UNIT - V

# Software Release

Prepared by

Dr. Banu Priya P, Dept of NWC
SRMIST, KTR

**Disclaimer:**
**The lecture notes have been prepared by referring to many books and notes prepared by the teachers. This document does not claim any originality and cannot be used as a substitute for prescribed textbooks.**

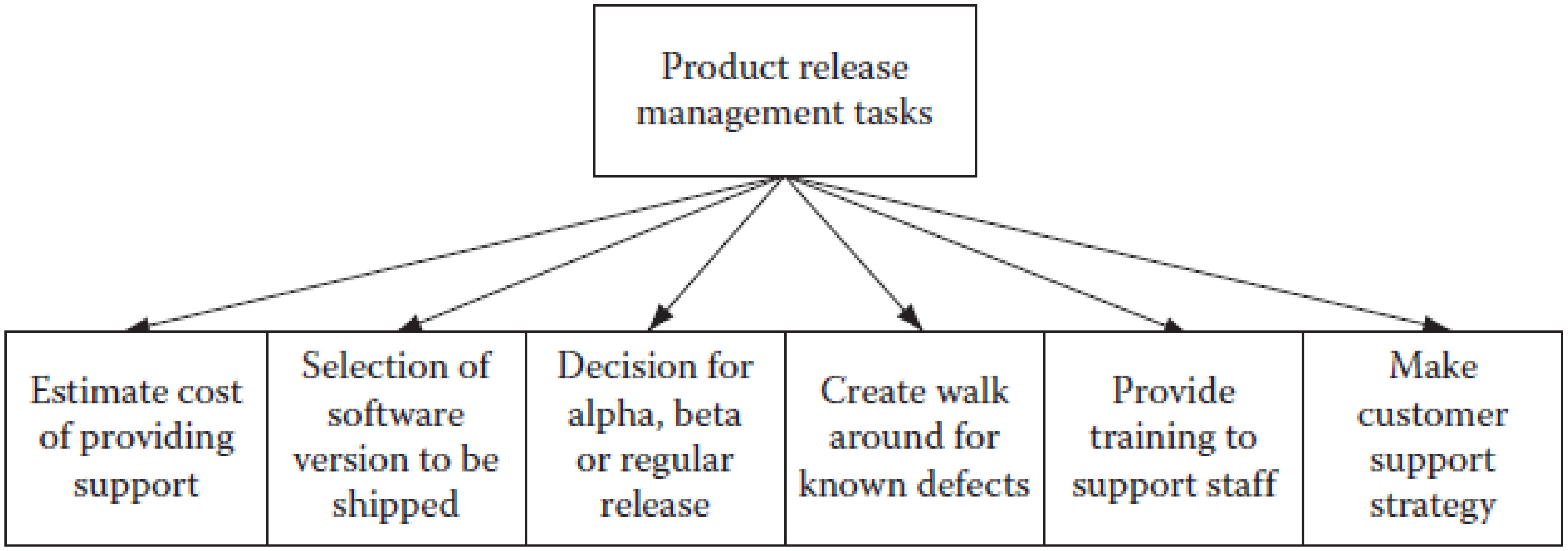
## Topics

- Product Release Management
- Product Implementation
- Risk management
- Reactive Vs proactive risk strategies
- Software risks
- Risk identification
- Risk projection
- Risk refinement
- RMMM - RMMM plan
- Maintenance and Reengineering

# Product Release

- The software product which has been built till now is now complete.
- It needs to be taken to the customer’s site and get it implemented so that the end users can start using it.
- However, do not run fast in anticipation of wrapping things as early as possible.
- After all this is the magnum opus and needs to be carefully handled.
- It is necessary to make sure that all the tasks are completed.
- For example, product support cost estimate, walk arounds for known bugs, which version of the software product to be released, if release should be an alpha, beta, or normal release, training needs fulfilled, and customer support strategy

# Product Release


Product release management tasks
Estimate cost of providing support
Selection of software version to be shipped
Decision for alpha, beta or regular release
Create walk around for known defects
Provide training to support staff
Make customer support strategy

## Product Release Management

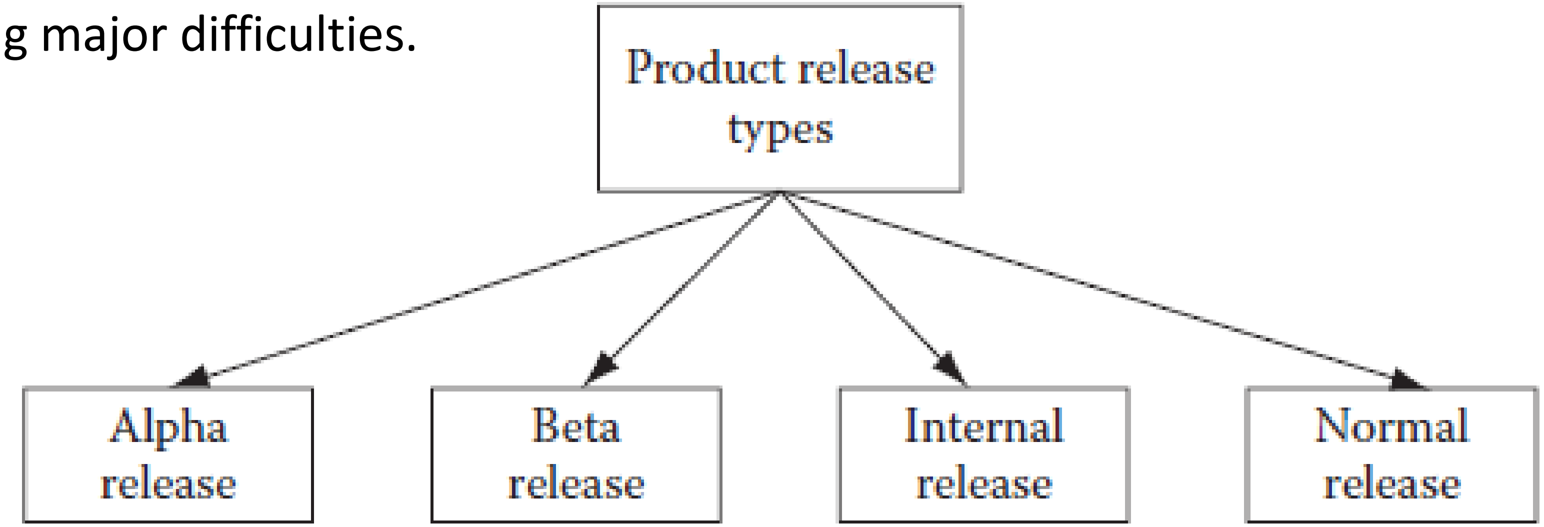
- Project teams working for software product vendors struggle to keep pace with release of the software product.
- There is pressure from the market to launch new versions by certain dates.
- New features are to be added, porting the product to new platforms, old features are to be enhanced, existing bugs are to be removed, and yet it has to meet the deadline.
- It is a constant struggle that calls for good product release strategies.
- Depending on the situation, the project manager may need to convince the management to cut short some of the product features to meet the deadline as well as meet quality standards.
- A half-baked product will never have any takers; instead, the project manager may be blamed for its poor-quality issues.

## Product Release Management

- Bargaining also must be done for other requirements of bug fixes, feature enhancements, etc.
- If quality concerns are paramount, then moving some of the tasks of new features to a future release may be the best solution for meeting quality standards.
- If the software vendor is not too sure about product quality, then he may opt for an alpha or beta release of the product.
- In that case, the product will be released only among a few selected groups and not in the market.
- The controlled product release is the best option in these conditions
- In fact, product release management is such a dynamic environment that if proper planning is not done at a minute level and constant vigilance is not applied over project activities, then a huge mess can be created and there will be no time to clear it.

# Product Release Management

- Finally, for the product's scheduled release, how the customer support will be provided should be chalked out.
- Walk arounds for known issues, estimated number of critical bugs remaining in the product, training for the support staff, etc., should be done.
- The cost of support, depending on the number of estimated users, walk arounds, and remaining bugs should be figured out.
- These measures will ensure that the product is transitioned into market without facing major difficulties.



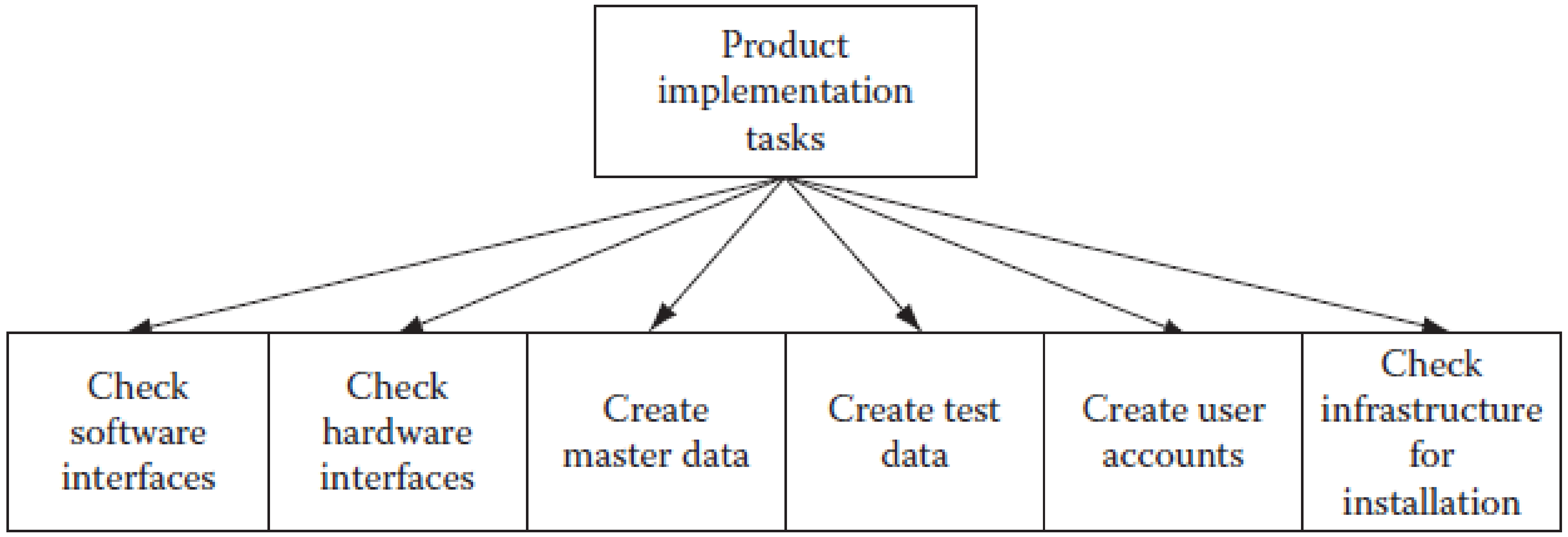
## Product Implementation

- The product that has been developed and thoroughly tested now it needs to be implemented at a customer site.
- It is necessary to prepare all master data and test transaction data for testing the implemented product.
- In addition, need to get all required hardware and software that needs to be there for installing the software product.
- Furthermore, need to make sure that development and testing is done for all the hardware and software interfaces for integrating the product, with existing legacy systems and infrastructure.

# Product Implementation

- Moreover, need to make sure that the product will run smoothly on customer premises without any interference with their existing applications (Figure above – Task list for software product implementation).
- Often project teams run into problems during implementation, due to unforeseen circumstances or negligence on part of the production team or customer’s team.
- Therefore, it is recommended to prepare a list of developer requirements and hand it over to customer’s support team so that they are prepared when developers arrive for implementation.

# Product Implementation


Product implementation tasks
Check software interfaces
Check hardware interfaces
Create master data
Create test data
Create user accounts
Check infrastructure for installation

## User Training

- Make sure that the user manual prepared by the team is up to date and in synch with the version of the software product, which will be implemented at the customer site.
- It is not possible to provide training to all users.
- So, it is better to prepare a list of roles that are needed to operate the product.
- This list needs to be given to the end users and ask them to select one user per role who will receive the training.
- Apart from the user manual, it is necessary to prepare a tutorial to include probable scenarios that may arise during operation of the product.

## User Training

- The tutorial will provide a step-by-step guide for using the product under those scenarios.
- This will be a very important step in training, because if users do not learn it during training, then they may contact later after implementation and ask for information as to how to use the product in those circumstances.
- This will lead to a waste of the support team’s time.
- It is lot better to train them now, during user training, rather than face user requests later.

# Risk Management Process

- Risk Management is the process of identifying, assessing, and controlling risks that could potentially affect a software project’s objectives such as cost, schedule, scope, or quality.
- A **risk** is a potential event or condition that may negatively (or positively) impact a project. In SEPM, most risks are related to:
  - Technical feasibility
  - Requirements uncertainty
  - Human resources
  - Budget constraints
  - Scheduling issues
  - External factors (regulatory, client-side changes)
- There are two primary risk management strategies: **Reactive and Proactive**

## Risk Management Process : Steps

**1. Risk Identification:**

- Detect potential risks early.
- Tools: Brainstorming, checklists, interviews, historical data.
- Examples: Delays in delivery, scope creep, technology failures.

**2. Risk Analysis:**

- Assess probability and impact.
- Qualitative: High/Medium/Low.
- Quantitative: Use of metrics, simulations.

**3. Risk Prioritization:**

- Use Risk Exposure: Risk Exposure = Probability × Impact
- Prioritize high-risk areas.

# Risk Management Process : Steps

**4. Risk Mitigation Planning:** Develop strategies to reduce the likelihood or impact.

- **Types of strategies:**
  - Avoidance: Change plans to sidestep the risk.
  - Mitigation: Take actions to reduce risk.
  - Transfer: Shift the risk to another party (e.g., insurance).
  - Acceptance: Acknowledge and plan for it if it occurs.
- **Risk Monitoring and Control:**
  - Track identified risks.
  - Identify new risks during the project.
  - Update mitigation plans as needed.
- **Risk Documentation:** Maintain a Risk Register to log all risks, their status, mitigation plans, and updates.

## Reactive Risk Management

- A strategy where action is taken after a risk has materialized (i.e., when a problem occurs).
- **Characteristics:**
  - No formal risk analysis or planning in advance.
  - "Fix-it-when-it-breaks" approach.
  - Relies on contingency plans developed on the fly.
  - Usually driven by crisis management.
- **Examples:**
  - Bug is discovered during deployment → Fix and patch urgently
  - Key team member leaves suddenly → Reassign tasks under pressure.

## Reactive Risk Management

- **Pros:**
  - Less time spent upfront in planning.
  - May be sufficient for small, low-risk projects.
- **Cons:**
  - Often leads to higher costs.
  - Can delay the project significantly.
  - Stressful for the team.
  - May damage stakeholder confidence.

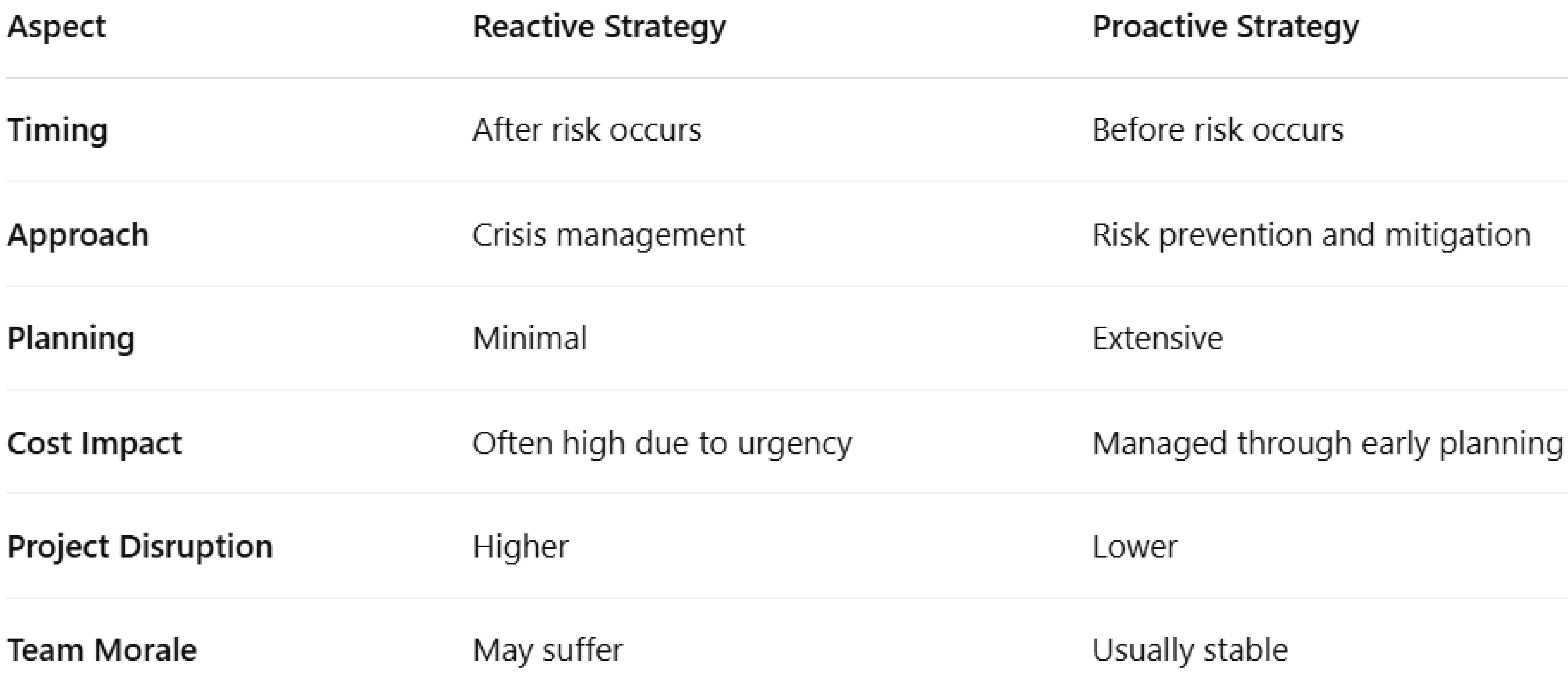
## Proactive Risk Management

- A strategy where risks are identified, assessed, and mitigation plans are made before they occur.
- **Characteristics:**
  - Involves risk identification, analysis, prioritization, and planning.
  - Uses risk logs, assessments, and management plans.
  - Focus on minimizing impact or likelihood of risks.
- **Examples:**
  - Identifying possible technology risks early and planning a backup tech stack.
  - Planning for buffer time if requirements are unclear.
  - Training backup personnel in case of attrition.

## Proactive Risk Management

- **Pros:**
  - Reduces surprises and crises.
  - Allows better resource and budget planning.
  - Enhances stakeholder trust.
  - Leads to higher project predictability and quality.
- **Cons:**
  - Takes more upfront effort.
  - May seem unnecessary if risks do not materialize.
  - Requires experienced risk analysts.

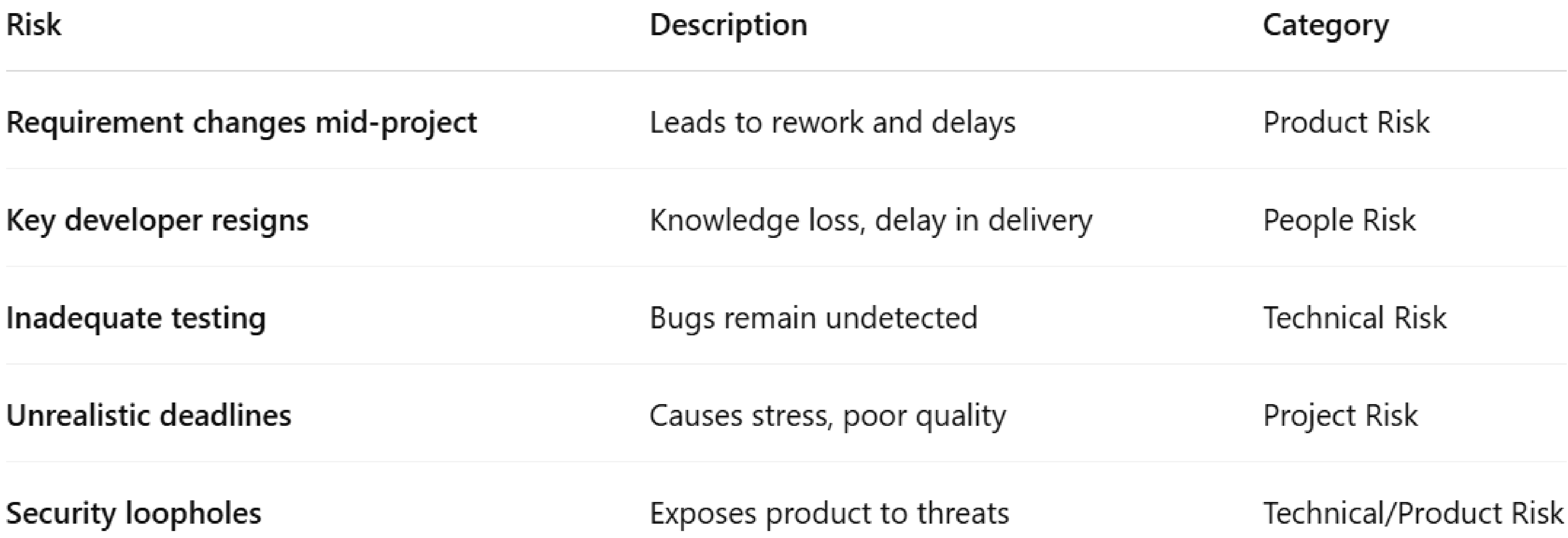
# Comparison



| Aspect | Reactive Strategy | Proactive Strategy |
|---|---|---|
| **Timing** | After risk occurs | Before risk occurs |
| **Approach** | Crisis management | Risk prevention and mitigation |
| **Planning** | Minimal | Extensive |
| **Cost Impact** | Often high due to urgency | Managed through early planning |
| **Project Disruption** | Higher | Lower |
| **Team Morale** | May suffer | Usually stable |

# Software Risks

- **Software risks** refer to potential issues or uncertain events that could negatively affect the success of a software project—impacting timelines, quality, budget, or overall delivery.



| Risk | Description | Category |
|---|---|---|
| **Requirement changes mid-project** | Leads to rework and delays | Product Risk |
| **Key developer resigns** | Knowledge loss, delay in delivery | People Risk |
| **Inadequate testing** | Bugs remain undetected | Technical Risk |
| **Unrealistic deadlines** | Causes stress, poor quality | Project Risk |
| **Security loopholes** | Exposes product to threats | Technical/Product Risk |

## Classification of Software Risks

1. **Project Risks:** Risks that affect the project plan and management process.
   - Poor project planning or unrealistic deadlines
   - Inadequate resource allocation
   - Scope creep (uncontrolled changes in project scope)Budget overruns
   - Miscommunication among stakeholders
2. **Technical Risks:** Risks related to technology, tools, or technical constraints.
   - Use of new or unproven technologies
   - Integration challenges with legacy systems
   - Performance bottlenecks
   - Security vulnerabilities
   - Inadequate testing

# Classification of Software Risks

**3. Product Risks:** Risks that impact the functionality and quality of the final software product.

- Misunderstood or changing requirements
- Incomplete or incorrect specifications
- Low usability
- Failure to meet customer expectations
- Quality issues (bugs, crashes, etc.)

**4. People Risks:** Risks associated with the human factor in the project.

- Lack of skilled developers or key personnel
- High team turnover
- Poor communication or team conflicts
- Inadequate training or motivation

## Classification of Software Risks

**5. Process Risks:** Risks arising from flaws in the software development process.

- Inadequate process definition
- Poor version control or configuration management
- Lack of adherence to software engineering standards
- Improper documentation

**6. Organizational and External Risks:** Risks related to the company environment or external influences.

- Changes in organizational priorities
- Market or economic fluctuations
- Regulatory and compliance issues
- Stakeholder conflicts or changing business needs

## Risk Identification

- The process of recognizing potential risks that might negatively impact the software project.
- **Goal:** To create a list of known or anticipated risks across all project areas—technical, business, people, and process.
- **Techniques:**
  - Brainstorming sessions with stakeholders or experts
  - SWOT Analysis (Strengths, Weaknesses, Opportunities, Threats)
  - Checklists from past projects
  - Interviews with experienced developers/project managers
  - Risk Taxonomy (grouping risks into categories)

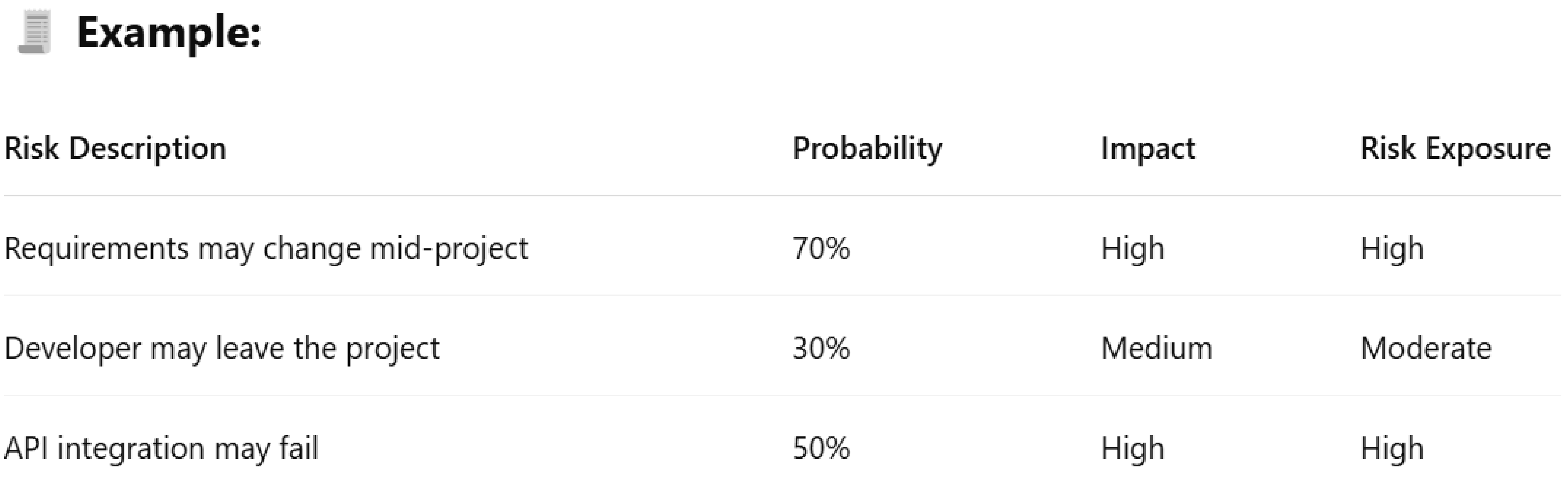
# Risk Identification

- **Examples of Identified Risks:**
  - Requirements may change frequently.
  - A critical developer may leave the team.
  - Integration with a third-party API may fail.
  - The project may exceed the budget.

## Risk Projection

- The process of evaluating each identified risk to estimate:
  - Probability of occurrence (Low, Medium, High or %)
  - Impact on project objectives (Schedule, Cost, Scope, Quality)
  - Exposure (Risk Exposure = Probability × Impact)
- **Goal:** To prioritize risks and focus on the ones that matter most.
- **Tools & Techniques:**
  - Risk Matrix (Probability vs. Impact grid)
  - Risk Exposure Calculation
  - Delphi Technique (expert consensus)
  - Monte Carlo Simulation (for complex risk analysis)

# Risk Projection



🧾 **Example:**

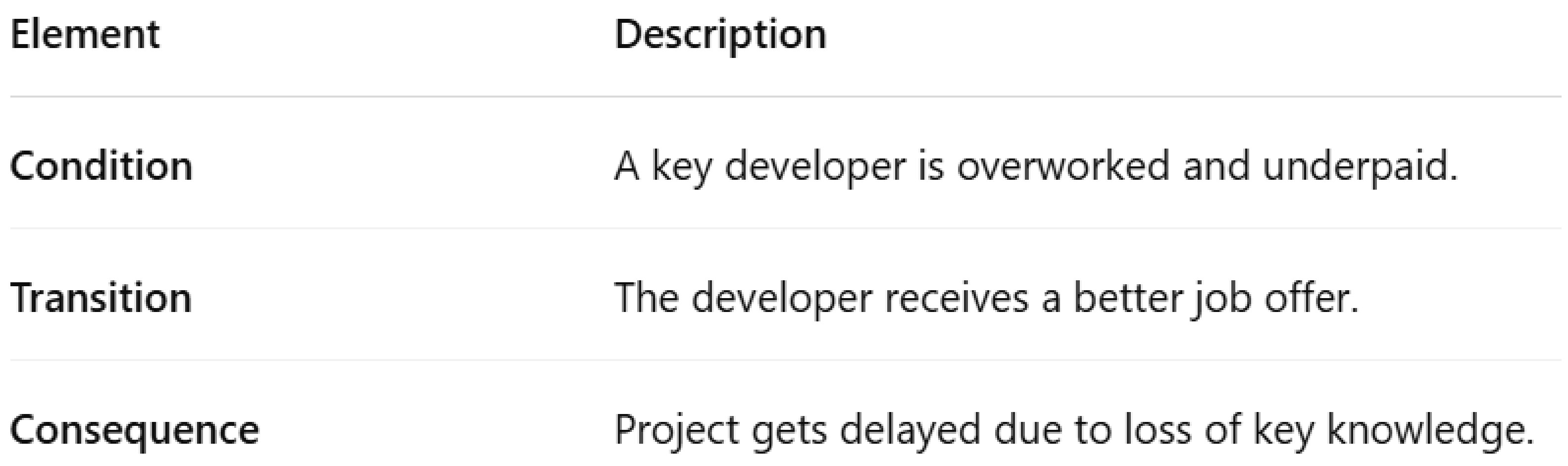
| Risk Description | Probability | Impact | Risk Exposure |
|---|---|---|---|
| Requirements may change mid-project | 70% | High | High |
| Developer may leave the project | 30% | Medium | Moderate |
| API integration may fail | 50% | High | High |

## Risk Refinement

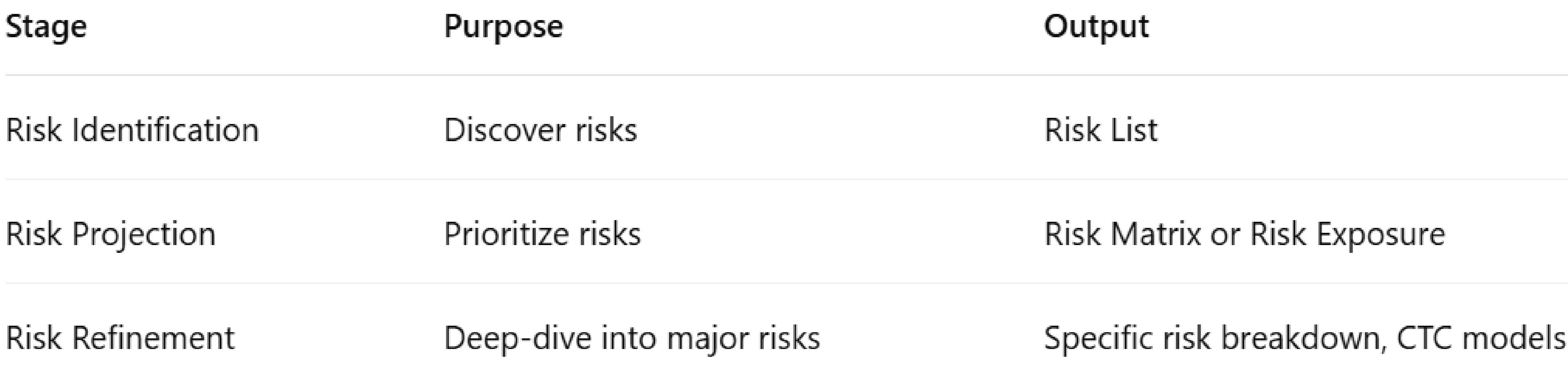
- The process of analyzing and breaking down high-priority risks into more specific sub-risks and their causes/effects to better understand and manage them.
- **Goal:** To gain deeper insight into the nature, sources, and interactions of risks—especially the most serious ones.
- **Method:** Use of Causal Analysis or Root Cause Analysis
- **Condition-Transition-Consequences (CTC) Model:**
  – Condition: Circumstance under which the risk may occur
  – Transition: Trigger event or action
  – Consequence: Impact on the project

# Risk Refinement

- Example Using CTC Model: Risk: “Team member may leave the project”



| Element | Description |
|---|---|
| **Condition** | A key developer is overworked and underpaid. |
| **Transition** | The developer receives a better job offer. |
| **Consequence** | Project gets delayed due to loss of key knowledge. |



| Stage | Purpose | Output |
|---|---|---|
| Risk Identification | Discover risks | Risk List |
| Risk Projection | Prioritize risks | Risk Matrix or Risk Exposure |
| Risk Refinement | Deep-dive into major risks | Specific risk breakdown, CTC models |

## RMMM

- **Risk Mitigation, Monitoring, and Management** — a structured plan in Software Engineering Project Management to handle risks effectively throughout the project life cycle.
- The RMMM Plans the strategies and actions that will be taken to:
  - Mitigate risks before they happen,
  - Monitor risks during the project,
  - Manage or respond to risks if they actually occur.
- **Components of RMMM:**
  - Risk Mitigation
  - Risk Monitoring
  - Risk Management (Contingency Planning)

## Risk Mitigation

- **Goal:** Reduce the likelihood or impact of a risk before it occurs.

**Mitigation Strategies:**

- Proactive planning: Define backup plans
- Training: Upskill the team to reduce people risks
- Prototype development: Minimize tech feasibility risks
- Using proven technology: Avoid tech failure
- Stakeholder communication: Control scope creep

**Example:**

- Risk: New technology may not integrate well.
- Mitigation: Build a prototype early to test integration.

## Risk Monitoring

- **Goal:** Track risks continuously throughout the project life cycle and detect new risks early.

**Monitoring Actions:**

- Regular status reports
- Use of risk checklists during reviews
- Frequent team check-ins
- Monitoring trigger events (conditions that indicate a risk may occur)

**Example:**

- Risk: Developer may leave.
- Monitoring: Watch for signs of disengagement or job dissatisfaction.

## Risk Management (Contingency Planning)

- **Goal:** Have a contingency plan or backup action ready if the risk materializes.

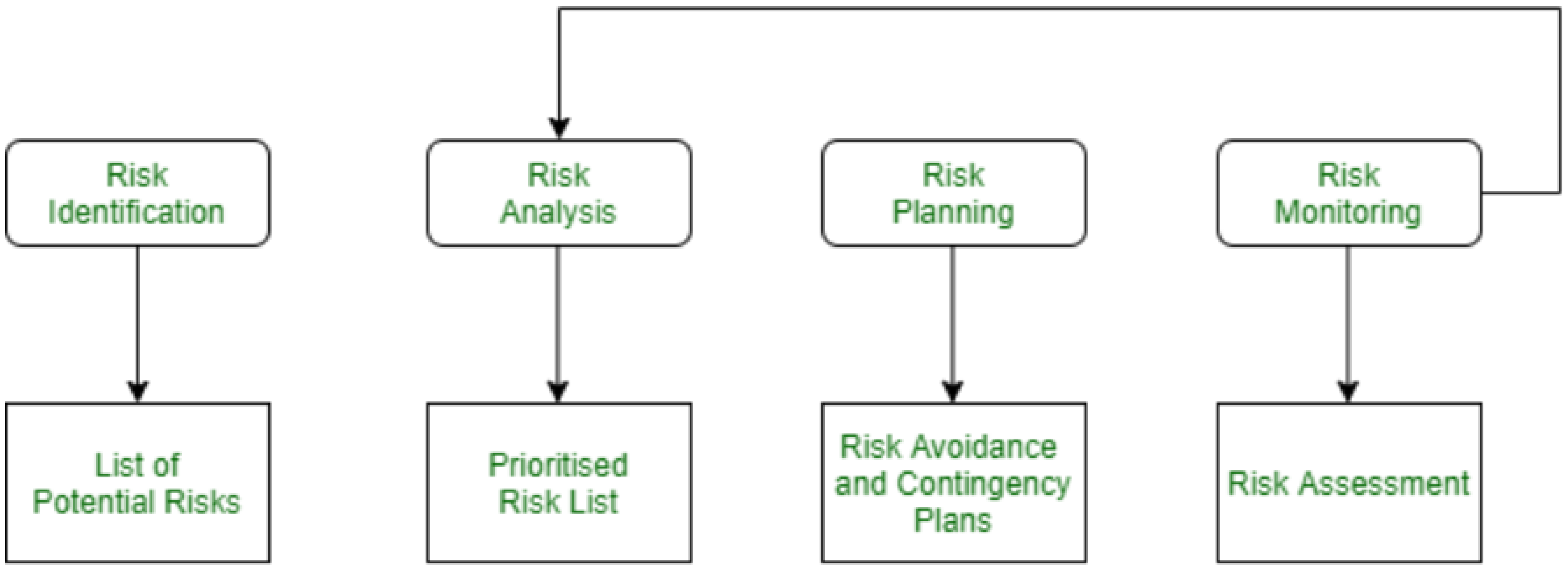
**Management Actions:**

- Plan B execution
- Reallocation of resources
- Schedule or scope adjustments
- Activate external support or vendors

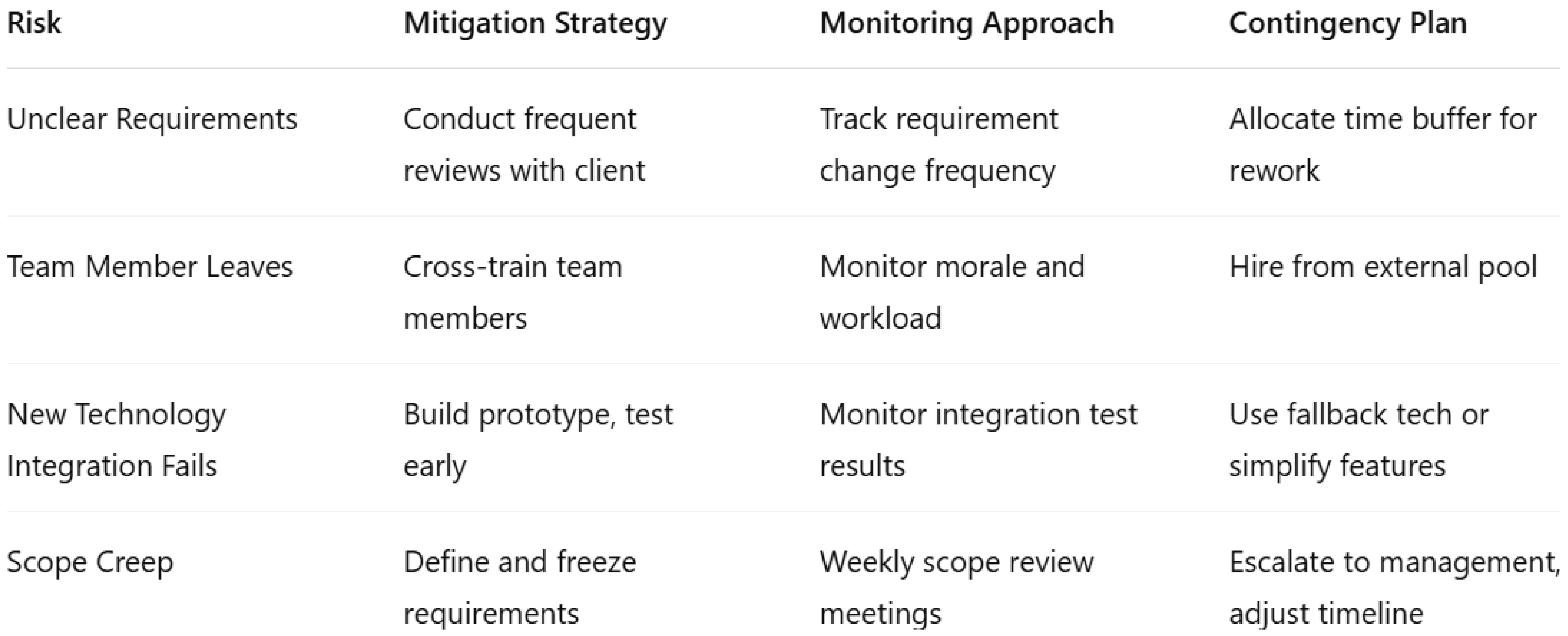
**Example:**

- Risk: Key developer resigns.
- Management: Onboard a trained backup developer from the bench.

## Risk Management Process



# Sample RMM Table



| Risk | Mitigation Strategy | Monitoring Approach | Contingency Plan |
|---|---|---|---|
| Unclear Requirements | Conduct frequent reviews with client | Track requirement change frequency | Allocate time buffer for rework |
| Team Member Leaves | Cross-train team members | Monitor morale and workload | Hire from external pool |
| New Technology Integration Fails | Build prototype, test early | Monitor integration test results | Use fallback tech or simplify features |
| Scope Creep | Define and freeze requirements | Weekly scope review meetings | Escalate to management, adjust timeline |

## Importance of RMMM

- Minimizes surprises during project execution
- Enables data-driven decision-making
- Ensures project continuity even when risks occur
- Helps deliver projects on time and within budget
- Improves stakeholder confidence

# RMMM Plan

- RMMM Plan typically includes the following key sections:

    1. Introduction
    2. Risk Identification
    3. Risk Analysis & Projection
    4. Risk Mitigation Plan
    5. Risk Monitoring Plan
    6. Risk Management (Contingency) Plan
    7. Risk Register (or Risk Table)
    8. Roles and Responsibilities
    9. Risk Review & Updating Process

## RMMM Plan

**1. Introduction:**

- Purpose of the RMMM plan
- Scope (what parts of the project it covers)
- Reference to the project plan or risk management policy

**2. Risk Identification:** List of potential risks that may impact the project, **Categorized into:**

- Project risks
- Technical risks
- Product risks
- People risks
- Organizational/external risks
- Tools used: brainstorming, checklists, past project data, risk taxonomy

# RMMM Plan

**3. Risk Analysis & Projection:**

**Evaluation of each risk:**

- Probability of occurrence (Low/Medium/High or percentage)
- Impact level (Low/Medium/High)
- Risk exposure (Risk = Probability × Impact)

- Use of risk matrix or table to prioritize
- **Optional:** assign numeric values to better assess severity

**4. Risk Mitigation Plan:**

- Strategies to reduce the probability or impact of each identified risk

**Example:**

- **Risk:** Team member may leave
- **Mitigation:** Cross-training, good work culture, incentives

## RMMM Plan

**5. Risk Monitoring Plan:**

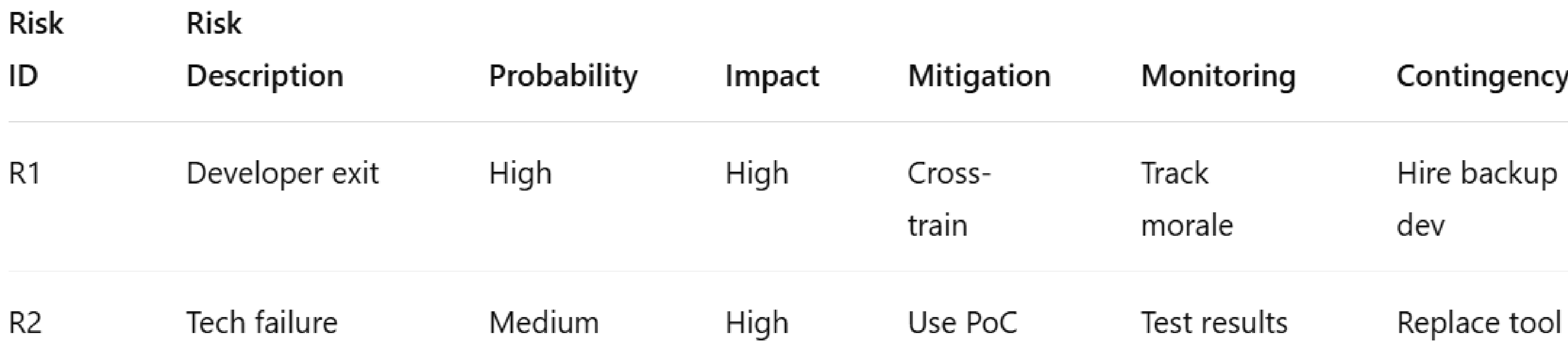
- How risks will be tracked during the project
- Indicators or trigger conditions for each risk
- Frequency of risk review meetings
- Person/team responsible for monitoring each risk

**6. Risk Management (Contingency) Plan:**

- Predefined actions if the risk occurs (backup plan)
- Reallocation of resources, alternate tools, or adjusted timelines
- Communication plan for stakeholders in case of escalated risk

**7. Risk Register (or Risk Table):** A tabular summary of all risks and corresponding plans

# RMMM Plan



| Risk ID | Risk Description | Probability | Impact | Mitigation | Monitoring | Contingency |
|---|---|---|---|---|---|---|
| R1 | Developer exit | High | High | Cross-train | Track morale | Hire backup dev |
| R2 | Tech failure | Medium | High | Use PoC | Test results | Replace tool |

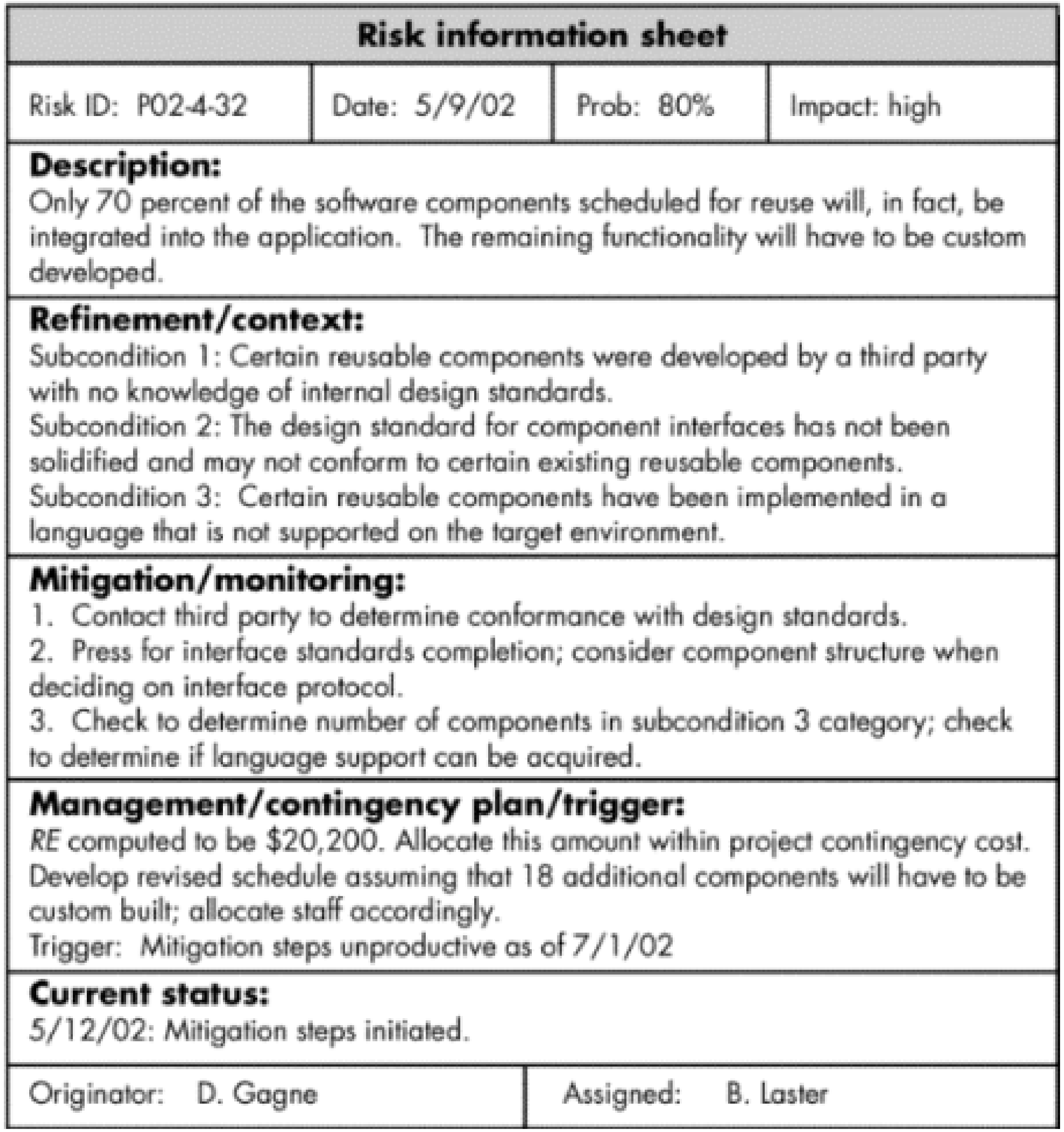
**8. Roles and Responsibilities:** Who is responsible for:

- Monitoring risks
- Executing mitigation plans
- Escalating risks when needed

**9. Risk Review & Updating Process:**

- How often the RMMM plan will be reviewed and updated
- Change control mechanism for risk addition or update

# RMMM Plan



**Risk information sheet**

| | | | |
|---|---|---|---|
| Risk ID: P02-4-32 | Date: 5/9/02 | Prob: 80% | Impact: high |

**Description:**
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.

**Refinement/context:**
Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards.
Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.

**Mitigation/monitoring:**
1. Contact third party to determine conformance with design standards.
2. Press for interface standards completion; consider component structure when deciding on interface protocol.
3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.

**Management/contingency plan/trigger:**
*RE* computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly.
Trigger: Mitigation steps unproductive as of 7/1/02

**Current status:**
5/12/02: Mitigation steps initiated.

| | |
|---|---|
| Originator: D. Gagne | Assigned: B. Laster |

## Maintenance Introduction

- Software products do not age or wear out like physical products.
- Then why is there a need to have maintenance of software products? Well, there are some factors which make it absolutely necessary.

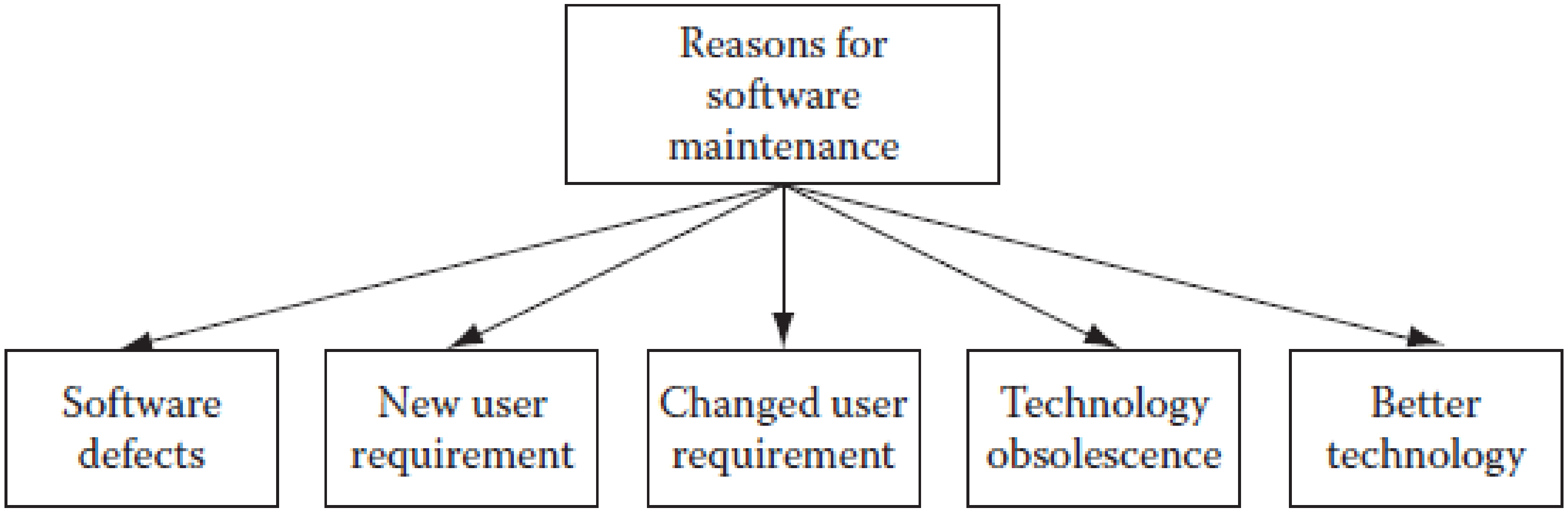
**Here are some of the reasons:**

1. ***Technology obsolescence*****:** The software platform (operating system, medium of user, interface) or the hardware platform on which the software product runs gets obsolete.
2. ***Software defects*****:** There are major software defects in the product and it is difficult to operate. For this reason, a software patch may be needed to be applied so that these defects are removed.

## Maintenance Introduction

3. ***Change in user requirements***: The business organization that was using the software product has seen a change in business transactions or business workflows that are not supported by the software product.

- It is estimated that more than 70% of all costs associated with software product development, implementation, and support and maintenance is consumed in the activities of supporting and maintaining software products.



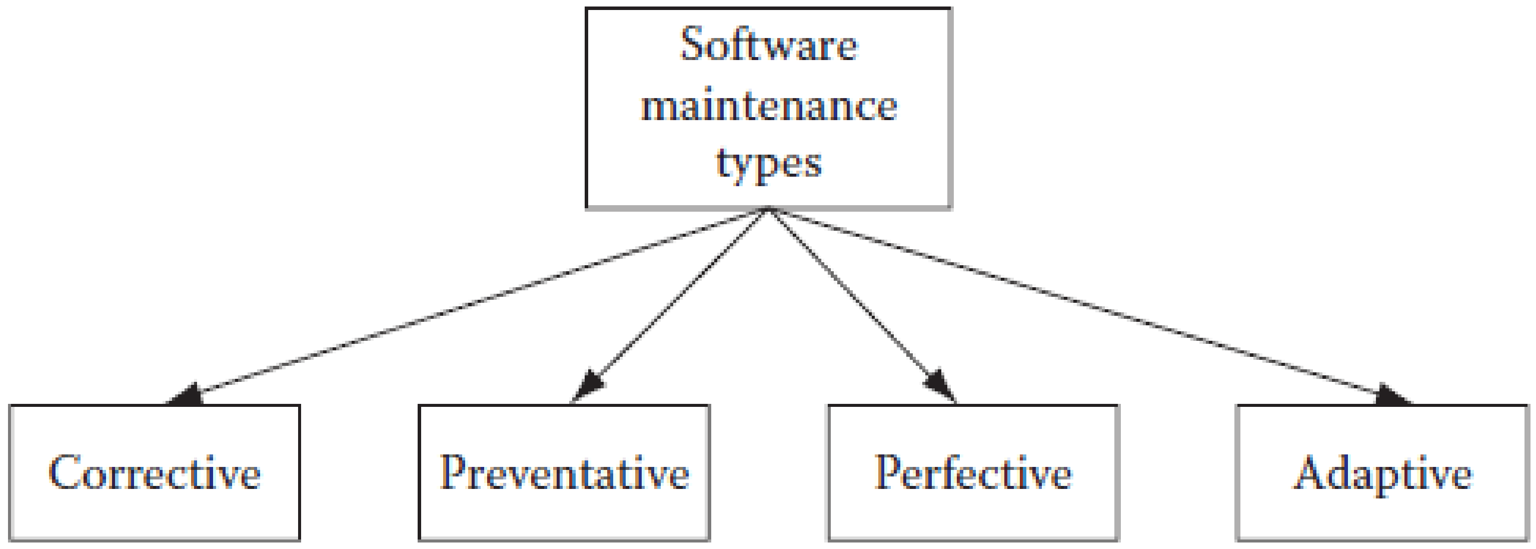
## Maintenance Introduction

- These kinds of queries have always puzzled the business community.
- This recognition has resulted in an awareness of the importance of finding ways to build such a software product.
- This situation has led to including maintainability characteristics during the entire product development cycle.
- Yet, a lot of work remains to be done during the maintenance phase of any software product.
- How to manage these activities so that costs can be minimized is an area of concern yet to be resolved.

## Maintenance Types

- Software maintenance is of four types: **corrective, adaptive, preventive, and perfective maintenance.**
- If the software has some defects, then it will take a corrective maintenance to rectify it.
- If there are some changes in the operating environment of the software product, then the product can be made useful by doing adaptive maintenance.
- If there is an insecurity that although the product is running fine in future we may have difficulty in using it, then preventive maintenance is employed.
- If there are some deficiencies in the product that must be rectified, then perfective maintenance will fit the bill (Figure below – Software maintenance types).

# Maintenance Types



## Maintenance Types

**Corrective:** Even after thorough reviews and testing, the software product contains many defects when it goes into production.

- These defects are uncovered as users start using the application.
- They are logged with the support staff and after a sizable number of errors are detected, the software vendor instructs his maintenance team to create a patch to rectify them.
- The maintenance team then makes a plan and fixes those defects.
- After application of the patch containing the fixes, the software starts running without these defects.

**Adaptive:** The operating environment in which a software product runs in operation includes the hardware and software platform as well as the interfaces for human and other machine interactions.

- If any of these change over time, it becomes difficult to run the software product.

## Maintenance Types

**Adaptive:**

- In such cases, it becomes necessary to do adaptive maintenance so that the software product becomes reusable.
- This kind of maintenance may involve changing the interface or porting the application to another hardware/ software platform.

**Perfective:**

- This kind of maintenance is needed when there is a change in the business environment, and thereby users need additional/modified functionality in the software product to do their tasks.
- A business workflow may have changed, a business transaction may have changed, or an altogether new business transaction was represented in the software product.
- For all these kinds of requirements, a perfective maintenance may be needed.

## Maintenance Types

**Preventive:**

- Generally, after a lapse of time, there are likely changes in business or operative environment, or there may be changes in hardware/software environment.
- These changes are bound to occur, and they affect the way the software product operates.
- Many of these changes can be perceived in advance.
- In such cases, preventive maintenance on the software product can make sure that the product will be useful even after these environmental changes occur.

## Maintenance Cost

- A software product is generally very valuable to an organization if it is used for doing a large portion of their daily business.
- If for some reason the software product has become unusable, then the organization in fact will be making losses on their revenue.
- Moreover, large enterprise software products are that much crucial!
- When the organization faces such a case, it is left with no alternative but to either get an entirely different software product that will replace the existing one or do maintenance of an existing product to make it usable.

## Maintenance Cost

- Following are some financial reasons for which a maintenance may be needed:

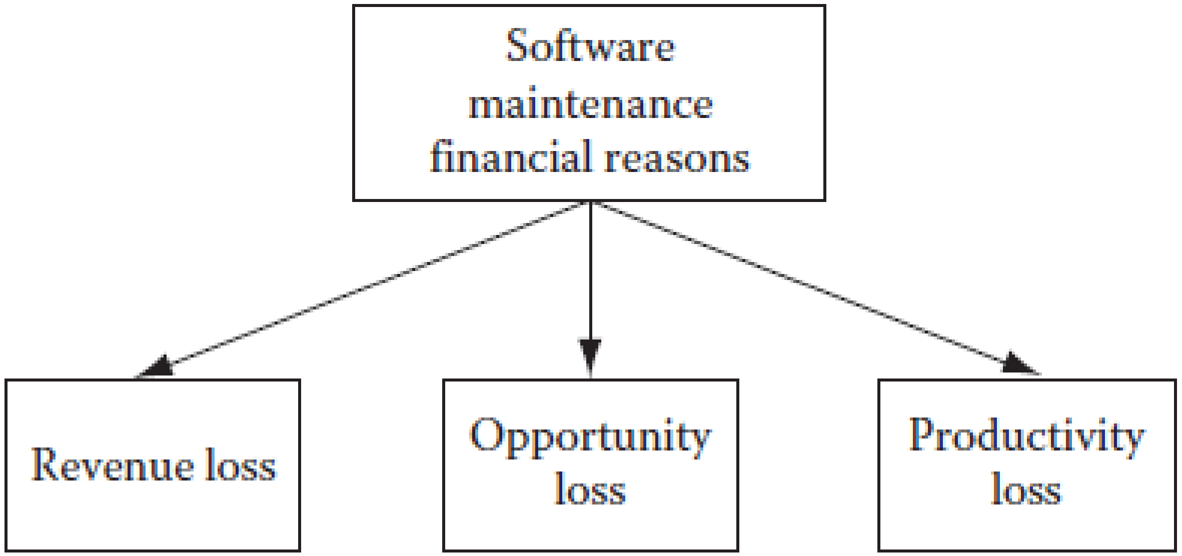
  1. ***Loss in business revenue*****:** It may happen that business transactions are faulty and thus the business may lose revenue.

  2. ***Opportunity loss*****:** Sometimes there could be some business opportunity in the marketplace, but due to some software problems it could not be availed.

  3. ***Productivity loss*****:** If the software product becomes difficult to operate due to many walk arounds or lengthy processing then productivity will become lower for business personnel

- Maintenance of an existing software product has its own share of problems.
- The maintenance will incur costs.

## Maintenance Cost

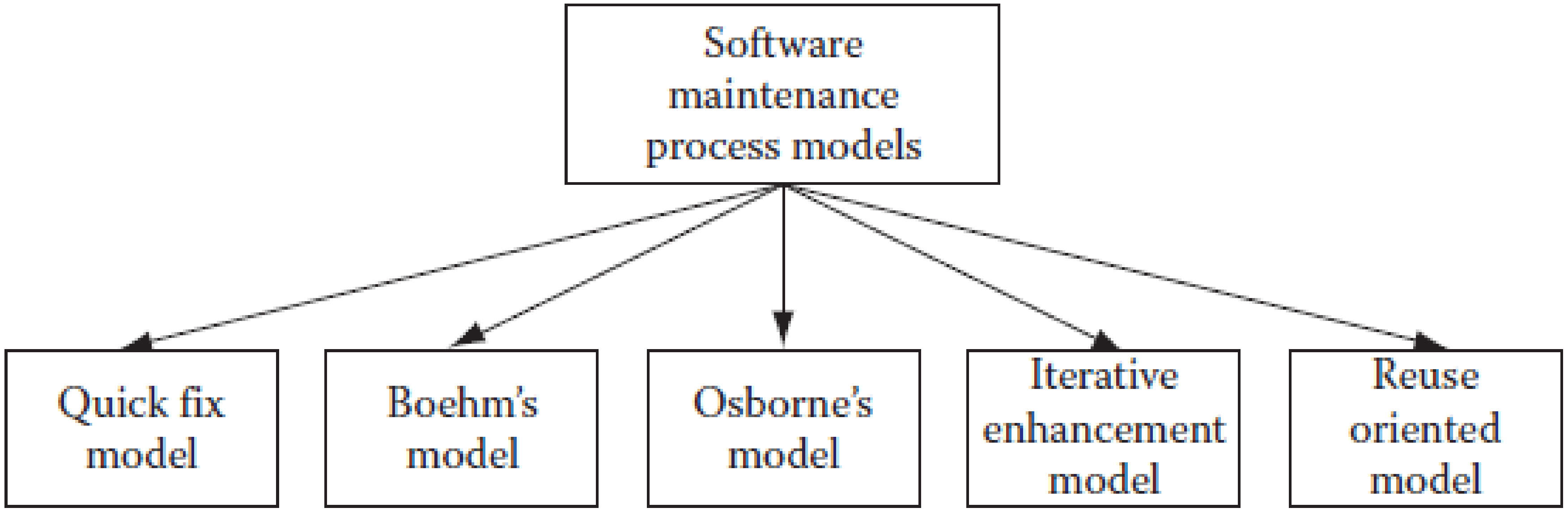


- A profit/loss analysis can be done, to see if it is more profitable to conduct a maintenance program on the software or keep using it as it is.
- The losses due to problems with the software can be compared to probable cost of maintenance and an ROI (return on investment) can be done.
- If we get a desirable ROI, then it is better to go for maintenance.

## Maintenance Process

- For any work, it is always better to have a process model instead of doing things on an ad hoc basis.
- When it comes to software maintenance, some process models have been defined.
- Some of the popular ones include the quick fix model, Boehm's model, Osborne's model, iterative enhancement model, and reuse oriented model (Below figure - Software maintenance models).
- ***Quick fix model*****:** This is the simplest of maintenance models; whenever any defects with the software products are found they are immediately fixed. There is no planning involved in the whole process, and it is mostly an ad-hoc approach.

## Maintenance Process



## Maintenance Process

- ***Boehm's model***: Boehm's model is based on economic models and often involves calculating ROI, for any planned maintenance.
- If ROI turns out to be good, then it is carried out or else it is dropped.
- ***Osborne's model***: Osborne realized that difficulties in carrying out maintenance work are due to gaps in communication.
- He proposed four steps to prevent this situation. He stated that a maintenance plan should include all change requests in the form of maintenance requirements.
- A quality assurance plan should accompany the maintenance plan. Metrics should be developed to measure and assess quality of work carried out during maintenance.
- Finally, reviews should be held after maintenance work to assess quality of work done.

# Maintenance Process

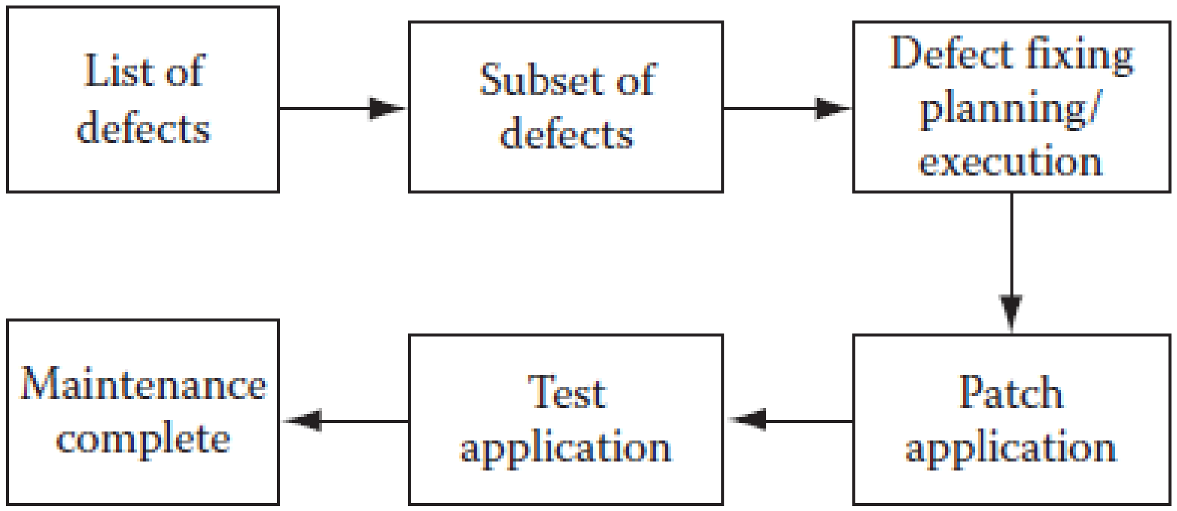
- ***Iterative enhancement model*****:** This model is based on the similar concept of iterative software development.
- All software defects and change requests are logged and then a small set from this list is taken for making fixes.
- This set is prepared based on the priority of changes required. High priority fixes are done before low priority fixes.
- ***Reuse oriented model*****:** This type of process is adopted for component-based software products.
- For fixing any defects, existing components are analyzed and then the appropriate changes are made.

# Maintenance Life Cycle

- Like the software development, software maintenance also has a life cycle.
- Requirements for software maintenance come from the list of defects that have been logged.
- Either the list of defects can be taken as a whole or a subset of defects from this list can be taken for a fixing plan.
- It makes a lot of sense to go for an iterative approach.
- This approach is like the concept of iterative software development.
- This way it can be ensured that highly visible, important, and priority defects are fixed first and other defects which do not make much impact on operations of the product are tackled later

# Maintenance Life Cycle

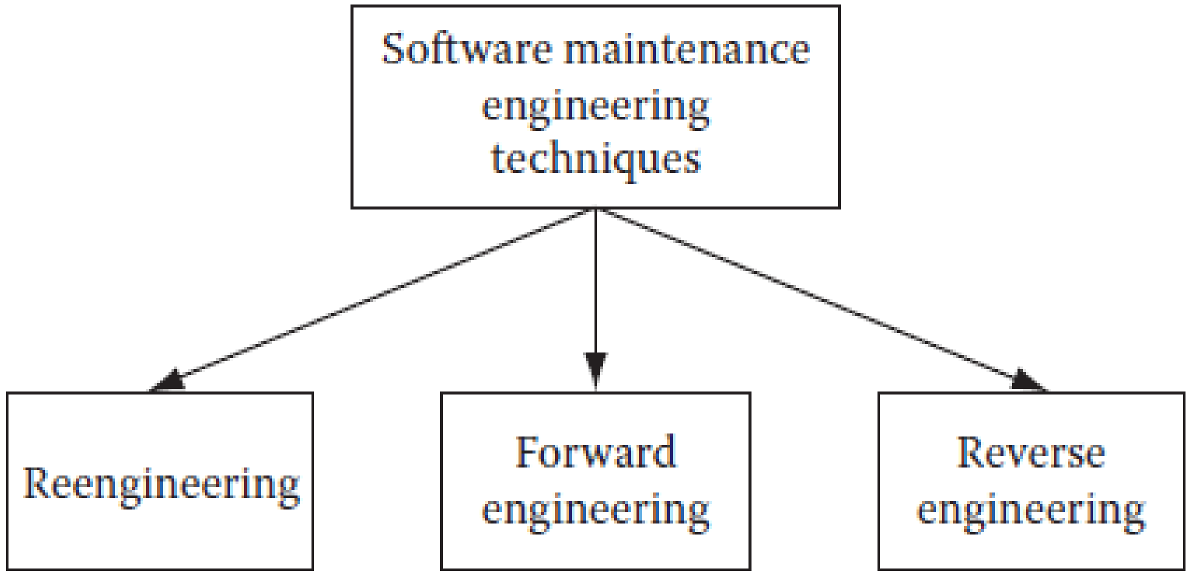
- In the software maintenance life cycle, testing is a crucial phase.
- This phase also consumes a lot of time and effort.
- But the value addition in all this effort and time spent helps in reducing defects, which in the long run is a much cheaper alternative compared to no testing/cursory testing and later spending money in providing support.



# Maintenance Techniques

- Maintenance of software products sometimes becomes a tough proposition.
- There is no proper documentation that can be used for understanding how the product is designed and constructed.
- Sometimes there is no documentation at all.
- Even if documentation is there, it is not up to date.
- This out-of-date documentation is not of much use for any maintenance work.
- Sometimes even if the documentation is up to date, the maintenance work is difficult due to dirty design or construction work.
- All these situations call for some specific techniques for maintenance work depending on the situation.
- Some of the common maintenance techniques include reengineering, reverse engineering, and forward engineering.

## Maintenance Techniques



## Maintenance Techniques

**Reengineering**

- Reengineering is also known as reuse engineering. This technique is a standard method for maintenance work for component-based software products.
- Details about all components in the software products are well known.
- When any maintenance work is needed, from the list of defects, each defect is specifically analyzed to find out the root cause of the defect.
- Once this analysis is successful, then fixing that defect becomes easy.

## Maintenance Techniques

**Reverse Engineering**

- Reverse engineering technique is the most useful when nonexistent or sketchy documentation is available for the software product.
- Due to unavailability of documentation, there is no information as to what the design is and how the product is constructed.
- In such a situation, it is almost impossible to do any modification in the source code for any maintenance work.
- In such cases, the reverse engineering technique is adopted.
- Using this technique, similar components or product parts are constructed as compared to existing product components/parts.
- This way the software product functionality is changed as the new constructed parts will have the desired functionality.

## Maintenance Techniques

**Forward Engineering**

- Forward engineering is just the opposite of reverse engineering.
- In this situation, we have ample documentation about the existing product.
- Due to new customer needs, the existing product needs to be extended so that the new needs can be fulfilled.
- All new extended development is based on the existing design and construction methods and will be made for the same hardware/software platform.

## Software Release – Case Study

- In the series of case studies, here is the piece related to software release and maintenance.
- SaaS vendor releases minor versions of its product on a quarterly basis and major versions on a yearly basis.
- For each minor release, new features to be added are carefully planned.
- The product manager makes sure that the release plan for a minor release will be on time by assigning priority to each new feature.
- The high priority features will be definitely added and the low priority features for that iteration will be added if any time remains in the iteration.

## Software Release – Case Study

- SaaS vendor does not release alpha or beta releases of its product as they do not serve mass markets.
- Their product is an enterprise computing product and is used by large retailers, government offices, logistics providers, manufacturers, and distributors.
- They always release new versions of their product to their existing customers.
- Since they do not do alpha or beta releases, they make sure that their new version is tested thoroughly by their testing team, and no major defects are passed in the production instances.
- Since there are no immediate customers who will be available for doing user acceptance testing, the internal testing team does the user acceptance testing as well.

# Software Maintenance – Case Study

- The software vendor keeps all of the production instances of its software product at its data centre (also known as operations center).
- All previous versions of the software as well as the current working version of the software product run at this centre as production instances of different versions of their software product.
- The maintenance team makes sure that all versions of the product are available for users.
- They run sanity test scripts daily on all instances.
- If any problems are found, then it is immediately resolved.
- These scripts are run at night.

# Software Maintenance – Case Study

- If any problems are found then, it is made sure that they are rectified before office hours start and people start using the application.
- In packaged software or custom-built software that are not used in a SaaS environment, this kind of quick fix is not possible.
- So in those cases, a maintenance plan is made to fix all or most defects found by users during a time span of 3 months or more.
- But with SaaS environments, this kind of maintenance is not needed at all.
- All defects are quickly and easily fixed, without hampering work of end users.

## REFERENCES

- Ashfaque Ahmed, Software Project Management: A Process-driven approach, Boca Raton, Fla: CRC Press, 2012.

THANK YOU