly chosen process model that becomes a roadblock to accomplishment.
- Unclear definition of roles resulting in a lack of accountability and resultant finger-pointing.
- "Continuous and repeated exposure to failure" that leads to a loss of confidence and a lowering of morale.

Agile Teams

- Team members must have trust in one another.
- The distribution of skills must be appropriate to the problem.
- Mavericks may have to be excluded from the team, if team cohesiveness is to be maintained.
- Team is "self-organizing"
  - An adaptive team structure
  - Uses elements of Constantine's random, open, and synchronous paradigms
  - Significant autonomy

Team Coordination & Communication

- *Formal, impersonal approaches* include software engineering documents and work products (including source code), technical memos, project milestones, schedules, and project control tools (Chapter 23), change requests and related documentation, error tracking reports, and repository data (see Chapter 26).
- *Formal, interpersonal procedures* focus on quality assurance activities (Chapter 25) applied to software engineering work products. These include status review meetings and design and code inspections.
- *Informal, interpersonal procedures* include group meetings for information dissemination and problem solving and "collocation of requirements and development staff."
- *Electronic communication* encompasses electronic mail, electronic bulletin boards, and by extension, video-based conferencing systems.
- *Interpersonal networking* includes informal discussions with team members and those outside the project who may have experience or insight that can assist team members.

11

The Product Scope

- Scope
    - **Context.** How does the software to be built fit into a larger system, product, or business context and what constraints are imposed as a result of the context?
    - **Information objectives.** What customer-visible data objects (Chapter 8) are produced as output from the software? What data objects are required for input?
    - **Function and performance.** What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?
- Software project scope must be unambiguous and understandable at the management and technical levels.

12

Problem Decomposition

- Sometimes called *partitioning* or *problem elaboration*
- Once scope is defined …
    - It is decomposed into constituent functions
    - It is decomposed into user-visible data objects
    or
    - It is decomposed into a set of problem classes
- Decomposition process continues until all functions or problem classes have been defined

The Process

- Once a process framework has been established
    - Consider project characteristics
    - Determine the degree of rigor required
    - Define a task set for each software engineering activity
        - Task set =
            - Software engineering tasks
            - Work products
            - Quality assurance points
            - Milestones

Melding the Problem and the Process

Figure 21.2 Melding the Problem and the Process

15

The Project

- Projects get into trouble when …
    - Software people don't understand their customer's needs.
    - The product scope is poorly defined.
    - Changes are managed poorly.
    - The chosen technology changes.
    - Business needs change [or are ill-defined].
    - Deadlines are unrealistic.
    - Users are resistant.
    - Sponsorship is lost [or was never properly obtained].
    - The project team lacks people with appropriate skills.
    - Managers [and practitioners] avoid best practices and lessons learned.

16

Common-Sense Approach to Projects

- *Start on the right foot.* This is accomplished by working hard (very hard) to understand the problem that is to be solved and then setting realistic objectives and expectations.
- *Maintain momentum. The* project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team's way.
- *Track progress.* For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using formal technical reviews) as part of a quality assurance activity.
- *Make smart decisions.* In essence, the decisions of the project manager and the software team should be to "keep it simple."
- *Conduct a postmortem analysis.* Establish a consistent mechanism for extracting lessons learned for each project.

To Get to the Essence of a Project

- Why is the system being developed?
- What will be done?
- When will it be accomplished?
- Who is responsible?
- Where are they organizationally located?
- How will the job be done technically and managerially?
- How much of each resource (e.g., people, software, tools, database) will be needed?

Barry Boehm

Critical Practices

- Formal risk management
- Empirical cost and schedule estimation
- Metrics-based project management
- Earned value tracking
- Defect tracking against quality targets
- People aware project management

Software Engineering: A Practitioner's Approach, 6/e

Chapter 22
Process and Project Metrics

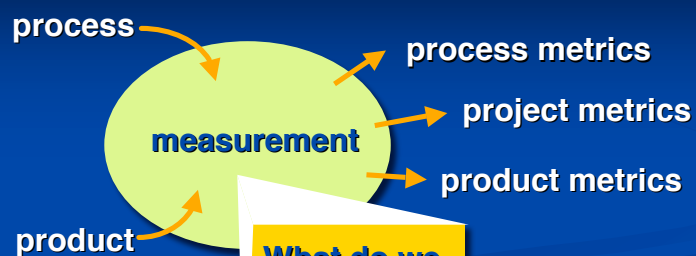copyright © 1996, 2001, 2005
R.S. Pressman & Associates, Inc.

For University Use Only
May be reproduced ONLY for student use at the university level
when used in conjunction with *Software Engineering: A Practitioner's Approach.*
Any other reproduction or use is expressly prohibited.

1

A Good Manager Measures

process → process metrics

project metrics

measurement

product metrics

product →

What do we use as a basis?
- **size?**
- **function?**

2

Why Do We Measure?

- assess the status of an ongoing project
- track potential risks
- uncover problem areas before they go "critical,"
- adjust work flow or tasks,
- evaluate the project team's ability to control quality of software work products.

3

Process Measurement

- We measure the efficacy of a software process indirectly.
  - That is, we derive a set of metrics based on the outcomes that can be derived from the process.
  - Outcomes include
    - measures of errors uncovered before release of the software
    - defects delivered to and reported by end-users
    - work products delivered (productivity)
    - human effort expended
    - calendar time expended
    - schedule conformance
    - other measures.
- We also derive process metrics by measuring the characteristics of specific software engineering tasks.

4

Process Metrics Guidelines

- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who collect measures and metrics.
- Don't use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them.
- Never use metrics to threaten individuals or teams.
- Metrics data that indicate a problem area should not be considered "negative." These data are merely an indicator for process improvement.
- Don't obsess on a single metric to the exclusion of other important metrics.

Software Process Improvement

Process Metrics

- Quality-related
  - focus on quality of work products and deliverables
- Productivity-related
  - Production of work-products related to effort expended
- Statistical SQA data
  - error categorization & analysis
- Defect removal efficiency
  - propagation of errors from process activity to activity
- Reuse data
  - The number of components produced and their degree of reusability

Project Metrics

- used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.
- every project should measure:
  - *inputs*—measures of the resources (e.g., people, tools) required to do the work.
  - *outputs*—measures of the deliverables or work products created during the software engineering process.
  - *results*—measures that indicate the effectiveness of the deliverables.

Typical Project Metrics

- Effort/time per software engineering task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on software engineering tasks

9

Metrics Guidelines

- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who have worked to collect measures and metrics.
- Don't use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them.
- Never use metrics to threaten individuals or teams.
- Metrics data that indicate a problem area should not be considered "negative." These data are merely an indicator for process improvement.
- Don't obsess on a single metric to the exclusion of other important metrics.

10

Typical Size-Oriented Metrics

- errors per KLOC (thousand lines of code)
- defects per KLOC
- $ per LOC
- pages of documentation per KLOC
- errors per person-month
- Errors per review hour
- LOC per person-month
- $ per page of documentation

11

Typical Function-Oriented Metrics

- errors per FP (thousand lines of code)
- defects per FP
- $ per FP
- pages of documentation per FP
- FP per person-month

12

Comparing LOC and FP

Programming Language	LOC per Function point			
	avg.	median	low	high
Ada	154	-	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	63	53	77	-
JavaScript	58	63	42	75
Perl	60	-	-	-
PL/1	78	67	22	263
Powerbuilder	32	31	11	105
SAS	40	41	33	49
Smalltalk	26	19	10	55
SQL	40	37	7	110
Visual Basic	47	42	16	158

Representative values developed by QSM

Why Opt for FP?

- Programming language independent
- Used readily countable characteristics that are determined early in the software process
- Does not "penalize" inventive (short) implementations that use fewer LOC that other more clumsy versions
- Makes it easier to measure the impact of reusable components

Object-Oriented Metrics

- Number of scenario scripts (use-cases)
- Number of support classes (required to implement the system but are not immediately related to the problem domain)
- Average number of support classes per key class (analysis class)
- Number of subsystems (an aggregation of classes that support a function that is visible to the end-user of a system)

15

WebE Project Metrics

- Number of static Web pages (the end-user has no control over the content displayed on the page)
- Number of dynamic Web pages (end-user actions result in customized content displayed on the page)
- Number of internal page links (internal page links are pointers that provide a hyperlink to some other Web page within the WebApp)
- Number of persistent data objects
- Number of external systems interfaced
- Number of static content objects
- Number of dynamic content objects
- Number of executable functions

16

Measuring Quality

- Correctness — the degree to which a program operates according to specification
- Maintainability—the degree to which a program is amenable to change
- Integrity—the degree to which a program is impervious to outside attack
- Usability—the degree to which a program is easy to use

17

Defect Removal Efficiency

$$DRE = E / (E + D)$$

E is the number of errors found before delivery of the software to the end-user

D is the number of defects found after delivery.

18

Metrics for Small Organizations

- time (hours or days) elapsed from the time a request is made until evaluation is complete, t_{queue}.
- effort (person-hours) to perform the evaluation, W_{eval}.
- time (hours or days) elapsed from completion of evaluation to assignment of change order to personnel, t_{eval}.
- effort (person-hours) required to make the change, W_{change}.
- time required (hours or days) to make the change, t_{change}.
- errors uncovered during work to make change, E_{change}.
- defects uncovered after change is released to the customer base, D_{change}.

Establishing a Metrics Program

- Identify your business goals.
- Identify what you want to know or learn.
- Identify your subgoals.
- Identify the entities and attributes related to your subgoals.
- Formalize your measurement goals.
- Identify quantifiable questions and the related indicators that you will use to help you achieve your measurement goals.
- Identify the data elements that you will collect to construct the indicators that help answer your questions.
- Define the measures to be used, and make these definitions operational.
- Identify the actions that you will take to implement the measures.
- Prepare a plan for implementing the measures.