Chapter 24

Project Management Concepts

Software Engineering: A Practitioner's Approach, 7/e
by Roger S. Pressman

The Four 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

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

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 resulting program 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 sociality (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 [Con'93]

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 Constantin’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 manuals, project milestones, software, and project control tools [Chapter 23]), change requests and project documentation, user group meetings, and project data.

- Informal, interpersonal procedures include group meetings for information dissemination and problem solving and “back-of-the-postenroom” reports, and project data.

- Electronic communication encompasses electronic mail, electronic bulletin boards, and videoconferencing systems. Informal networking includes informal discussions with team members and those outside the project who may have experience or insight that can assist team members.

The Product Scope

- **Scope**
  - Context: How does the software be built 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 6) 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.

Problem Decomposition

- Sometimes called **partitioning or problem elaboration**
- Once a 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

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 unrealistically set,
  - 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

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 (Boehm96)

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
Chapter 25

Process and Project Metrics

A Good Manager Measures

Why Do We Measure?

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 compliance
  - other measures.
- We also derive process metrics by measuring the characteristics of specific software engineering tasks.

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

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 approaches 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
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.

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

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

Comparing LOC and FP

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>LOC per Function Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>180</td>
</tr>
<tr>
<td>C++</td>
<td>160</td>
</tr>
<tr>
<td>C</td>
<td>80</td>
</tr>
<tr>
<td>C#</td>
<td>40</td>
</tr>
<tr>
<td>JavaScript</td>
<td>20</td>
</tr>
<tr>
<td>Perl</td>
<td>18</td>
</tr>
<tr>
<td>PL/I</td>
<td>8</td>
</tr>
<tr>
<td>Pascal</td>
<td>6</td>
</tr>
<tr>
<td>Fortran</td>
<td>5</td>
</tr>
<tr>
<td>COBOL</td>
<td>4</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>2</td>
</tr>
<tr>
<td>Fortran</td>
<td>1</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>0.5</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>0.2</td>
</tr>
</tbody>
</table>
| Representative values developed by GQM

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)

WebApp 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 interfaaced
- Number of static content objects
- Number of dynamic content objects
- Number of executable functions

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

Defect Removal Efficiency

\[ DRE = \frac{E}{E + D} \]

where:
- \( E \) is the number of errors found before delivery of the software to the end-user
- \( D \) is the number of defects found after delivery.
Metrics for Small Organizations

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

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.
- Formulate 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.

Software Project Planning

The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.

Why?
So the end result gets done on time, with quality!

Project Planning Task Set-I

- Establish project scope
- Determine feasibility
- Analyze risks
  - Risk analysis is considered in detail in Chapter 25.
- Define required resources
  - Determine require human resources
  - Define reusable software resources
  - Identify environmental resources

Project Planning Task Set-II

- Estimate cost and effort
  - Decompose the problem
  - Develop two or more estimates using size, function points, process tasks or use-cases
  - Reconcile the estimates
- Develop a project schedule
  - Scheduling is considered in detail in Chapter 27.
  - Establish a meaningful task set
  - Define a task network
  - Use scheduling tools to develop a timeline chart
  - Define schedule tracking mechanisms

Estimation

- Estimation of resources, cost, and schedule for a software engineering effort requires
  - Experience
  - Access to good historical information (metrics)
  - The courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty

Write it Down!

To Understand Scope ...

- Understand the customers needs
- Understand the business context
- Understand the project boundaries
- Understand the customer’s motivation
- Understand the likely paths for change
- Understand that ...

Even when you understand, nothing is guaranteed!
What is Scope?

- **Software scope** describes:
  - the functions and features that are to be delivered to end-users
  - the data that are input and output
  - the "content" that is presented to users as a consequence of using the software
  - the performance, constraints, interfaces, and reliability that bound the system.

- Scope is defined using one of two techniques:
  - A narrative description of software scope is developed after communication with all stakeholders.
  - A set of use-cases is developed by end-users.

Resources

- **Past (similar) project experience**
- **Conventional estimation techniques**
  - task breakdown and effort estimates
  - size (e.g., FP) estimates
- **Empirical models**
- **Automated tools**

Estimation Techniques

- **Predicated on**
  - the degree to which the planner has properly estimated the size of the product to be built
  - the ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects)
  - the degree to which the project plan reflects the abilities of the software team
  - the stability of product requirements and the setting that supports the software engineering effort.

Estimation Accuracy

- **Functional Decomposition**
  - Perform a Grammatical "parse"

Project Estimation

- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process.

Conventional Methods: LOC/FP Approach

- compute LOC/FP using estimates of information domain values
- use historical data to build estimates for the project

Example: LOC Approach

<table>
<thead>
<tr>
<th>Function</th>
<th>Source Code Effort</th>
<th>LOC (Lines of Code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>5</td>
<td>15,000</td>
</tr>
<tr>
<td>function</td>
<td>7</td>
<td>20,000</td>
</tr>
<tr>
<td>function</td>
<td>9</td>
<td>25,000</td>
</tr>
</tbody>
</table>

Average productivity for systems of this type = 620 LOC/pm,
Bureaucrat labor rate = $6000 per month, the cost per line of code is approximately $10.
Based on the LOC estimate and the historical productivity, the total-estimated project cost is $437,000 and the estimated effort is 54 person-months.

Example: FP Approach

The estimated number of FP is derived:

$$ F_P_{estimated} = \sum F_P + 0.65 \times 0.15 \times F_P $$

organizational average productivity = 8.5 FP/pm, function labor rate = $6000 per month, approximately $1200/FP.
Based on the FP estimate and the historical productivity data, total estimated project cost is $461,000 and estimated effort is 56 person-months.
**Process-Based Estimation**

Obtained from "process framework"

- framework activities
- application functions

- Effort required to accomplish each framework activity for each application function

**Process-Based Estimation Example**

<table>
<thead>
<tr>
<th>Project</th>
<th>LOC</th>
<th>Effort</th>
<th>LOC/FP</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100</td>
<td>5</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>B</td>
<td>200</td>
<td>10</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>C</td>
<td>300</td>
<td>15</td>
<td>0.5</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Based on an average burdened labor rate of $8,000 per month, the total estimated project cost is $552,000 and the estimated effort is 46 person-months.

**Tool-Based Estimation**

- project characteristics
- calibration factors
- LOC/FP data

**Estimation with Use-Cases**

Using 620 LOCpm as the average productivity for systems of this type and a burdened labor rate of $8,000 per month, the cost per line of code is approximately $10. Based on the use-case estimate and the historical productivity data, the total estimated project cost is $552,000 and the estimated effort is 46 person-months.

**Empirical Estimation Models**

**COCOMO-II**

- COCOMO II is actually a hierarchy of estimation models that address the following areas:
  - Application composition model: Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
  - Early design stage model: Used once requirements have been stabilized and basic software architecture has been established.
  - Post-architecture-stage model: Used during the construction of the software.

**The Software Equation**

A dynamic multivariable model

\[ E = [LOC \times B^{3.32/PP}] \times (1/t) \]

where

- \( E \) = effort in person-months or person-years
- \( t \) = project duration in months or years
- \( B \) = "special skills factor"
- \( P \) = "productivity parameter"

**Estimation for OO Projects-I**

- Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
- Using object-oriented requirements modeling (Chapter 6), develop use-cases and determine a count.
- From the analysis model, determine the number of key classes (called analysis classes in Chapter 6).
- Categorize the type of interface for the application and develop a multiplier for support classes:
  - Interface type
    - No GUI: 2.0
    - Text-based user interface: 2.25
    - GUI: 2.5
    - Complex GUI: 3.0

**Estimation for OO Projects-II**

- Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.
- Multiply the total number of classes (key + support) by the average number of work-units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
- Cross check the data-based estimate by multiplying the average number of work-units per use-case.
Estimation for Agile Projects

- Each user scenario (a mini-use-case) is considered separately for estimation purposes.
- The scenario is decomposed into the set of software engineering tasks that will be required to develop it.
- Each task is estimated separately. Note: estimation can be based on historical data, an empirical model, or “experience.”
- Alternatively, the “volume” of the scenario can be estimated in LOC, FC, or some other volume-oriented measure (e.g., use case counts).
- Estimates for each task are summed to create an estimate for the scenario.
- Alternatively, the volume estimate for the scenario is translated into effort using historical data.
- The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

Chapter 27

Project Scheduling

Software Engineering: A Practitioner’s Approach, 7th ed.
by Roger R. Pressman

Why Are Projects Late?

- Unrealistic deadlines established by someone outside the software development group
- Changing customer requirements that are not reflected in schedule changes
- An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job
- Predictable and/or unpredictable risks that were not considered when the project commenced
- Technical difficulties that could not have been foreseen in advance
- Human difficulties that could not have been foreseen in advance
- Miscommunication among project staff that results in delays
- Failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem

Effort and Delivery Time

- Front end activities: customer communication, analysis, design
- Construction activities: coding, generation, testing and installation, unit, integration, white box, black box regressions

Effort Allocation

- 40-50%: front end activities
- 15-20%: construction activities
- 30-40%: testing and installation

Computing Expected Cost

\[
\text{expected cost} = \sum \text{(path probability)} \times \text{(estimated path cost)}
\]

For example, the expected cost to build is:
- \(0.30 \times 3360K + 0.70 \times 5450K\)
- \(= 529K\)
- Reuse
- \(= 312K\)
- Buy
- \(= 841K\)

Scheduling Principles

- compartmentalization—define distinct tasks
- interdependency—indicate task interrelationship
- effort validation—be sure resources are available
- defined responsibilities—people must be assigned
- defined outcomes—each task must have an output
- defined milestones—review for quality

Defining Task Sets

- determine type of project
- assess the degree of rigor required
- identify adaptation criteria
- select appropriate software engineering tasks
Task Set Refinement

1.1 Concept scoping determines the overall scope of the project.

- Concept scoping: Identifies the overall goals and objectives of the project.
- Concept scoping: Identifies the stakeholders and their needs.
- Concept scoping: Identifies the project scope and project boundaries.

Define a Task Network

- Define a Task Network: Identifies the tasks and activities required to complete the project.
- Define a Task Network: Establishes the dependencies between tasks.
- Define a Task Network: Determining the sequence and timing of tasks.

Timeline Charts

- Timeline Charts: Visual representation of the project timeline.
- Timeline Charts: Helps in tracking the progress of the project.
- Timeline Charts: Facilitates in identifying any delays or bottlenecks.

Use Automated Tools to Derive a Timeline Chart

- Use Automated Tools to Derive a Timeline Chart: Helps in improving the accuracy of timelines.
- Use Automated Tools to Derive a Timeline Chart: Reduces the time and effort in creating timelines.

Schedule Tracking

- Schedule Tracking: Conducts periodic project status meetings.
- Schedule Tracking: Evaluates the results of all reviews.
- Schedule Tracking: Determines whether formal project milestones have been accomplished.
- Schedule Tracking: Compares actual start-date to planned start-date for each project task listed in the resource table.
- Schedule Tracking: Seeks informal feedback from practitioners to obtain their subjective assessment of progress to date and problems on the horizon.
- Schedule Tracking: Uses earned value analysis (Section 27.6) to assess progress quantitatively.

Progress on an OO Project-I

- Technical milestone: OO analysis completed
  - All classes and the data hierarchy have been defined and reviewed.
  - Class attributes and operations associated with a class have been defined and reviewed.
  - Class interface (Chapter 6) has been completed and reviewed.
- Technical milestone: OO design completed
  - The set of subsystems (Chapter 8) has been defined and reviewed.
  - Classes are allocated to subsystems and reviewed.
  - Task Allocation has been established and reviewed.
  - Responsibilities and collaborations (Chapter 8) have been identified.
  - All classes and operations have been designed and reviewed.
- Technical milestone: OO design completed
  - The communication model has been created and reviewed.

Progress on an OO Project-II

- Technical milestone: OO coding completed
  - Each new class has been implemented in code from the design model.
  - Extracted classes (from a reuse library) have been implemented.
  - Prototype or increment has been built.

Technical milestone: OO testing

- The correctness and completeness of OO analysis and design models have been reviewed.
- A class responsibility / behavior network (Chapter 6) has been developed and reviewed.
- Test cases are designed and class-level tests (Chapter 10) have been conducted for each class.
- Test cases are designed and cluster testing (Chapter 10) is completed and the classes are integrated.
- System level tests have been completed.

Earned Value Analysis (EVA)

- Earned value
  - A measure of progress
  - Enables us to assess the "percent of completeness" of a project using quantitative analysis rather than rely on a gut feeling
  - Provides accurate and reliable readings of performance from as early as 15 percent into the project.

Computing Earned Value

- The budgeted cost of work scheduled (BCWS) is determined for each work task represented in the schedule.
- BCWS is the effort planned for work task i.
  - To determine progress at a given point in time on the project schedule, the value of BCWS is the sum of the BCWS values for all work tasks that should have been completed by that point in time on the project schedule.
- The BCWS values for all work tasks are summed to derive the budget at completion, BAC. Hence,

\[ BAC = \sum BCWS_i \] for all tasks i
Computing Earned Value-II

- Next, the value for budgeted cost of work performed (BCWP) is computed.
- The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.
- "The distinction between the BCWS and the BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed." (NIMB)
- Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:
  - Schedule performance index, SPI = BCWP/BCWS
  - Schedule variance, SV = BCWP - BAC
  - SPI is an indication of the efficiency with which the project is utilizing scheduled resources.

Project Risks

- What can go wrong?
- What is the likelihood?
- What will the damage be?
- What can we do about it?

Reactive Risk Management

- Project team reacts to risks when they occur
- Mitigation—plan for additional resources in anticipation of fire fighting
- Fix on failure—resource are found and applied when the risk strikes
- Crisis management—failure does not respond to applied resources and project is in jeopardy

Proactive Risk Management

- Formal risk analysis is performed
- Organization corrects the root causes of risk
- TQM concepts and statistical SQA
- Examining risk sources that lie beyond the bounds of the software
- Developing the skill to manage change

Seven Principles

- Maintain a global perspective—view software risk within the context of system and the business problem
- Take a forward-looking view—think about the risks that may arise in the future, establish contingencies ahead
- Encourage open communication—if someone states a potential risk, don’t discount it
- Integrate—identification of risk must be integrated into the software process
- Emphasize a continuous process—risk management must be vigilant throughout the software process, modifying identified risks as more information is known and adding new ones as better insight is achieved
- Develop a shared product vision—k all stakeholders share the same vision of the software, it will get that better risk identification and assessment will occur
- Encourage teamwork—the talents, skills and knowledge of all stakeholders should be pooled

Risk Management Paradigm

- Risk management processes associated with the development of the software to be built or modified
- Business objectives associated with constraints imposed by management or the marketplace
- Customer characteristics of the software associated with the expectations of the customer and the developer’s ability to communicate with the customer in a timely manner
- Process definition—risks associated with the degree to which the software process has been defined and is followed by the development organization
- Staff skills associated with the availability and quality of the staff that will be used to build the product
- Technical—risk exposure associated with the unavailability of the system to be built and the “novelty” of the technology that is being managed by the system
- Staff site and safety risks associated with the overall technical and project experience of the software engineers who will do the work,

Chapter 28

- Risk Analysis
  - Software Engineering: A Practitioner’s Approach, 7th Edition by Roger S. Pressman
Assessing Project Risk-I

- Have top software and customer managers formally committed to support the project?
- Are end-users enthusiastically committed to the project and the system/product to be built?
- Are requirements fully understood by the software engineering team and their customers?
- Have customers been involved fully in the definition of requirements?
- Do end-users have realistic expectations?

Assessing Project Risk-II

- Is project scope stable?
- Does the software engineering team have the right mix of skills?
- Are project requirements stable?
- Does the project team have experience with the technology to be implemented?
- Is the number of people on the project team adequate to do the job?
- Do all customer/user constituencies agree on the importance of the project and the requirements for the system/product to be built?

Risk Components

- **Performance risk**—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
- **Cost risk**—the degree of uncertainty that the project budget will be maintained.
- **Support risk**—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
- **Schedule risk**—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

Risk Projection

- **Risk projection**, also called **risk estimation**, attempts to rate each risk in two ways:
  - the likelihood or probability that the risk is real
  - the consequences of the problems associated with the risk, should it occur.
- The are four risk projection steps:
  - establish a scale that reflects the perceived likelihood of a risk
  - delineate the consequences of the risk
  - estimate the impact of the risk on the project and the product
  - note the overall accuracy of the risk projection so that there will be no misunderstandings.

Building a Risk Table

<table>
<thead>
<tr>
<th>Risk</th>
<th>Probability</th>
<th>Impact</th>
<th>RMM/M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk Mitigation Monitoring &amp; Management</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Building the Risk Table

- Estimate the probability of occurrence
- Estimate the impact on the project on a scale of 1 to 5, where:
  - 1 = low impact on project success
  - 5 = catastrophic impact on project success
- Sort the table by probability and impact

Risk Exposure (Impact)

The overall **risk exposure**, RE, is determined using the following relationship [Ha1986]:

\[ RE = P \times C \]

where

- \( P \) is the probability of occurrence for a risk, and
- \( C \) is the cost to the project should the risk occur.

Risk Exposure Example

- **Risk Identification**: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.
- **Risk Probability**: 80% (likely).
- **Risk Impact**: 50 reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is $14.00, the overall cost (impact) to develop the components would be 18 x 100 x 14 = $25,200.
- **Risk Exposure**: \( RE = 0.80 \times 25,200 = $20,200.\)
Risk Due to Product Size

Attributes that affect risk:
- estimated size of the product in LOC or FP?
- estimated size of product in number of programs, files, transactions?
- percentage deviation in size of product from average for previous products?
- size of database created or used by the product?
- number of users of the product?
- number of projected changes to the requirements for the product before delivery? after delivery?
- amount of reused software?

Risk Due to Business Impact

Attributes that affect risk:
- affect of this product on company revenue?
- visibility of this product by senior management?
- reasonableness of delivery deadline?
- number of customers who will use this product?
- interoperability constraints?
- sophistication of end users?
- amount and quality of product documentation that must be produced and delivered to the customer?
- governmental constraints?
- costs associated with late delivery?
- costs associated with a defective product?

Risks Due to the Customer

Questions that must be answered:
- Have you worked with the customer in the past?
- Does the customer have a solid idea of requirements?
- Has the customer agreed to spend time with you?
- Is the customer willing to participate in reviews?
- Is the customer technically sophisticated?
- Is the customer willing to let your people do their job—that is, will the customer resist looking over your shoulder during technically detailed work?
- Does the customer understand the software engineering process?

Risks Due to Process Maturity

Questions that must be answered:
- Have you established a processes framework?
- Is it followed by project teams?
- Do you have management support for software engineering?
- Do you have a proactive approach to SQA?
- Do you conduct formal technical reviews?
- Are CASE tools used for analysis, design and testing?
- Are the tools integrated with one another?
- Have document formats been established?

Technology Risks

Questions that must be answered:
- Is the technology new to your organization?
- Are new algorithms, I/O technology required?
- Is new or unproven hardware involved?
- Does the application interface with new software?
- Is a specialized user interface required?
- Is the application radically different?
- Are you using new software engineering methods?
- Are you using unconventional software development methods, such as formal methods, AI-based approaches, artificial neural networks?
- Are there significant performance constraints?
- Is there doubt the functionality requested is "do-able?"

Staff/People Risks

Questions that must be answered:
- Are the best people available?
- Does staff have the right skills?
- Are enough people available?
- Are staff committed for entire duration?
- Will some people work part time?
- Do staff have the right expectations?
- Have staff received necessary training?
- Will turnover among staff be low?

Recording Risk Information

Project: Embedded software for XYZ system
Risk type: schedule risk
Priority: 1 low, 2 critical: 4
Risk factor: Project completion will depend on tasks which require software component under development. Hardware component development is not expected to delay the project大大大.
Probability: 60%
Impact: Project completion will be delayed by up to 8 weeks. The project will not be feasible. The project will be delayed by up to 4 weeks.
Mitigation approach: Reassign resources to hardware development.
Contingency plan: Reassign resources to hardware development. Additional person will be added to team to assist in this effort.
Estimated resources: 2 additional person months beginning in July.