CSE- 321
Software Engineering

Development & Coding

Lecture Outlines

- ✧ Coding
- ✧ Coding Standards and Guidelines
- ✧ Coding Documentation
- ✧ Software Reuse
- ✧ Application Frameworks
- ✧ Application system reuse

Coding

Coding

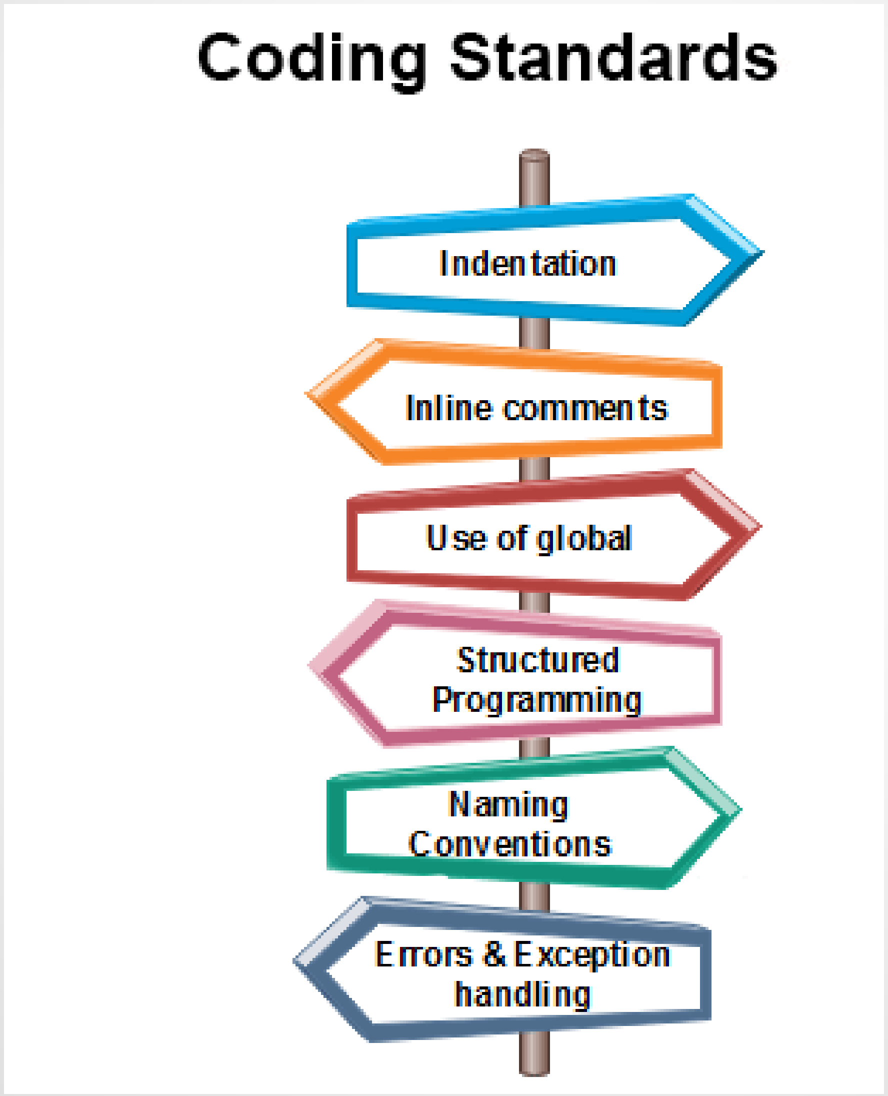
- The coding is the process of transforming **the design of a system into a computer language format**.
- This coding phase of software development is concerned with software **translating design specification into the source code.**
- It is necessary to write source code & internal documentation so that conformance of the code to its specification can be easily verified.

Goals of Coding

- To translate the design of system into a computer language format
- To reduce the cost of later phases
- Making the program more readable

Coding Standards

Coding standards refers to how the developer writes code.



Coding Standards

Indentation: Proper and consistent indentation is essential in producing easy to read and maintainable programs.
Indentation should be used to:

- Emphasize the body of a control structure such as a loop or a select statement.
- Emphasize the body of a conditional statement
- Emphasize a new scope block

Inline comments: Inline comments analyse the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.

Rules for limiting the use of global: These rules file what types of data can be declared global and what cannot.

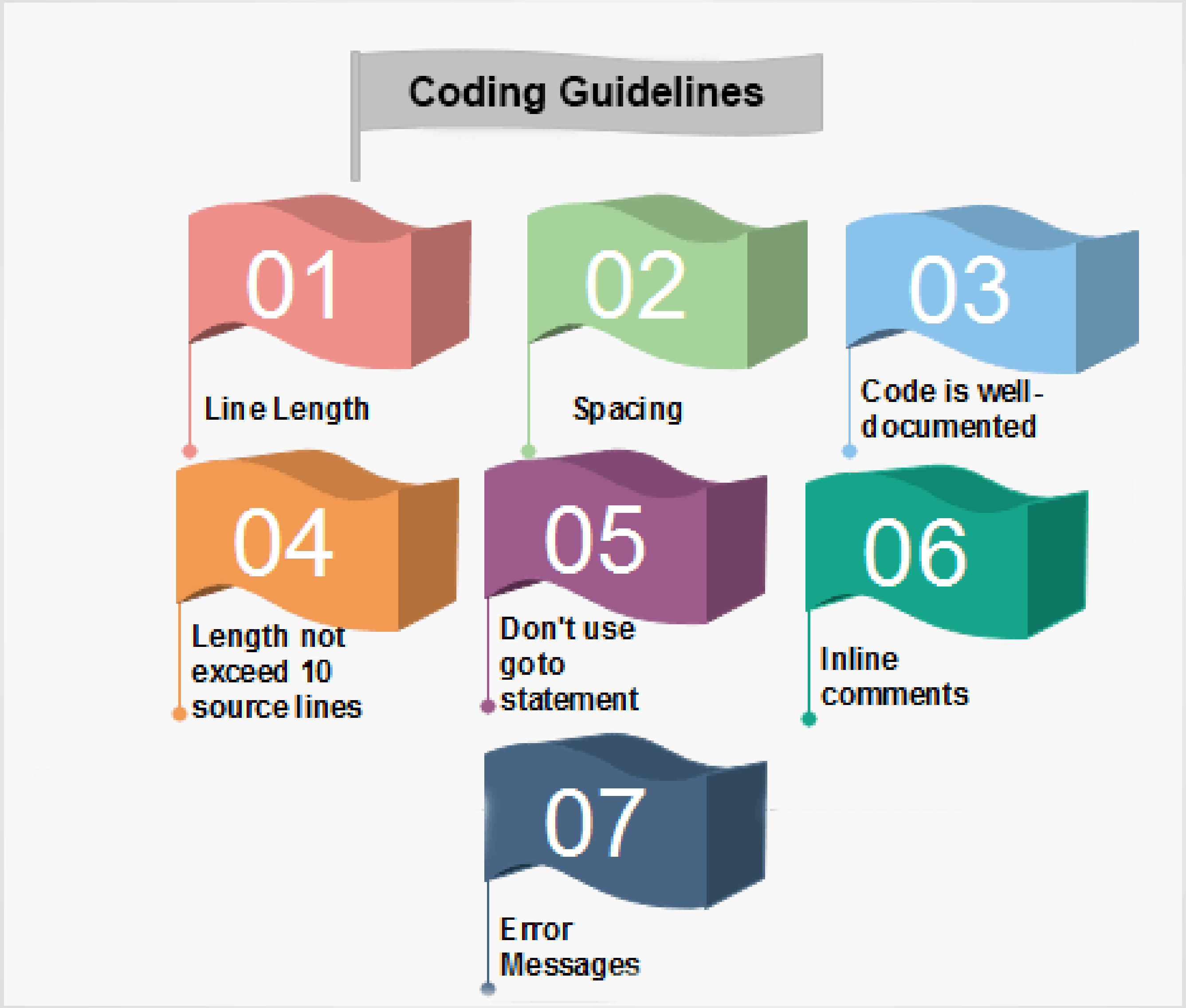
Coding Standards

Structured Programming: Structured (or Modular) Programming methods shall be used.

Naming conventions for global variables, local variables, and constant identifiers: A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

Error return conventions and exception handling system: Different functions in a program report the way error conditions are handled should be standard within an organization.

Coding Guidelines

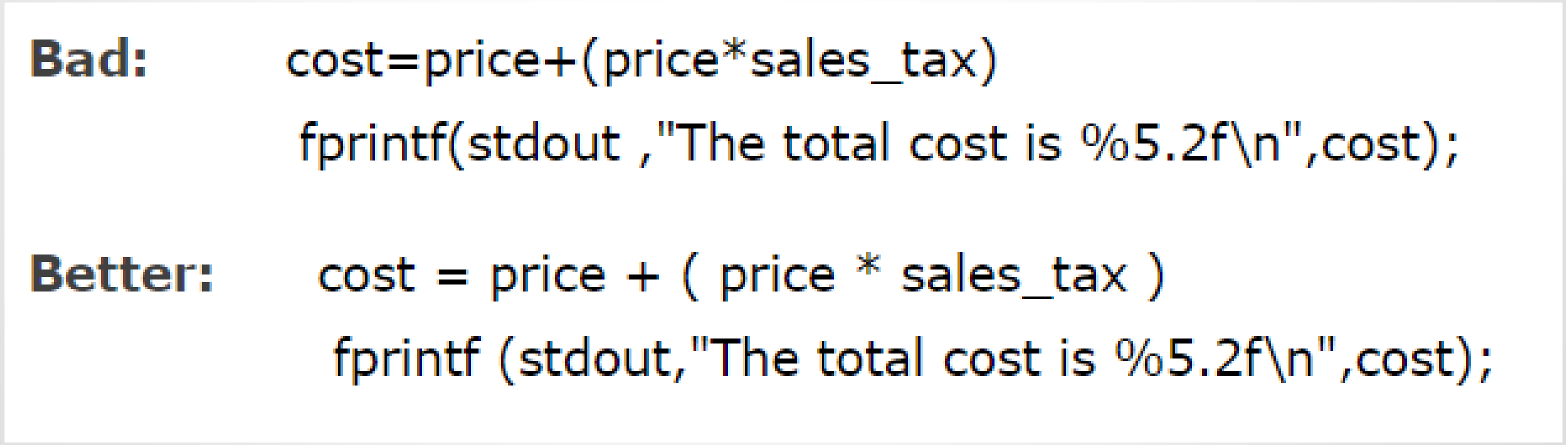


Coding Guidelines

1. **Line Length:** It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

2. Spacing: The appropriate use of spaces within a line of code can improve readability.

Example:



Bad:

```
cost=price+(price*sales_tax)
 fprintf(stdout ,"The total cost is %5.2f\n",cost);
```

Better:

```
cost = price + ( price * sales_tax )
 fprintf (stdout,"The total cost is %5.2f\n",cost);
```

Coding Guidelines

3. The code should be well-documented: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

4. The length of any function should not exceed 10 source lines: A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.

5. Do not use goto statements: Use of goto statements makes a program unstructured and very tough to understand.

6. Inline Comments: Inline comments promote readability.

7. Error Messages: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

Coding Guidelines

Naming: In a program, you are required to name the module, processes, and variable, and so on. Care should be taken that the naming style should not be cryptic and non-representative.

For Example: a = 3.14 * r * r

area_of_circle = 3.14 * radius * radius;

Control Constructs: It is desirable that as much as a possible single entry and single exit constructs used.

Information hiding: The information secure in the data structures should be hidden from the rest of the system where possible. Information hiding can decrease the coupling between modules and make the system more maintainable.

Coding Guidelines

Nesting: Deep nesting of loops and conditions greatly harm the static and dynamic behaviour of a program. It also becomes difficult to understand the program logic, so it is desirable to avoid deep nesting.

User-defined types: Make heavy use of user-defined data types like enum, class, structure, and union. These data types make your program code easy to write and easy to understand.

Module size: The module size should be uniform. The size of the module should not be too big or too small. If the module size is too large, it is not generally functionally cohesive. If the module size is too small, it leads to unnecessary overheads.

Coding Documentation

Code documentation is a manual-cum-guide that helps in understanding and correctly utilizing the software code. The coding standards and naming conventions written in a commonly spoken language in code documentation provide enhanced clarity for the designer.

Moreover, they act as a guide for the software maintenance team (this team focuses on maintaining software by improving and enhancing the software after it has been delivered to the end user) while the software maintenance process is carried out. In this way, code documentation facilitates code reusability.

Coding Documentation

These are some guidelines for creating the documents –

- Documentation should be from the point of view of the reader
- Document should be unambiguous
- There should be no repetition
- Industry standards should be used
- Documents should always be updated
- Any outdated document should be phased out after due recording of the phase out

Advantages of Documentation

These are some of the advantages of providing program documentation –

- Keeps track of all parts of a software or program
- Maintenance is easier
- Programmers other than the developer can understand all aspects of software
- Improves overall quality of the software
- Assists in user training
- Ensures knowledge de-centralization, cutting costs and effort if people leave the system abruptly

Code Documentation Tools

Code Documentation Tools

While writing software code documentation, it is important to consider the code documentation tools required for writing the software code. The software documentation tools conform to standards by generating the required elements automatically with configurable format and style. These tools combine the selected **comment sections** with the software code to generate a usable documentation with the essential level of details in it.

Code Documentation Tools

Documentation Tools	Language Supported
Cocoon	C++
CcDoc	C++
CxRef	C
DOC++	C, C++, Java
JavaDoc	Java
Perceps	C++
RoboDoc	Assembler, C, Perl, LISP, Fortran, Shell scripts, COBOL
DocJet	Java, C, C++, Visual Basic
ObjectManual	C++
Together	Java, C++
Doc-o-matic	C++, C#, ASP.NET, VB.NET, Java, JavaScript, JSP

Software reuse

- In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.
- Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need a design process that is based on systematic software reuse.

- There has been a major switch to reuse-based development over the past 10 years.

- **Software reuse** is the process of implementing or updating software systems using existing software assets.

Reuse-based software engineering

- System reuse
  - Complete systems, which may include several application programs may be reused.
- Application reuse
  - An application may be reused either by incorporating it without change into other or by developing application families.
- Component reuse
  - Components of an application from sub-systems to single objects may be reused.
- Object and function reuse
  - Small-scale software components that implement a single well-defined object or function may be reused.

Benefits of software reuse

- Accelerated development
- Effective use of specialists
- Increased dependability
- Lower development costs
- Reduced process risk
- Creating, maintaining, and using a component library
- Finding, understanding, and adapting reusable components

Problems with software reuse

Creating, maintaining, and using a component library: Populating a reusable component library and ensuring the software developers can use this library can be expensive. Development processes have to be adapted to ensure that the library is used

Finding, understanding, and adapting reusable components: Software components have to be discovered in a library, understood and, sometimes, adapted to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they include a component search as part of their normal development process

Increased maintenance costs: If the source code of a reused software system or component is not available then maintenance costs may be higher because the reused elements of the system may become increasingly incompatible with system changes.

Problems with software reuse

Lack of tool support: Some software tools do not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account. This is particularly true for tools that support embedded systems engineering, less so for object-oriented development tools.

Not-invented-here syndrome: Some software engineers prefer to **rewrite components** because they believe they can improve on them. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.

Approaches that support software reuse

Approach
Application frameworks
Application system integration
Design patterns
Architectural patterns
Aspect-oriented software development
Configurable application systems
Legacy system wrapping
Model-driven engineering
Program generators
Program libraries
Service-oriented systems

Reuse planning factors

Key factors for reuse planning:

- ❖ The development schedule for the software.
- ❖ The expected software lifetime.
- ❖ The background, skills and experience of the development team.
- ❖ The criticality of the software and its non-functional requirements.
- ❖ The application domain.
- ❖ The execution platform for the software.

Application system reuse

Application system reuse

- ✧ An **application system product** is a software system that can be adapted for different customers **without changing the source code** of the system.

- ✧ Application systems have **generic features** and so can be used/reused in different environments.

- ✧ Application system products are adapted by using built in configuration mechanisms that allow the functionality of the system to be tailored to specific customer needs.

▪ For example, in a hospital patient record system, separate input forms and output reports might be defined for different types of patient.

Application system reuse

Benefits of application system reuse:

- ✧ As with other types of reuse, **more rapid deployment** of a reliable system may be possible.
- ✧ It is possible to see what functionality is provided by the applications and so it is easier to judge whether or not they are likely to be suitable.
- ✧ Some development **risks are avoided** by using existing software. However, this approach has its own risks, as I discuss below.
- ✧ Businesses can **focus on their core activity** without having to devote a lot of resources to IT systems development.
- ✧ As operating platforms evolve, technology updates may be simplified as these are the responsibility of the **commercial-off-the-shelf (COTS)** product vendor rather than the customer.

Application system reuse

Problems of application system reuse:

- ✧ Requirements usually have to be adapted to reflect the functionality and mode of operation of the COTS product.
- ✧ The COTS product may be based on assumptions that are practically impossible to change.
- ✧ **Choosing the right COTS system** for an enterprise can be a difficult process, especially as many COTS products are not well documented.
- ✧ There may be a **lack of local expertise** to support systems development.
- ✧ The COTS product vendor controls system support and evolution.

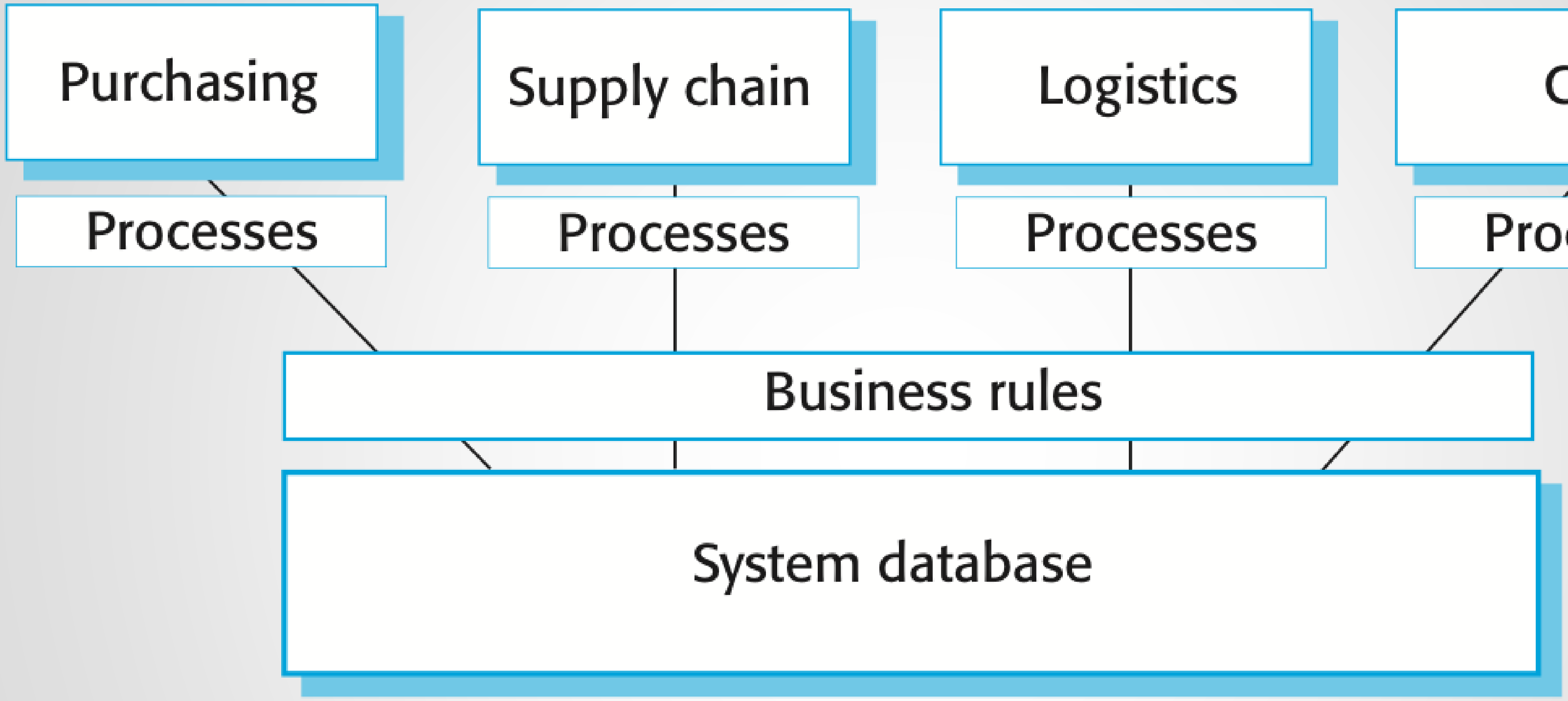
Configurable application systems

- ✧ **Configurable application systems** are generic application systems that may be designed to **support a particular** business type, business activity or, sometimes, a complete business enterprise.
- ✧ For example, an application system may be produced for dentists that handles appointments, dental records, patient recall, etc.
- ✧ Domain-specific systems, such as systems to support a business function (e.g. document management) provide functionality that is likely to be required by a range of potential users.

Enterprise Resource Planning (ERP)

- ✧ An Enterprise Resource Planning (ERP) system is a generic system that supports **common business processes** such as ordering and invoicing, manufacturing, etc.
- ✧ These are very widely used in large companies - they represent probably the most common form of software reuse.
- ✧ The generic core is adapted by including modules and by incorporating knowledge of business processes and rules.
- ✧ A number of modules to support different business functions. A defined set of business processes, associated with each module, which relate to activities in that module.
- ✧ A common database that maintains information about all related business functions. A set of business rules that apply to all data in the database.

Enterprise Resource Planning (ERP)



ERP system architecture

Key elements of an ERP system architecture:

- ✧ A number of modules to **support different business functi**ons.
- ✧ A defined set of business processes, associated with each module, which relate to activities in that module.
- ✧ A **common database** that maintains information about all related business functions.
- ✧ A **set of business rules** that apply to all data in the database.

ERP system configuration

An ERP system configuration usually involves:

- ✧ Selecting the required functionality from the system.
- ✧ Establishing a data model that defines how the organization's data will be structured in the system database.
- ✧ Defining business rules that apply to that data.
- ✧ Defining the expected interactions with external systems.
- ✧ Designing the input forms and the output reports generated by the system.
- ✧ Designing new business processes that conform to the underlying process model supported by the system.
- ✧ Setting parameters that define how the system is deployed on its underlying platform.

Question

The reuse of software raises a number of copyright and intellectual property issues. If a customer pays a software contractor to develop a system.

- Who has the right to reuse the developed code?
- Does the software contractor have the right to use that code as a basis for a **generic component**?
- What payment mechanisms might be used to reimburse providers of reusable components?
- Discuss these issues and other ethical issues associated with the reuse of software.

Solution:
https://paigepecksite.wordpress.com/2016/09/29/hw15-chapter-15/

Question

Because the **customer** paid the software contractor specifically to develop this specific system, the customer should have the **rights to reuse** the developed code.
When a **software developer** is working for a company designing code, it is my understanding that the code they write is owned by the company, unless it is an open source platform. With that being said, With that being said, a software developer has learned coding techniques and how to do algorithms in a proficient manner, so **the generic code itself could be reused** as long as they aren’t copy and pasting it from the actual developed code.

They would have to rewrite it as a whole new system and be wary not to be designing something that would be considered in direct competition of the software they developed for the previous company. This leaves a very gray line that could easily be misused, but I don’t see how a software developer could not reuse some components especially if they are in the same field but with a different company.

Question

While there are things such as drawings, poems, beats to a song that can be copyrighted because they can be unique, software is a different entity entirely.

As for a payment mechanism for the reimbursement of reusable components, this is a difficult one. Again, it is not like a software developer should just stop using an algorithm because they created it for a different company when it can help them accomplish what they need, but if it is a component that can be "sold", then perhaps it could be sold like assets are sold for 3D modeling online. It is a very gray area that is difficult to pinpoint what should/could be done, but it is of my opinion that a software developer should be allowed to better themselves without worrying about copyright and intellectual property issues.

Which means in the grand scheme of things, it should not be the customer who is allowed to sell the reusable components as they paid the contractor to write it, not to create this system for only them alone which should be a higher cost if such a thing were to happen.