Bachelor of Science in Computer Science (BSc in CS)

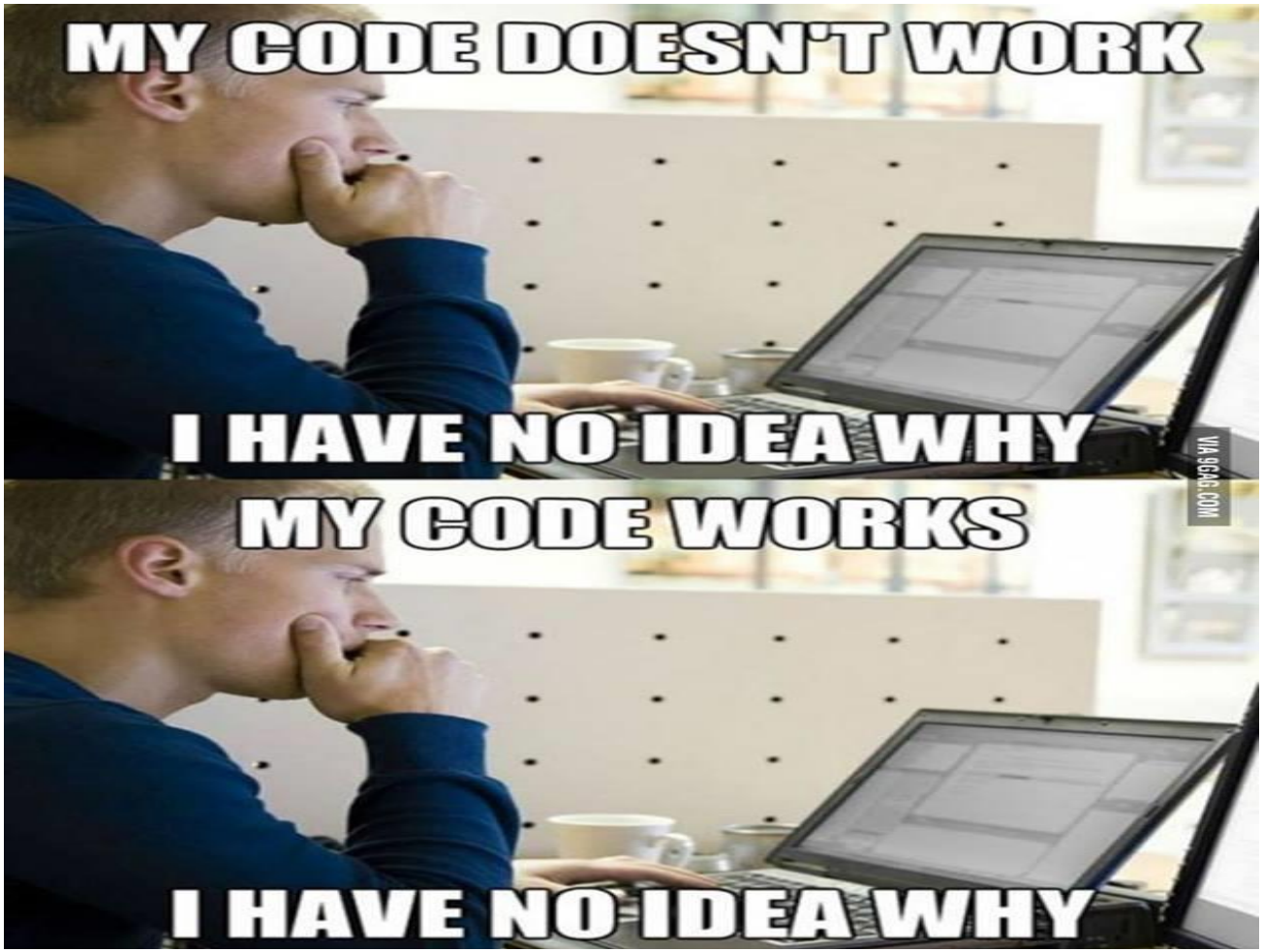
CS1303: Introduction to Software Engineering

Implementation

Prof. K. P. Hewagamage

UCSC

It works with or without your understanding



Learning Objectives

- Select appropriate programming language and development tools for a given problem
- Identify the features of a good program, good programming practices and program documentation
- Adopt Modern Practices of Code Development

Implementation (Coding)

- The goal of the coding is to implement the design in the best possible manner.
- Coding activity affects both testing and maintenance phases.
- Time spent in coding is a small percentage of the total software cost, but the goal should be to reduce the cost of later phases.
- Having readability and understandability as a clear objective of the coding activity can itself help in producing software that is more maintainable.

What is Implementation?

- Transforms the design specification to source code that can be executed on a computer.
- Coding is relatively straight forward given a design Specification?
- Coding is a minor activity compared to the other phases of development.
- A good design may be spoiled by the bad choice of a language.
- However, a bad design cannot be corrected through coding.

What is Implementation?

- Choice of the language and the coding style are important issues to consider.
- The programmer translates the design into source code of the chosen programming language.
- The language translator converts the source code in to executable code in several steps.
- Certain design issues may not be supported by the language in which case the coder may choose to violate design.
- Although design quality should not be compromised because of a language issue, design approach may depend on the language choice.

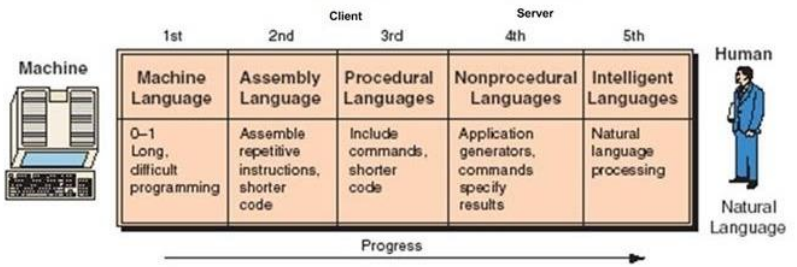
PROGRAMMING LANGUAGES AND DEVELOPMENT TOOLS

Evolution of Programming Languages

1. First generation (1GL) – Binary languages
2. Second generation (2GL) – Assembly languages
3. Third generation (3GL) - C, Pascal
   - Hardware-independent, no longer a one-to-one
   - Structured programming was introduced
4. Forth generation (4GL)
   - domain specific languages like MATLAB
   - Query languages such as SQL

- Fifth generation – AI? Neural networks?
  - Logic programing - Prolog

Assembly Language	Machine Language
`ST 1,[801]`	`00100101 11010011`
`ST 0,[802]`	`00100100 11010100`
`TOP: BEQ [802],10,BOT`	`10001010 01001001 11110000`
`INCR [802]`	`01000100 01010100`
`MUL [801],2,[803]`	`01001000 10100111 10100011`
`ST [803],[801]`	`11100101 10101011 00000010`
`JMP TOP`	`00101001`
`BOT: LD A,[801]`	`11010101`
`CALL PRINT`	`11010100 10101000`
	`10010001 01000100`



Programming Paradigms

- **Procedural/Structured Programming**
  - uses a list of instructions to tell the computer what to do step-by-step
  - based upon the concept of calling procedure
    - FORTRAN, ALGOL, COBOL, BASIC, Pascal and C.

```
result = [];
for i = 0; i < length(people); i++ {
    p = people[i];
    if length(p.name)) > 5 {
        addToList(result, toUpper(p.name));
    }
}
return sort(result);
```

Programming Paradigms

- **Object Oriented Programming**
  - OOP is based on the sending of messages to objects
  - Objects respond to messages by performing operations, generally called methods

```
result = []
for p in people {
    if p.name.length > 5 {
        result.add(p.name.toUpper);
    }
}
return result.sort;
```

Programming Paradigms

- **Functional Programming**
  - A programming paradigm in which try to bind everything in pure mathematical functions style.
  - It uses expressions instead of statements.
    - expression is evaluated to produce a value
    - statement is executed to assign variables
  - Haskell, JavaScript, Scala, Erlang, Lisp, ML, Clojure, OCaml, Common Lisp, Racke

```
show_output(f)

    f();

print_gfg()
    print("hello gfg");

show_output(print_gfg)
```

Programming Paradigms

- **Declarative Programming**
  - Control flow in declarative programming is implicit

```
select upper(name)
from people
where length(name) > 5
order by name
```

Language Features

This table describes the strengths and weaknesses of some of the more important languages in current use.

Language	Features	Strengths	Weaknesses
Ada	- Procedural -Some object orientation built in exception handling. - Strong type checking - No pointers - Defined standards	-Strong type checking reduces errors on applications. - MS DOS support -Good for safety critical and military applications. -Compliers meet the defined standards.	- Large run time system requires. -Expensive to develop applications. -Requires powerful machine to run on.
C	- Procedural - Weak type checking - Very low level pointers - Flexible	- Close to hardware/OS -Fast and efficient applications can be built. - Widely used	-Poor exception handling support. - Memory handling leads to unreliable code. - Compilers all different, don't meet standard.

Language Features

Language	Features	Strengths	Weaknesses
C++	- OO extension to C. - Weak type checking - Flexible - Pointers	- All those of C and OO concepts of Polymorphism, Inheritance (single & multiple), Encapsulation etc.	- As for C and can simple write C.
COBOL	- Procedural -Strong I/O handling for transaction processing (TP). - Defined standards	-Suited to batch TP applications - Widely used -Most 4GLs interface to COBOL.	-Large run time system required. - Old - many features simply added on later. -Not all compilers meet standard.
Fortran	- Procedural -Strong arithmetic support through libraries.	- Suited to data analysis where significant arithmetic processing required.	- Old - more modern languages provide most of the features.

Language Features

Language	Features	Strengths	Weaknesses
Java	- Object Oriented -Better type checking than C but still reasonably weak. -Standard defined by Sun Microsystems.	- Platform independent. -Dynamic downloading of classes. -Good user interface and network support through libraries. -Ideal for network applications.	-Requires own run time environment. - Controlled by a commercial organization.
Pascal	- Procedural - Strong type checking - Defined standard	Good teaching language	- Not widely used in industry -Poor environment support
Visual Basic	- Simple procedural language. - Interpreted -Extensive Windows programming support	-Suited to small applications and prototyping. -Some OO concepts in user interface handling. - Widely used	- Performance -Complex data structures cannot be modeled.

Language Features

Language	Features	Strengths	Weaknesses
Visual C++	- C++ programming environment for MS Windows	-Support for windows programming. - User interface design. - Syntax sensitive editors. - Some code generation	- Portability of code.
C #	-Object oriented language derived from C++ and Java. -.NET includes a Common Execution engine and a rich class library. -The classes and data types are common to all of the .NET languages	-In C# Microsoft has taken care of C++ problems such as Memory management, pointers. -It supports garbage collection, automatic memory management and a lot.	- Cannot perform unsafe casts like convert double to a Boolean.

SELECTING LANGUAGES AND TOOLS

Selecting Languages and Tools

- No language suitable for all tasks,
- When choosing a programming language, you have to balance
  - programmer productivity,
  - maintainability, efficiency, portability,
  - tool support,
  - software and hardware interfaces
- Key factor : Programmers Productivity in the project
  - Features of the Programming Language (e.g. Modularity, Type checking)
  - External Factors – Availability of integrated development environment

Development Platform Tools

- An integrated compiler and syntax-directed editing system that allows you to create, edit and compile code.
  - VS-Code
- A language debugging system (integrated to IDE)
- Graphical editing tools, such as tools to edit UML models.
- Testing tools, such as Junit that can automatically run a set of tests on a new version of a program.
- Project support tools that help you organize the code for different development projects.

Integrated Development Environments (IDEs)

- Software development tools are often grouped to create an integrated development environment (IDE).
- An IDE is a set of software tools that supports different aspects of software development, within some common framework and user interface.
- IDEs are created to support development in a specific programming language such as Java.
  - The language IDE
  - General- purpose IDE, with specific language-support tools.

Recap…

- The goal of the coding is to implement_____in the best possible manner.
- Coding activity affects both _____and _____phases.
- Having_____and_____as a clear objective of the coding activity can itself help in producing software that is more maintainable.
- In the coding stage, it transforms the design specification to _____that can be executed on a computer.
- The programmer translates_____into source code of the chosen programming language.
- The language translator converts_____in to executable code in several steps.

Recap…

- The goal of the coding is to implement **the design** in the best possible manner.
- Coding activity affects both **testing** and **maintenance** phases.
- Having **readability** and **understandability** as a clear objective of the coding activity can itself help in producing software that is more maintainable.
- In the coding stage, it transforms the design specification to **source code** that can be executed on a computer.
- The programmer translates **the design** into source code of the chosen programming language.
- The language translator converts **the source code** in to executable code in several steps.

CODING PRACTICES

Coding Challenges

- Software development is teamwork
  - Several members write code in one project
- Software development is concurrent
  - Coding is parallel activity
  - Modular wise code development
  - Code based management using a repository (e.g. github)
- Independent Verification
  - QA team verify the code independently
- Traceability when coding
  - Mapping between design and implementation, vice versa
- Independent Maintenance
  - Separate team will maintain the project in the future

Coding Practices

- Allow code to be written in a more predictable and maintainable fashion.
  - Others should understand the code
- While working code can be written without these techniques it is rare for such code to be maintainable other than by the original author.
  - Clean/Good Code: others will be able to modify the code
    - Correct Errors
    - Add new functionality/features

Coding Practices

- There are three main areas to consider;
  1. **Reliability** -> Is the software fault free?

     Two complementary approaches:
     1. Fault avoidance-develop so that errors are not introduced.
     2. Fault tolerance-develop so that the program continues when faults occur.
  2. **Readability** -> can be code to be easily maintained.
  3. **Reuse** -> can the code be used again.

1. Programming for Reliability

- Reliable code is that which conforms to its specification and continues in operation in all but most extreme circumstances.
- There are two techniques used to increase the reliability of code,
  1. avoidance and
  2. tolerance

1. Programming for Reliability

- Fault avoidance
  - Aims to ensure the code has few errors as possible.
  - How?
- Fault tolerance
  - Aims to produce code that will continue to function in the presence of errors.
  - Fault tolerance includes:
    - Exception Handling
    - Defensive programming
    - Fault recovery

Exception Handling

- An exception is an error or unexpected event. Traditional languages, Pascal, C etc. have no specific support for handling exceptions, newer languages, Ada, Java, have some in built exception handling.

Example:

- Procedure A calls Procedure B
- Procedure B calls Procedure C
- An exception occurs in procedure C
- B cannot continue
- Need to signal exception to A

Exception Handling

With Java exception handling is built into the language

Throw exception

```
class counter {
    int total;
    int number_values;
    // Counter methods
    public int Average() {
     if (this.number_values == 0)
       throw new DivideException (),
     else
     // division code
     }
    }
```

Catch exception

```
class myClass {
 // Class definition
 public void myMethod {
  counter counterOne;
  // Method implementation
  // Get the average
  try {
    counterOne.Average() ;
  } catch ( DivideException e) {
   System.out.println("Division by zero");
  }
 }
}
```

2. Programming for Readability

- One of the major problems with reviewing and maintaining code is that the code is unreadable.
- A number of straight forward techniques are available for improving the readability of the code and therefore reducing review and maintenance effort.
  - Naming conventions
  - Data types
  - Control constructs

Naming conventions

- It is important on projects of more than one person or on the development of software applications which are to be maintained that some form of naming conventions are drawn up at the start of the project.
- The naming conventions would cover such things as program names, function and procedure names, variable names, constant names source file names and so on.
- This is important in reducing the effort required for anyone except the author to read and understand the code.

Naming conventions …..

- Naming conventions help during code reviews/maintenance
  - Others (someone unfamiliar) can easily understand (readability)
  - Others can change the code easily
- Company-wide naming standards for all projects (large organizations)
- Meaningful Names according to naming conventions (standard)
  - for example, the name of a function should reflect the purpose of the function.

Example: Naming conventions in C

1. All constant names are in upper case and begin C, for example C_SPEED_OF_LIGHT
2. All variable names are in mixed case with each word beginning with a capital, variable names may be prefixed with a lower case letter indicating the type of the variable, i for int, l for long etc.,

   for example, `int iCurrentTime;`
3. All variable names should reflect the use of the variable, i.e. it should describe the real world object represented.

   for example;

   ```
   List UnknownWordList; // Good
   List u_list; // Bad
   ```

   Loop indexes can use `i, j` etc.

Example: Naming conventions in C ….

4. Function and procedure names are in mixed case with each word beginning with a capital, function names may be prefixed with a lower case letter indicating the return type of the function, i for int, v for void etc.,

   for example,

```
void vCalculateTotal(List SubTotalList)
{
}
```

5. Function and procedure names must describe the purpose of the function or procedure.
6. Source code file names must reflect the contents of the file for example,

   `CalculateTotal.c` - contains the function CalculateTotal

Data Types

- Use abstract data types to make the code clearer.

Example:

```
Type TrafficLightColour is (red, amber, green);
ColourShowing, NextColour: TrafficLightColour;
```

3. Reuse

- From the 1960s to the 1990s, most new software was developed from scratch, by writing all code in a high-level programming language.
  - The only significant reuse or software was the reuse of functions and objects in programming language libraries.

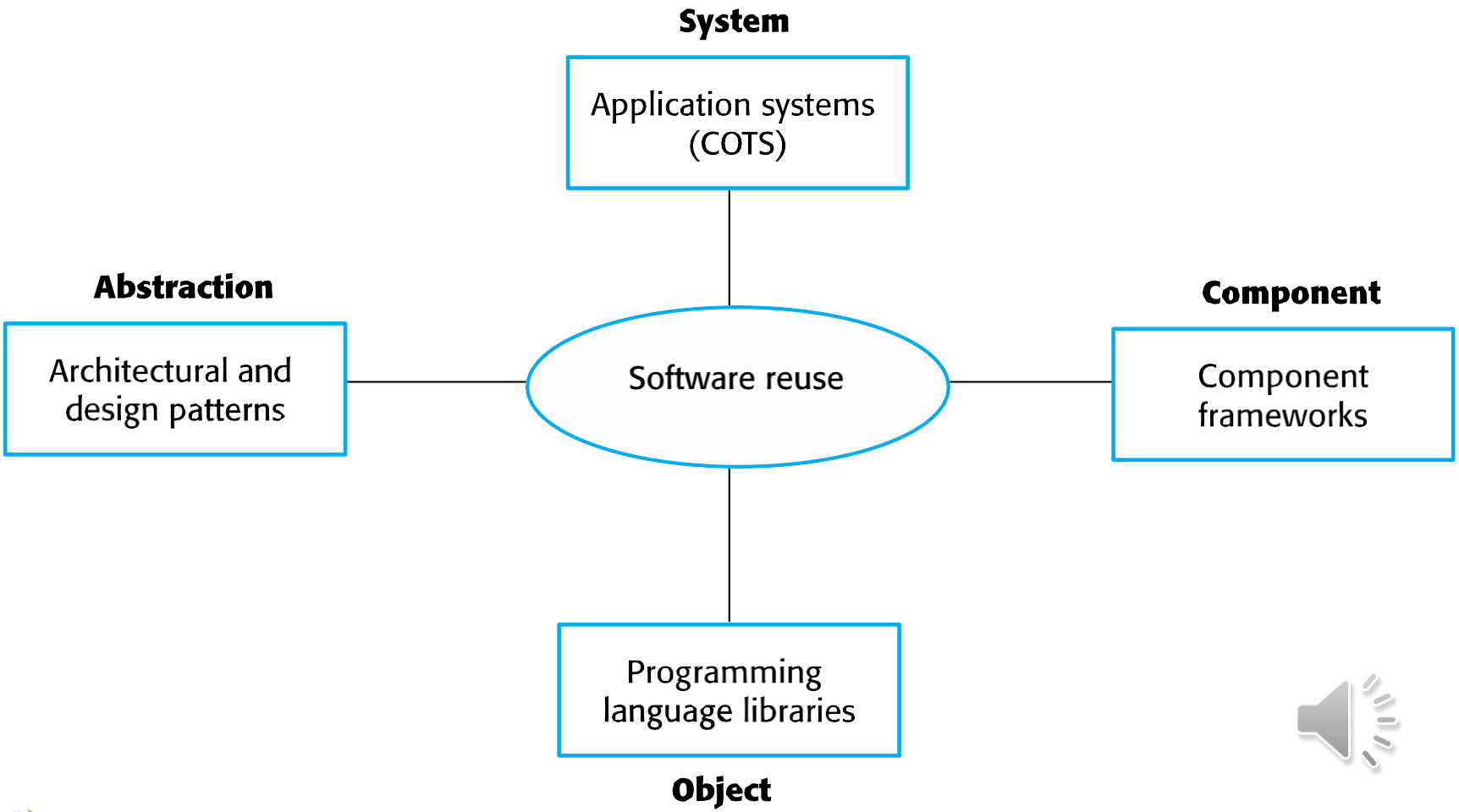
3. Reuse ……

- Costs and time schedule affects negatively development of reusable software
  - Software developed for current project should be further improved for reusability
  - Sometimes not feasible commercial projects
- Reusability of Software – Good feature
  - Popular methodology in software development

Reuse levels

- The abstraction level
  - don’t reuse software directly but use knowledge of successful abstractions in the design of your software. (e.g. design patterns)
- The object level
  - directly reuse objects from a library rather than writing the code yourself.
- The component level
  - Components are collections of objects and object classes that you reuse in application systems.
- The system level
  - reuse entire application systems.

Software reuse in practice



Reuse costs

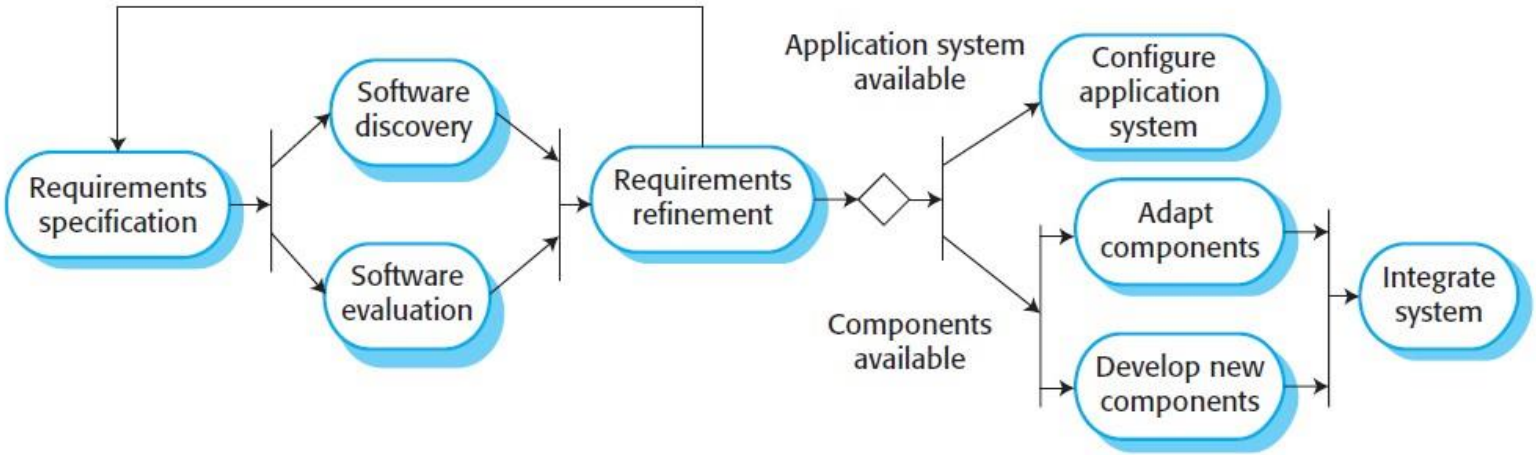
- The costs of the time spent in looking for software to reuse and assessing whether or not it meets your needs.
- Where applicable, the costs of buying the reusable software. For large off-the-shelf systems, these costs can be very high.

Reuse costs

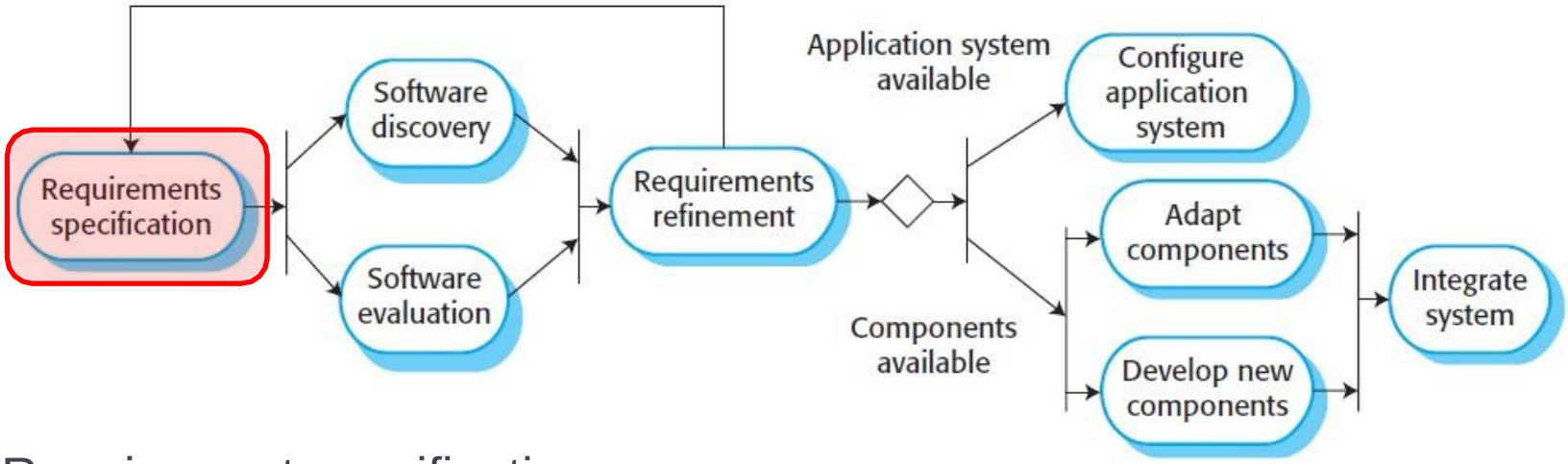
- The costs of adapting and configuring the reusable software components or systems to reflect the requirements of the system that you are developing.
- The costs of integrating reusable software elements with each other (if you are using software from different sources) and with the new code that you have developed.

Reuse-Oriented Software Engineering (*Integration and Configuration)*

Process Stages of Reuse-

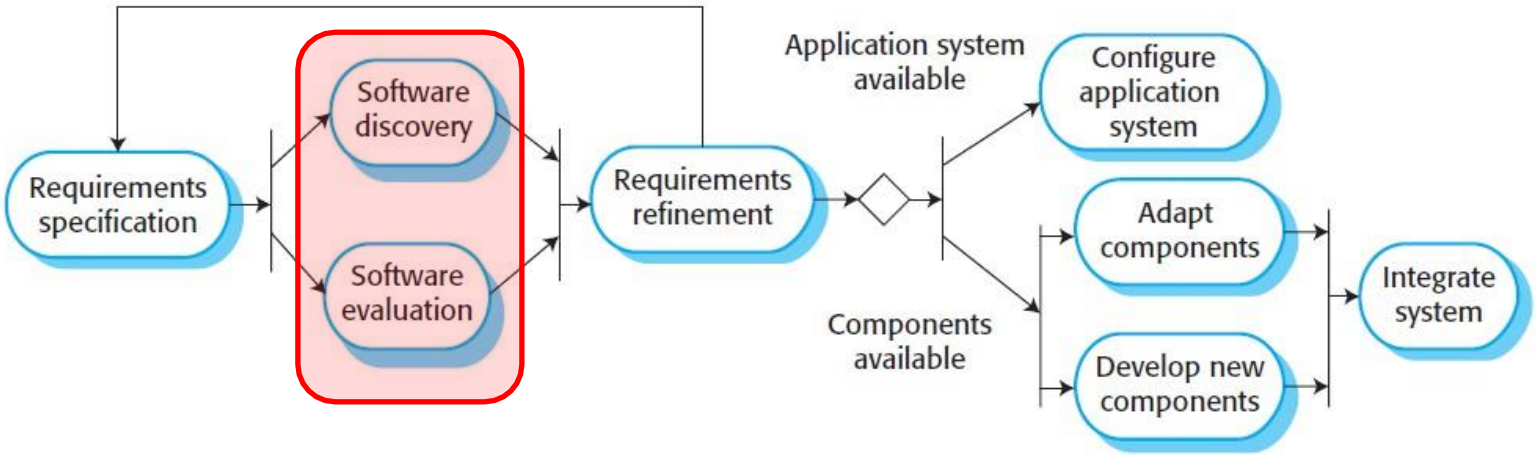


Process Stages of Reuse-



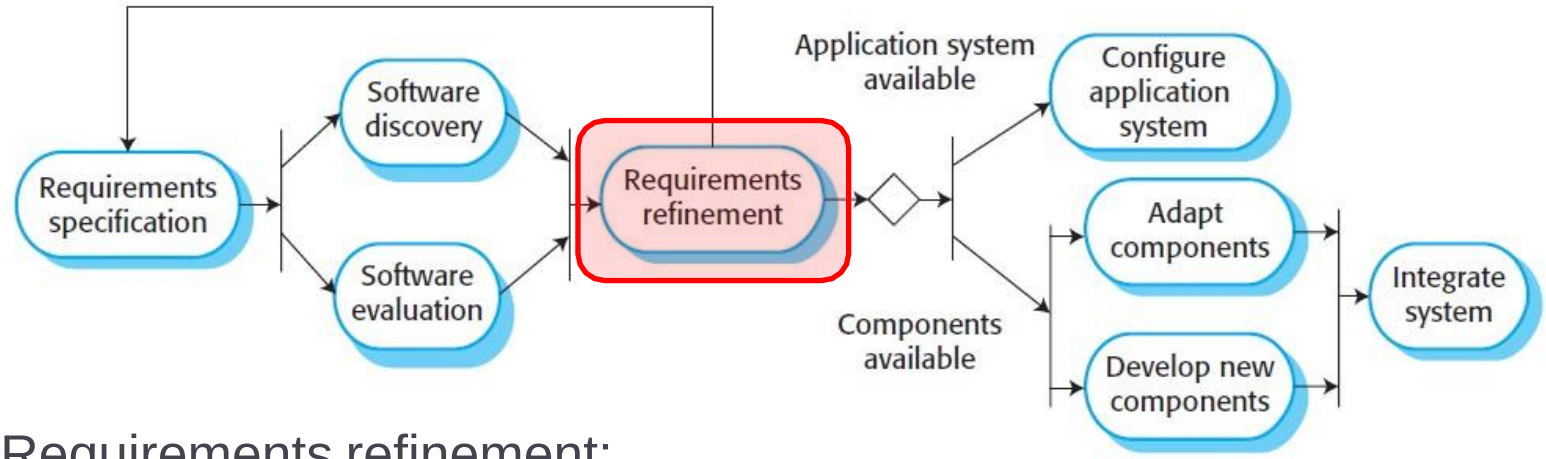
- Requirement specification;
  - initial requirements for the system are proposed.
  - These do not have to be elaborated in detail but should include brief descriptions of essential requirements and desirable system features.

Process Stages of Reuse-



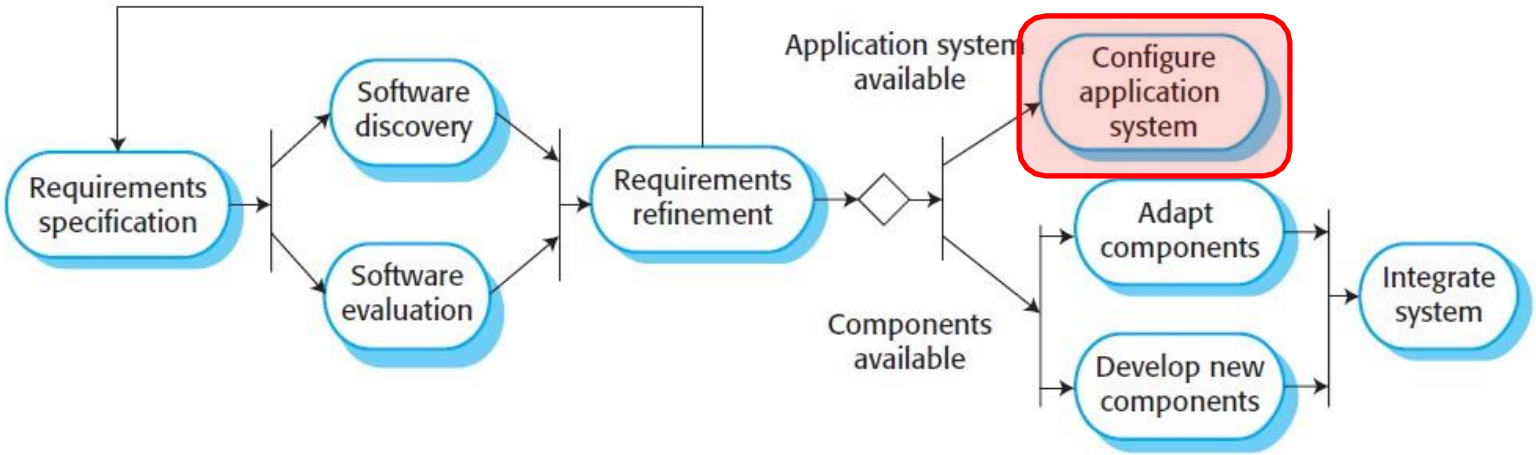
- Software Discovery and Evaluation
  - Given an outline of the software requirements, a search is made for components and systems that provide the functionality required.
  - Candidate components and systems are evaluated to see if they meet the essential requirements and if they are generally suitable for use in the system

Process Stages of Reuse-



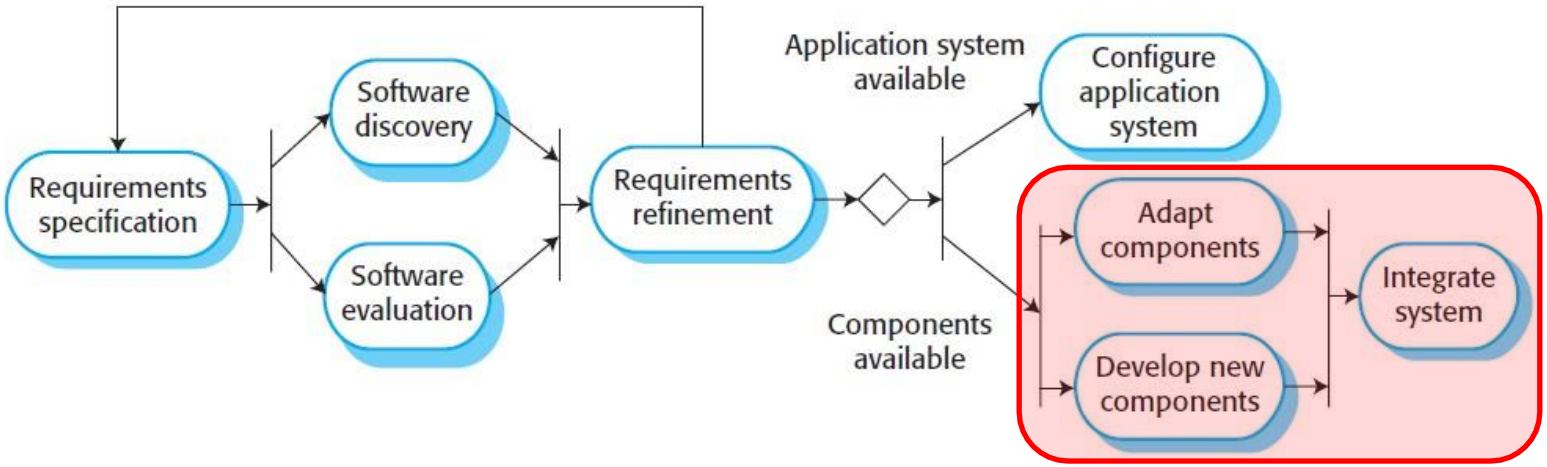
- Requirements refinement:
  - During this stage, the requirements are refined using information about the reusable components and applications that have been discovered.
  - The requirements are modified to reflect the available components, and the system specification is re-defined. Where modifications are impossible, the component analysis activity may be reentered to search for alternative solutions.

Process Stages of Reuse-Based



- Application system configuration:
  - If an off-the-shelf application system that meets the requirements is available, it may then be configured for use to create the new system.

Process Stages of Reuse-



- Component adaptation and integration:
  - If there is no off-the-shelf system, individual reusable components may be modified and new components developed.
  - These are then integrated to create the system.

Technical considerations for selecting PL

- **Environment**
  - Direct support of programming language to operating system
    - *E.g. Visual C++ for Windows*
  - Based on the operating environment, select the suitable programming language
  - System that runs on many environment
    - Java : Code once runs anywhere
- **Language support**
  - Access to libraries when developing programmes
    - Easy and faster development of target applications

Technical considerations .....

- **Performance**
  - Programming language affects the execution speed/performance
    - Interpreted languages are slow (e.g. Basic)
    - Programmes running on the Virtual Machine (e.g. Java)
- **Safety / security**
  - safety critical and secure applications usually require a programming language which supports these requirements.
    - E.g. a nuclear power station control system might be written in *Ada* rather than 'C'.
- **Interfaces to other systems**
  - if the application must interface to other systems then these may restrict the languages which can be used.

Non-technical considerations for selecting PL

More important than technical consideration

e.g. Project Goals, Business Objectives of organization

- **Timescales**
  - Select known PL among developers to develop the system on time

- **Maintenance**
  - Current maintenance team familiarity of PL
    - New PL will force to train the maintenance team (additional cost and time)

Non-technical considerations

- **Other applications**
  - if all other applications built and maintained by the organization are written in COBOL, for example, then there is a very strong argument for building new applications in COBOL.

- **Available tools**
  - what compilers, debuggers are currently available in the workplace? Buying new software to support the development may increase the development costs significantly.

Non-technical considerations

- **Risk**
  - adopting new technology always carries an associated risk, on a project which is critical to the business such risks may be unacceptable.
- **Morale**
  - programmers love to use the latest technology, on a non-critical project it may be beneficial to use an innovative approach to development to retain the interest of the staff, for example Java could be used to develop some small in-house utility. This has the added advantage of bringing new technical knowledge into the organization.

Good Programming Practices

Before you write one line of code, be sure you:

- Understand the problem you're trying to solve
- Understand basic design principles and concepts
- Pick a programming language that meets the needs of the software to be built and the environment in which it will operate
- Select a programming environment that provides tools that will make your work easier
- Create a set of unit tests that will be applied once the component your code is completed

Good Programming Practices

As you begin writing code, be sure you:

- Constrain your algorithms by following structured programming language
- Select data structures that will meet the needs of the design
- Understand the software architecture and create interfaces that are consistent with it
- Keep conditional logic as simple as possible
- Create nested loops in a way that makes them easily testable
- Select meaningful variable names and follow other local coding standards
- Write code that is self-documenting
- Create visual layout that aids understanding

Code Reviews

- A code review is also called a technical review.
- The code review for a module is carried out after the module is successfully compiled and all the syntax errors eliminated.
- These are extremely cost effective strategies for reduction in coding errors in order to produce high quality code.
- In a review, a work product is examined for defects by individuals other than the person who produced it.

Code Reviews

- Code reviews can often find and remove common vulnerabilities such as format string exploits, race conditions, memory leaks and buffer overflows, thereby improving software security.
- Improve the quality of the code being reviewed
- Improve programmers

Code Reviews

- A code review is not a meeting to;
  - negatively criticize someone else's code
  - criticize architecture or design (unless it is an architectural or design review)
  - criticize a colleague
  - determine whether someone is to be removed from a project or not
  - determine whether someone is performing up to standard

Code Reviews

- There are two types of code reviews;
  1. Code walk-throughs
  2. Code inspection

Code Walkthrough

- This an informal code analysis technique. In this technique when a module has been coded, it is compiled and eliminate all syntax errors.
- Some members of the development team are given the code few days before the walkthrough meeting to read and understand the code.
- Each member selects some test cases and simulates the execution of the code by hand.
- The team performing the code walkthrough consists of 3-7 members

Code Walkthrough

- Objectives of Code Walkthrough are;
  - To discover algorithmic and logical errors in the code.
  - To consider alternative implementations
  - To ensure compliance with standards & specifications

Code Inspection

- Errors detected during code inspection;
  - Use of uninitialized variables
  - Jumps in to loops
  - Non terminating loops
  - Incompatible assignments
  - Array indices out of bounds
  - Improper storage allocation and deallocation
  - Mismatches between actual and formal parameters in procedure calls
  - Improper modification of loops

Tips on Finding Errors

- Refine the test cases that produce the error
- Reproduce the error in several different ways
- Use the results of negative tests
- Brainstorm for hypothesis
- Narrow the suspicious region of code
- Be suspicious of routines that have had errors before
- Check code that's changed recently
- Expand the suspicious region of code
- Integrate incrementally
- Check for common errors
- Talk to someone else about the problem
- Take a break when checking for errors

Tips on Fixing Errors

- Understand the problem before you fix it
- Understand the program, not just the problem
- Confirm the error diagnosis
- Relax
- Save the original source code
- Fix the problem, not the symptom
- Make one change at a time
- Check your fix
- Look for similar errors

FREE AND OPEN SOURCE SOFTWARE (FOSS)

Freedom of use

User may have to pay depending on the license

you can modify as per your needs, and share with others without any licensing violation burden

Free Software

- Users have the freedom to
  - Run
  - Copy
  - Distribute
  - Study
  - Change and
  - Improve

    the software

Free Software

- Free software is often confused as “free” in terms of money.
- This is HUGE MISCONCEPTION.

- The “free” in “free software” is “free” as in “free speech”, not “free” as in “free lunch”!

Can we sell a free software?

Many people believe that the spirit of the GNU Project is that you should not charge money for distributing copies of software, or that you should charge as little as possible—just enough to cover the cost. This is a misunderstanding.

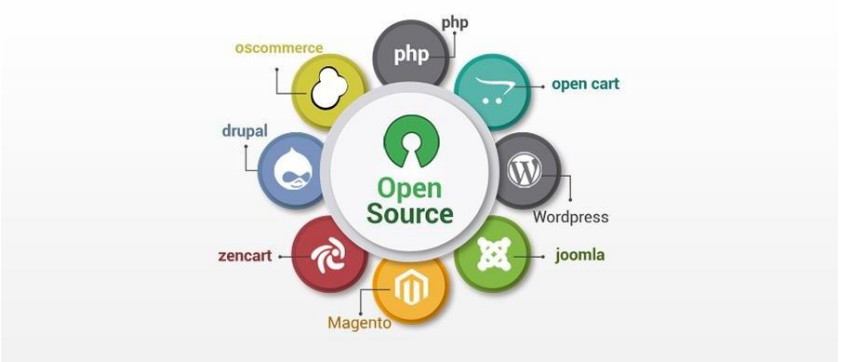
www.gnu.org/

Redistributing free software is a good and legitimate activity; if you do it, you might as well make a profit from it.

Free Software

- A program is “free software”, if the program’s users have following 4 essential freedoms.
  - Freedom 1: The freedom to run the program as you wish, for any purpose.
  - Freedom 2: The freedom to study how the program works, and change it so it does your computing as you wish. Access to the source code is a precondition for this.
  - Freedom 3: The freedom to distribute copies so you can help your neighbor.
  - The freedom to distribute copies of your modified version to others. By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

OPEN SOURCE DEVELOPMENT



Open Source Development

- Open source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process

Open Source Development

- Its roots are in the Free Software Foundation (www.fsf.org), which advocates that source code should not be proprietary but rather should always be available for users to examine and modify as they wish.
- Open source software extended this idea by using the Internet to recruit a much larger population of volunteer developers. Many of them are also users of the code.

Open Source Systems

- The best-known open source product is, of course, the Linux operating system which is widely used as a server system and, increasingly, as a desktop environment.
- Other important open source products are Java, the Apache web server and the mySQL database management system.

https://medium.com/blossom-capital/successful-open-source-business-models-2709e831e38a

Open Source Business

- More and more product companies are using an open source approach to development.
- Their business model is not reliant on selling a software product but on selling support for that product.
- They believe that involving the open source community will allow software to be developed more cheaply, more quickly and will create a community of users for the software.

TensorFlow and its business model

- TensorFlow is an open-source machine learning framework developed by the Google.
- facilitate the development and training of machine learning models, particularly deep learning models.
- provide a comprehensive set of tools and libraries for building and deploying artificial intelligence (AI) applications
  - Users are billed for the resources and services they consume
  - Reduce training cost of AI applications
  - Easing the talent shortage
  - Improve TensorFlow product for free
  - Sell services around it (e.g. lessons)
  - Sell hardware specifically optimized for TensorFlow
  - Consulting and support

TensorFlow

Open Source Licensing

- A fundamental principle of open-source development is that source code should be freely available, this does not mean that anyone can do as they wish with that code.
  - Legally, the developer of the code (either a company or an individual) still owns the code. They can place restrictions on how it is used by including legally binding conditions in an open source software license.
    - Some open source developers believe that if an open source component is used to develop a new system, then that system should also be open source.
    - Others are willing to allow their code to be used without this restriction. The developed systems may be proprietary and sold as closed source systems.

License Models-1

- The GNU General Public License (GPL). This is a so-called ‘reciprocal’ license that means that if you use open source software that is licensed under the GPL license, then you must make that software open source.

License Models -2

- The GNU Lesser General Public License (LGPL) is a variant of the GPL license where you can write components that link to open source code without having to publish the source of these components.

License Models -3

- The Berkley Standard Distribution (BSD) License is a non-reciprocal license, which means you are not obliged to re-publish any changes or modifications made to open source code. You can include the code in proprietary systems that are sold.

License Models - 4

- The Apache software license gives users permission to reuse code for nearly any purpose, including using the code as part of proprietary software.
  - Under the Apache license, users are permitted to modify, distribute, and sublicense the original open source code. Commercial use, warranties, and patent claims are also allowed with the Apache license.
  - The terms and conditions under the license don't place any restrictions on the code, but end users cannot hold the open source contributors liable for any reason.

License Models -5

- Massachusetts Institute of Technology (MIT) License grants the software end user rights such as copying, modifying, merging, distributing, etc. It is notable for what it does not contain, such as clauses for advertising and prohibition of the use of the copyright owner's name for promotional uses.
- The Apache license and MIT license are broadly similar, but there are some key differences
  - Apache 2.0 license text is much more thorough and contains more legal terminology than the MIT license. The MIT license aims to be the most simple and straightforward open source license for developers to distribute their software under.