CHAPTER 6

RECOMMENDATIONS

The impact of ICT towards Software Reuse Technology is another important, interesting and demanding research area. Hence, I wish to recommend for study in this area.

6.1. CODE REUSE

It is also called software reuse, is the use of existing software, or software knowledge, to build new software **[102].** Code reuse has been practiced from the earliest days of programming. Programmers have always reused sections of code, templates, functions, and procedures. Software reuse as a recognized area of study in software engineering, however, dates only from 1968 when Douglas McIlroy of Bell Laboratories proposed basing the software industry on reusable components.

Code reuse is the idea that a partial computer program written at one time can be, should be, or is being used in another program written at a later time. The reuse of programming code is a common technique which attempts to save time and energy by reducing redundant work.

For newly written code to use a piece of existing code, some kind of interface, or means of communication, must be defined. These commonly include a "call" or use of a subroutine, object, class, or prototype. In organizations, such practices are formalized and standardized by domain engineering aka software product line engineering.

The general practice of using a prior version of an extant program as a starting point for the next version, is also a form of code reuse.

Some so-called code "reuse" involves simply copying some or all of the code from an existing program into a new one. While organizations can realize time to market benefits for a new product with this approach, they can subsequently be saddled with many of the same code duplication problems caused by cut and paste programming.

Many researchers have worked to make reuse faster, easier, more systematic, and an integral part of the normal process of programming. These are some of the main goals behind the invention of object-oriented programming, which became one of the most common forms of formalized reuse. A somewhat later invention is generic programming.

Another, newer means is to use software "generators", programs which can create new programs of a certain type, based on a set of parameters that users choose. Fields of study about such systems are generative programming and meta programming.

6.2. TYPES OF REUSE

Concerning motivation and driving factors, reuse can be:

- Opportunistic - While getting ready to begin a project, the team realizes that there are existing components that they can reuse.
- Planned - A team strategically designs components so that they'll be reusable in future projects.

Opportunistic reuse can be categorized further:

- Internal reuse - A team reuses its own components. This may be a business decision, since the team may want to control a component critical to the project.
- External reuse - A team may choose to license a third-party component. Licensing a third-party component typically costs the team 1 to 20 percent of what it would cost to develop internally **[103]** .The team must also consider the time it takes to find, learn and integrate the component.

Concerning form or structure of reuse, code can be **[104]** :

- Referenced - The client code contains a reference to reused code, and thus they have distinct life cycles and can have distinct versions.
- Forked - The client code contains a local or private copy of the reused code, and thus they share a single life cycle and a single version.

Fork-reuse is often discouraged because it's a form of code duplication, which requires that every bug is corrected in each copy, and enhancements made to reused code need to be manually merged in every copy or they become out-of-date. However, fork-reuse can have benefits such as isolation, flexibility to change the reused code, easier packaging, deployment and version management **[105].**

6.3. EXAMPLES

6.3.1. Software libraries. A very common example of code reuse is the technique of using a software library. Many common operations, such as converting information among different well-known formats, accessing external storage, interfacing with external programs, or manipulating information (numbers, words, names, locations, dates, etc.) in common ways, are needed by many different programs. Authors of new programs can use the code in a software library to perform these tasks, instead of "re-inventing the wheel", by writing fully new code directly in a program to perform an operation. Library implementations often have the benefit of being well-tested, and covering unusual or arcane cases. Disadvantages include the inability to tweak details which may affect performance or the desired output, and the time and cost of acquiring, learning, and configuring the library **[106].**

6.3.2. Design patterns. Main article: Design pattern (computer science)

A design pattern is a general solution to a recurring problem. Design patterns are more conceptual than tangible and can be modified to fit the exact need. However, abstract classes and interfaces can be reused to implement certain patterns.

6.3.3. Frameworks. Main article: Software framework

Developers generally reuse large pieces of software via third-party applications and frameworks. Though frameworks are usually domain-specific and applicable only to families of applications.

6.4. SYSTEMATIC SOFTWARE REUSE

Systematic software reuse is still the most promising strategy for increasing productivity and improving quality in the software industry. Although it is simple in concept, successful software reuse implementation is difficult in practice. A reason put forward for this is the dependence of software reuse on the context in which it is implemented. Some problematic issues that needs to be addressed related to systematic software reuse are **[107].** A definition of software reuse is the process of creating software systems from predefined software components.

6.5. ADVANTAGE OF SOFTWARE REUSE [108]:

- The systematic development of reusable components.
- The systematic reuse of these components as building blocks to create new systems.

A reusable component may be code, but the bigger benefits of reuse come from a broader and higher-level view of what can be reused. Software specifications, designs, tests cases, data, prototypes, plans, documentation, frameworks, and templates are all candidates for reuse. Software reuse can cut software development time and costs. The major advantages for software reuse are to:

- Increase software productivity.
- Shorten software development time.
- Improve software system interoperability.
- Develop software with fewer people.
- Move personnel more easily from project to project.
- Reduce software development and maintenance costs.
- Produce more standardized software.
- Produce better quality software and provide a powerful competitive advantage