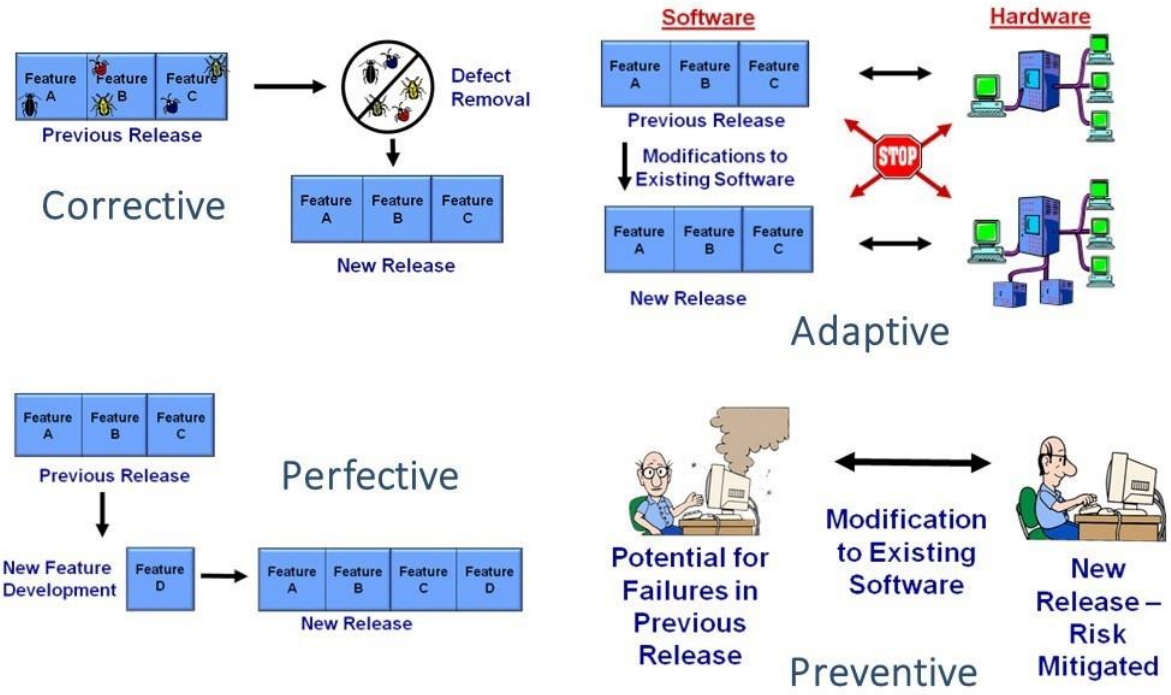
Types of Maintenance

By Linda Westfall

lwestfall@westfallteam.com



Successful software products tend to have very long life spans when measured from initial release to final retirement. For example, the initial release of Microsoft Word was in 1983, so that software product has been in use for more than 30 years with no retirement in sight. However, according to Sommerville, "During their lifetime, operational software systems have to change if they are to remain useful." [Sommerville-16]

The need to change existing software products may result from:

- The need for additional or changed functionality
- Incremental or evolutionary development
- Defects in the work products
- Changes to the business climate or business needs

- Changes to stakeholder requirements, industry standards, or regulations
- Changes to the hardware or target platform
- Technological advancements
- Competitive motivations and threats
- The need for more reliability, usability, security, or safety in the software

Therefore, changes are inevitable if the software is to remain technically current, competitive in the marketplace, retain its value, and continue to meet the stakeholders' needs.

Software maintenance is the process of making changes to software products after they have been released into operations. In some cases, these software products may be legacy software, a term used to mean older software written using antiquated methods, programming languages, or technologies. Legacy software refers to software that is out of date and that needs to be re-engineered or replaced but that is still in use in operations.

Software maintenance adapts, optimizes, enhances, or corrects defects in a released software product while still preserving that product's quality and integrity. In other words, the objective of software maintenance is to change part of a released software product without breaking another part of that product.

Since agile development processes are based on a continuous flow of software development supported by an emerging product backlog, there is no real distinction between initial development and ongoing post-delivery maintenance. All iterations after the software's initial release could be considered maintenance iterations.

Corrective Maintenance

Even with the best state-of-the-practice software quality assurance and control techniques, it is likely that the users of the software will encounter at least a few software failures once the software products are delivered to operations. The IEEE *Standard for Software Engineering—Software Life Cycle Processes— Maintenance* defines corrective maintenance as "the reactive modification of a software product performed after delivery to correct discovered problems." [IEEE-06] Corrective maintenance is performed to repair the defects that cause operational failures or other issues. Corrective maintenance can also be performed to correct latent defects that were found in released software during subsequent testing cycles.

Perfective Maintenance

As the software is used, the users, customers, or other external stakeholders may discover requirements that they need to add to the software product. Marketing or engineering may also come up with new requirements to add to the software to continue to innovate the software, to remain technically current, or to meet competitive threats. The IEEE/ISO/IEC Systems and Software Engineering—Vocabulary defines perfective maintenance as "improvements in the software's performance or functionality, for example, in response to user suggestions or requests." [IEEE/ISO/IEC-17] Perfective maintenance is performed to add new requirements to the software (to "perfect" the product). Perfective maintenance is the normal enhancement of successful, working software products over time so that the software retains its value to the stakeholders.

Adaptive Maintenance

Adaptive maintenance is Modifications to an existing software product performed after delivery, to adapt that product to new or changed external interfaces, environments, platforms, operating systems, or other elements. As time progresses, a software product's operational environment is likely to change. Examples of changes to the external environment in which that software must function include:

- External interface definitions may change (for example, communications, protocols, external data file structures)
- Government regulations or other business rules may change (for example, tax code or regulatory reporting requirements)
- Hardware and software that interface with the product may change (for example, new versions of the operating system, new or updated peripherals or other hardware, new or updated interfacing software applications, additional or modified fields/records in external databases)

Preventive Maintenance

Preventive maintenance is "modification of a software product after delivery to detect and correct latent faults in the software product before they manifest as failures." [IEEE-06]

Preventive maintenance is a proactive approach to prevent problems with the software before they occur. For example, additional error handling and fault tolerance might be added to a safety-critical system, or an enhanced encryption methodology might be added to improve software security. Self-diagnostic testing might also be added to the software product.

As software products change over time they can deteriorate, especially if good design techniques were not employed when the software was originally developed. It may become necessary to modify an existing, released software product by refactoring it to eliminate duplication, complexity, or awkward structure, or to reengineer it to make it easier to maintain going forward. This is another type of preventive maintenance.

References

IEEE-06: *IEEE Standards Software Engineering, Standard for Software Engineering—Software Life Cycle Processes—Maintenance, ISO/IEC 14764.* New York, NY: The Institute of Electrical and Electronics Engineers. 2006.

IEEE/ISO/IEC-17: *IEEE/ISO/IEC 24765 Systems and Software Engineering—Vocabulary.* Geneva, Switzerland: ISO. New York, NY: The Institute of Electrical and Electronic Engineers. 2017.

Sommerville-16: Sommerville, Ian. *Software Engineering. Tenth Edition,* Boston: Pearson. 2016.

