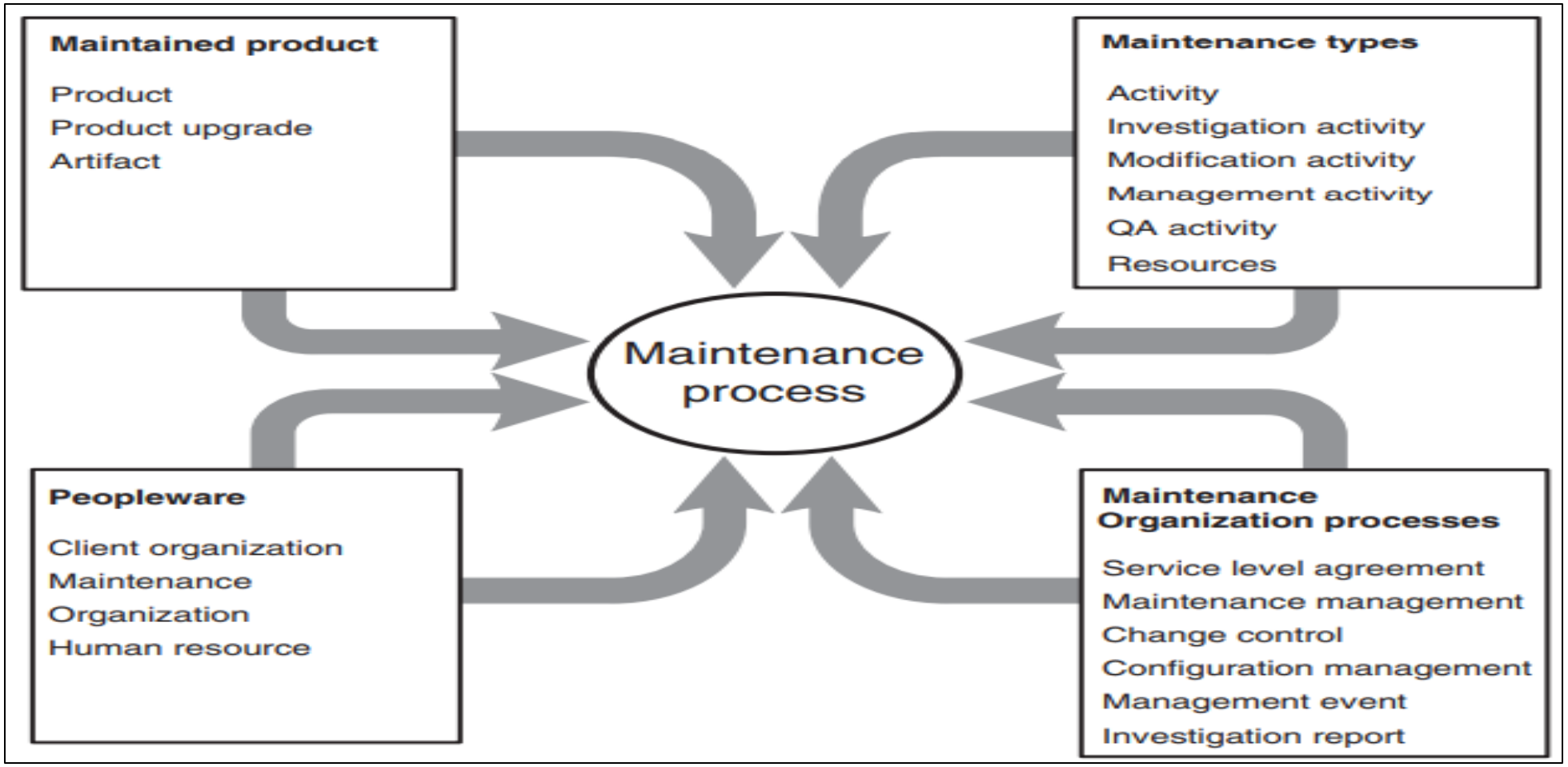
SOFTWARE RE-ENGINEERING

SE-409

Categories of Maintenance Concepts



Overview of concept categories affecting software maintenance

Maintained Product

- The maintained product dimension is characterized by three concepts:
  - **Product:** A product is a coherent collection of several different artifacts. Source code is the key component of a software product.
  - **Product upgrade:** Baseline is an arrangement of related entities that make up a particular software configuration. Any change or upgrade made to a software product relates to a specific baseline.
  - **Artifacts:** A number of different artifacts are used in the design of a software product. One can find the following types of artifacts: textual and graphical documents, component off-the-shelf products, and object code components.
- **The key elements of the maintained product are size, age, application type, composition, and quality.**

Maintenance Types

- **Activity:** A number of different broad classes of maintenance activities are performed on software products, including investigation, modification, management, and quality assurance.
- **Investigation activity:** This kind of activities evaluate the impact of making a certain change to the system.
- **Modification activity:** This kind of activities change the system's implementation.
- **Management activity:** This kind of activities relate to the configuration control of the maintained system or the management of the maintenance process.
- **Quality assurance activity:** This kind of activities ensure that modifications performed on a system do not damage the integrity of the system.
- **Resource:** A resource is a necessary asset whose main role is to help carry out a certain task or project. A resource can be a person, a team, a tool, finances, and time.

People ware

- Maintenance activities cannot ignore the human element, because software production and maintenance are human intensive activities.
- The three people-centric concepts related to maintenance are as follows:
  - **Maintenance organization:** This is the organization that maintains the product(s).
  - **Client organization:** A client organization uses the maintained system.
  - **Human resource:** Human resource includes personnel from the maintenance and client organizations.

Maintenance Organization Processes

- Two different levels of maintenance processes are followed within a maintenance organization:
  - **Individual-level maintenance processes** followed by maintenance personnel to implement a specific change request, and
  - **Organization-level process** followed to manage the change requests from maintenance personnel, users, and customers/clients.

Maintenance Organization Processes

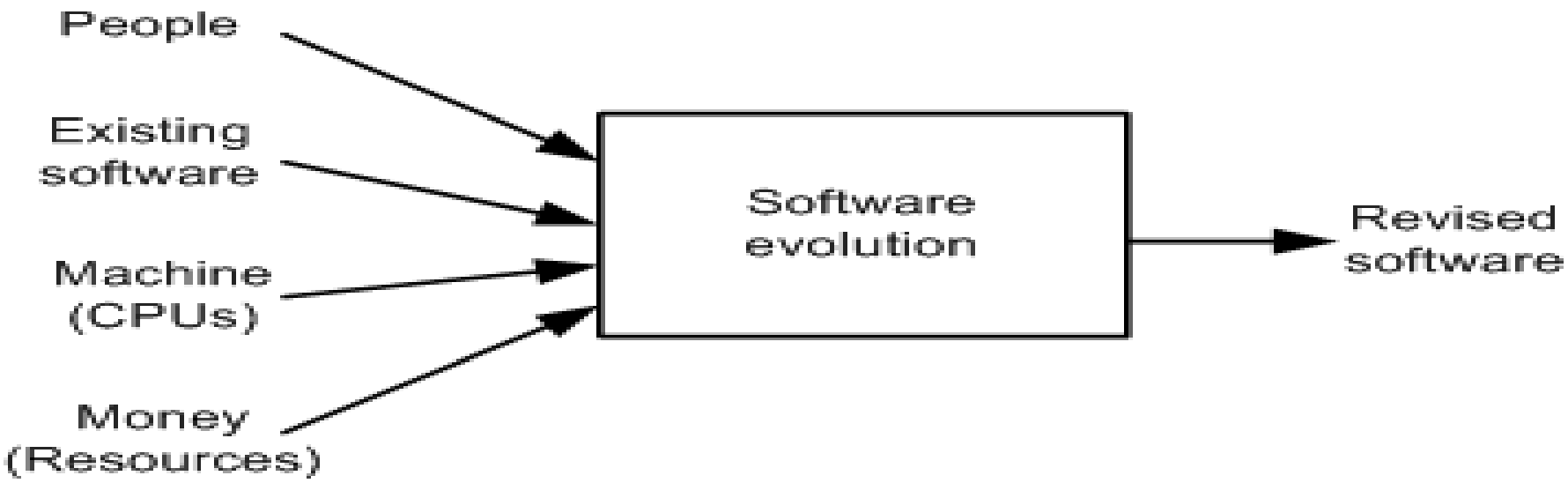
Major elements of a maintenance organization are:

- **Service-Level Agreement (SLA) :** A proposed modification activity is scheduled only after the modification is approved by the board and an Service Level Agreement (SLA) is signed with the client. Service level agreement (SLA) is a contract between the customers and the providers of a maintenance service.
- **Maintenance Management:** This process is used to manage the maintenance service, which is not the same as managing individual CRs.
- **Change Control:** Evaluation of results of investigations of maintenance events is performed in a process called change control. Based on the evaluation, the organization approves a system change.

Maintenance Organization Processes

- **Configuration management:** A system's integrity is maintained by means of a configuration management process. Integrity of a product is maintained in terms of its modification status and version number.
- **Maintenance Event:** This is a problem report or a CR originating from within the maintenance organization or from the customers.
- **Investigation Report:** This is the outcome of assessing the cause and impacts of a maintenance event.

Evolution of Software Systems



Inputs and outputs of software evolution

SPE Taxonomy

- The abbreviation SPE refers to
  - S (Specified),
  - P (Problem), and
  - E (Evolving) programs
- In 1980, Meir M. Lehman proposed an SPE classification scheme to explain the ways in which programs vary in their evolutionary characteristics.

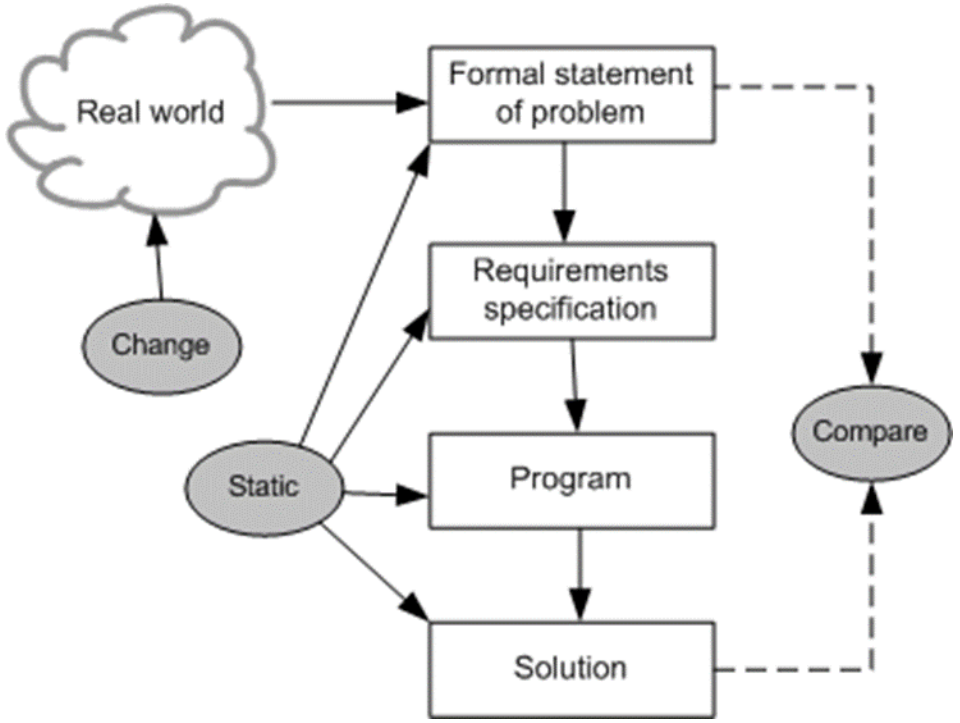
SPE Taxonomy

- S-type (Specified) programs have the following characteristics:
  - All the non-functional and functional program properties, that are important to its stakeholders, are formally and completely defined.
  - Correctness of the program with respect to its formal specification is the only criterion of the acceptability of the solution to its stakeholders.

Examples of S-type programs:

(i) Calculation of the lowest common multiple of two integers.

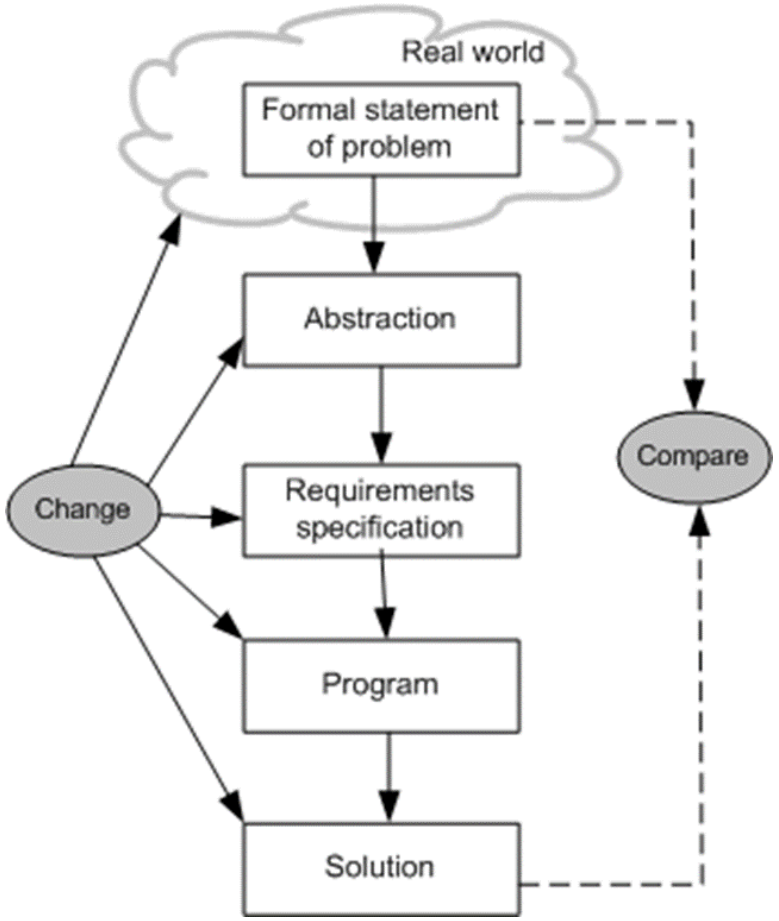
(ii) Perform matrix addition, multiplication, and inversion.



S-type programs

SPE Taxonomy

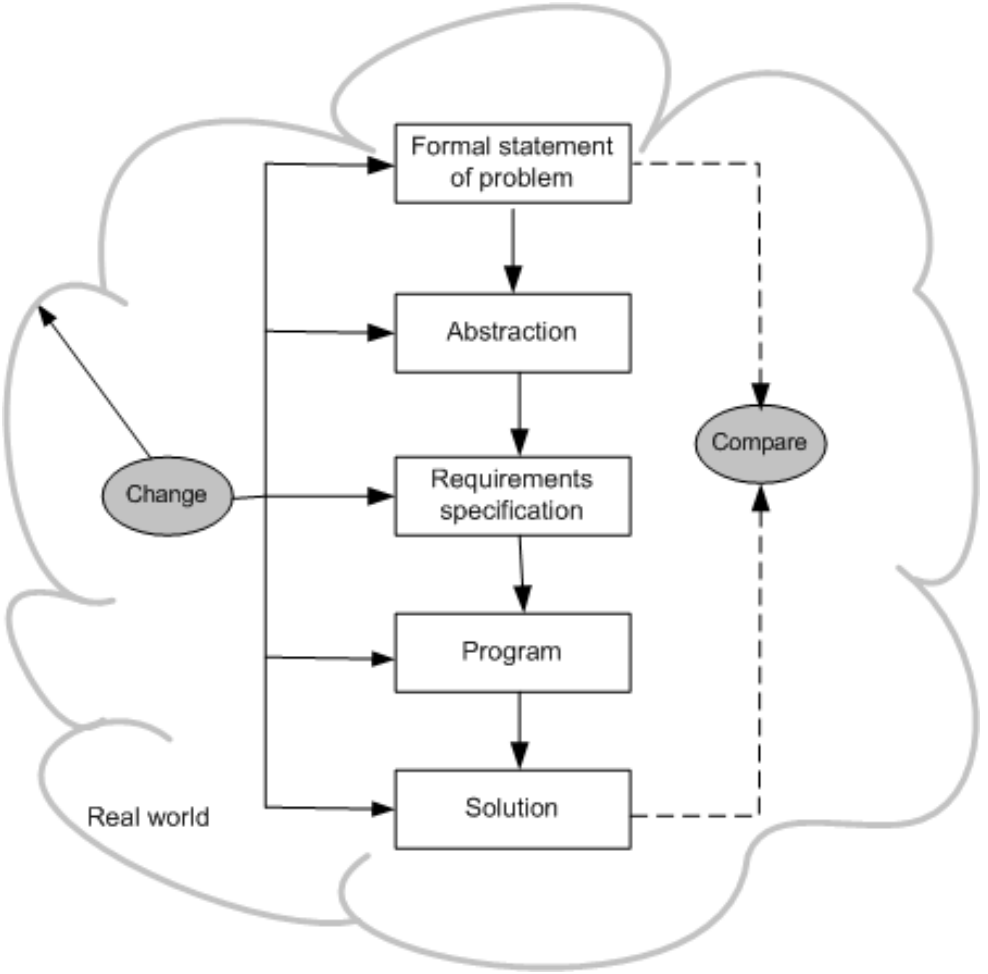
- P-type (Problem) program is based on a practical abstraction of the problem, instead of relying on a completely defined specification.
- Example: The goal is to create a program that can detect and filter out spam emails. However, spam emails continuously evolve, as spammers find new ways to bypass detection systems. The problem is not static, as the types of spam and techniques used by spammers change frequently.



P-type programs

SPE Taxonomy

- An E-type (Evolving) program is one that is embedded in the real world and it changes as the world does.
  - These programs mechanize a human or society activity, make simplifying assumptions, and interface with the external world by requiring or providing services.
  - The acceptance of an E-type program entirely depends upon the stakeholders' opinion and judgment of the solution.



E-type programs

Software Fault Classification

Software Fault Classification:

- Bohrbugs
- Mandelbugs
- Aging related Bugs

- Bohrbugs: Software bugs that are reproducible, easily found and (often) fixed during the testing and debugging phase
- Mandelbugs: Software bugs that are hard to find and fix; (often) remain in the software during the operational phase. These bugs may never be fixed, but if the operation is retried or the system is rebooted, the bugs may not manifest themselves as failures.
- Yet another cause software failures is resource exhaustion, e.g., memory leakage, swap space fragmentation. Software appears to “Age” due to resource exhaustion.

Fundamental concepts of software aging

- **Software aging** is a phenomenon that causes system performance degradation and eventual failures.
- A general characteristic of this phenomenon is the fact that, as the run-time of the system or process increases, its failure rate also increases.
- Software aging is the tendency for software to fail or cause a system failure after running continuously for a certain time, or because of ongoing changes in systems surrounding the software.
- Failure can take the form of incorrect service (e.g., erroneous outcomes), no service (e.g., halt and/or crash of the system), or partial failure (e.g., gradual increase in response time).

Fundamental concepts of software aging

- Aging effects can also be classified into **volatile** and **nonvolatile** effects.
- They are considered volatile if they are removed by re-initialization of the system or process affected, for example via a system reboot. In contrast, non-volatile aging effects still exist after reinitializing of the system/process.
- Physical memory fragmentation and OS resource leakage are examples for volatile aging effects. File system and database metadata fragmentation are examples for non-volatile aging effects.
- Aging effects in a system can only be detected while the system is running, by monitoring **aging indicators**. Aging indicators are markers for aging detection, like antigens are markers to detect cancer disease.

Fundamental concepts of software aging: Aging Indicators

- Aging Indicators: To detect and monitor software aging, various aging indicators are used. These indicators can include system performance metrics (e.g., response time, throughput), resource consumption (e.g., CPU usage, memory usage), error rates, system logs, and system health monitoring data.
- Tracking these indicators helps identify signs of aging and potential areas for improvement.

Tools for Aging Indicators:

- Performance Monitoring Tools: New Relic, AppDynamics
- Resource Monitoring Tools: Nagios, Zabbix
- Profiling and Performance Analysis Tools: Java VisualVM, Visual Studio Profiler (for .NET applications)

Fundamental concepts of software aging: Software rejuvenation

- Software rejuvenation: Proactive fault management technique aimed at postponing/preventing crash failures and/or performance degradation.
- Software rejuvenation refers to the process of periodically restarting or reinitializing components or the entire system to mitigate the effects of aging. Rejuvenation aims to remove accumulated errors, reset resource states, and restore the system to a more stable and reliable state.
  - Involves occasionally stopping the running software, “cleaning” its internal state and/or its environment and restarting it. Rejuvenation of the environment, not of software.
- **Common techniques for cleaning**: Software rejuvenation is the concept of periodically stopping the running software, cleaning its internal state through garbage collection, defragmentation, flushing operating system kernel tables and reinitializing internal data structures, and restarting it.

Example of software aging

- **Scenario: Memory Leaks**
  Description: Over time, a software application may experience memory leaks, where memory allocated for certain operations is not properly released. This can cause the application to consume excessive memory resources, leading to performance degradation or even crashes.
- **Solution:** Regularly perform memory profiling and debugging to identify and fix memory leaks. Use tools like memory analyzers to detect and resolve memory leaks in the code. Additionally, follow best practices for memory management, such as deallocating resources properly and optimizing memory usage.

Fundamental concepts of hazard & mishap

- Hazard and mishap are related terms commonly used in the context of safety and risk management.
- A hazard refers to any potential source or situation that has the potential to cause harm, injury, damage, or adverse effects to people, property, or the environment.
- A mishap, on the other hand, refers to an unintended event, accident, or incident that causes harm, damage, or unintended consequences. It represents the actual occurrence or manifestation of an undesired outcome.
- Hazards are the potential risks or dangers that exist in a situation, while mishaps are the actual incidents or accidents that result from those hazards.

Example of hazard & mishap

For each of the following situations, explain whether it is a hazard or a mishap:

Scenario: Water in a swimming pool becomes electrified.

Solution: This situation can be considered a hazard. The presence of electricity in the water poses a potential risk of electric shock to individuals who come into contact with it. It represents a potential danger or source of harm, highlighting the hazard of electric shock in the swimming pool.

Example of hazard & mishap

- **Scenario:** E-commerce Checkout Failure
- **Solution:** This situation can be considered a Mishap. Because during a peak shopping period, an e-commerce website experiences an issue where the checkout process fails to complete transactions for a significant number of customers. This results in frustrated customers, lost sales, and damage to the company's reputation.