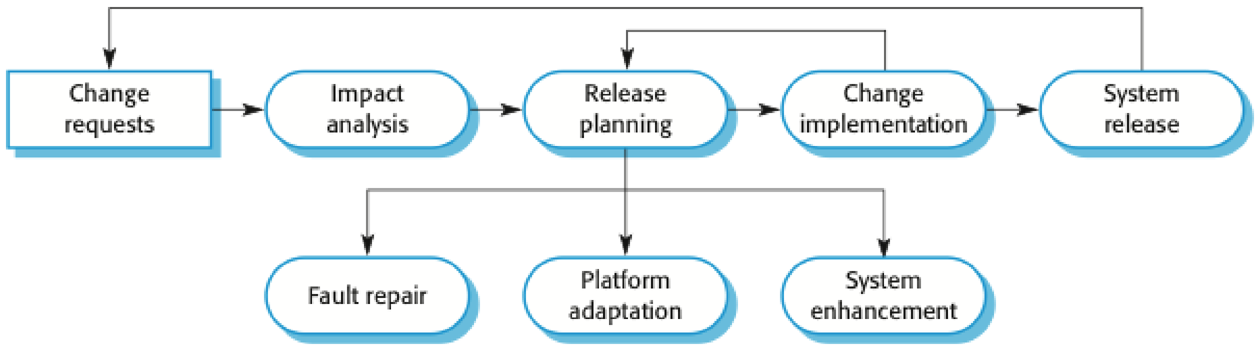
Software Evolution Process:

Software Evolution is a term that refers to the process of developing software initially, then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities, etc. The evolution process includes fundamental activities of change analysis, release planning, system implementation, and releasing a system to customers.
Software evolution processes depend on:

1. The type of software being maintained;
2. The development processes used;
3. The skills and experience of the people involved.

Proposals for change are the driver for system evolution. These should be linked with components that are affected by the change, thus allowing the cost and impact of the change to be estimated. Change identification and evolution continues throughout the system lifetime.



Change implementation can be viewed as an iteration of the development process where the revisions to the system are designed, implemented, and tested. A critical difference is that the first stage of change implementation may involve program understanding.

Software Evolution vs Software Maintenance:

A possible distinction is:

- *Software Maintenance*: The activities required to keep a software system operational and responsive after it is deployed.

- *Software Evolution*: A continuous change from a lesser, simpler, or worse state to a higher or better state.

Maintenance:

Maintenance has traditionally been divided into four types.

1. Perfective maintenance (50%): Enhancements where new operations and refinements are added to existing functions.
2. Adaptive maintenance (25%): Modifying the application to meet new operational circumstances.
3. Corrective maintenance (21%): Eliminating errors in the program's functionality.
4. Preventive maintenance (4%): Modifying a program to improve its future maintainability.

Some authors will consider emergency maintenance as being a type of corrective maintenance that is not scheduled.

The cost involved in evolution and maintenance:

Some common **technical** cost factors are:

1. **The complexity of code**: Complex control and logic structure are hard to understand and therefore hard to change.
2. **Changes in programming languages**: If the code to change is written in a language that is no longer commonly used, it is hard to change. Even harder is if new code has to be written in a different language.
3. **Changes in software infrastructure**: If the underlying software, middleware, or libraries have changed, then the programmers have to understand how the software to change interacts with them, which is very difficult.
4. **Quality of code formatting and style**: Messes always create maintenance debt. Perhaps the worst problem is when names are poorly chosen.