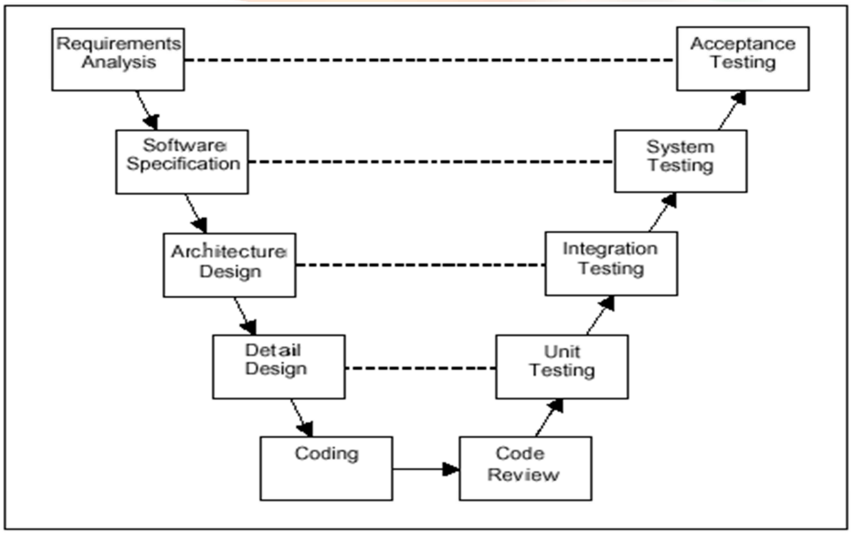
Coding Process

Instructor << >>

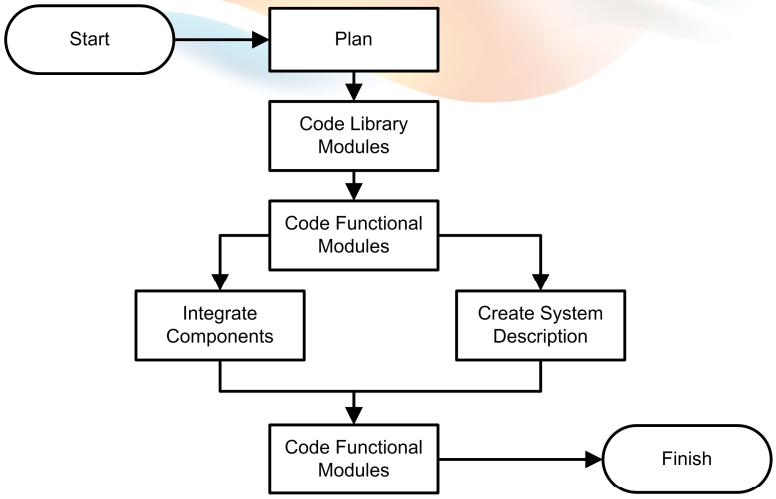
Agenda

- Coding Process
- Coding Convention
- Code Review Process
- Common Defects & Practices

Coding Process
Where the Coding is?



Coding Process
Coding Workflow



Coding Process
Code Planning

- Purpose: To plan and prepare for coding
- Steps:
  - Study design documents
  - Define and prepare resources and infrastructure for coding, unit test and integration, if necessary.
  - Create coding plan including targets, scope, required deliverables and acceptance criteria,
  - task and schedule, responsibilities
  - Review and obtain agreement on coding plan
  - Develop/customize coding convention
  - Review & conduct training on coding convention
  - Verify tools support for coding (if any)

Coding Process
Coding Library Modules

- Purpose: To build, construct and/or develop library modules
- Steps:
  - Create detail design for library modules
  - Code library modules
  - Review code of library modules
  - Fix defects of library modules
  - Summarize related documents

Coding Process
Coding Functional Modules

- Purpose: To build, construct and/or develop functional modules
- Steps:
  - Create detail design for modules and program units, if required in design documents
  - Code modules and program units
  - Review code
  - Fix defects for modules and program units
  - Summarize and submit result to Team Lead

Coding Process
Integrate Software Modules

- Purpose: assemble the software package from the software modules, ensure that the software package, as integrated and functions properly
- Steps:
  - Create integration plan (if needed)
  - Integrate modules
  - Evaluate integration results, conduct cause analysis, raise change request (if needed)
  - Review and approve results of integration

Coding Process
Create System Description

- Purpose: To develop System Description /User Manual documents that support in software operation
- Steps:
  - Make overview on system
  - Describe sub-systems and main functions including system structure scheme, flow charts, system interfaces, data flows
  - Describe system requirements including support data, memory capability, CPU and I/O requirements, storage capability, data for internal and external interfaces
  - Describe software structure including library of source codes (for system, sub-systems, objects) library of executive and supporting programs
  - Develop User Manual
  - Review and approve System Description/User Manual

Coding Process
Deliver & Summarize

- Purpose: To deliver software package
- Steps:
  - Review, do final inspection and summarize software products including documents
  - Deliver to test team
  - Create coding summary report
  - Maintain documents, records

Coding Convention Introduction

- Be specific to each programming language
- Recommend programming style, practices, and methods for each aspect of a piece program
- Common conventions may cover the following areas:
  - file organization,
  - naming conventions
  - indentation, white space,
  - comments, declarations, statements,
  - programming practices, principles, rules of thumb,
  - Etc.

Coding Convention Importance

Code conventions are important to programmers for a number of reasons:

- 80% lifetime software cost is for maintenance
- People maintain the software may be changed
- Following coding convention strictly helps:
  - Improve the readability of the software
  - Allowing engineers to understand new code more quickly and thoroughly

Coding Convention
Some Common Standards 1/2

- Tab and Indent
  - 4 spaces should be used as the unit of indentation
  - Tab characters should be avoided
- Line Length: avoid lines longer than 80 or 120 characters
- Wrapping Lines: When an expression will not fit on a single line, break it according to below principles:
  - Break after a comma.
  - Break after a logical operator.
  - Break before an operator.
  - Prefer higher-level breaks to lower-level breaks
  - …
- Comments: beginning, block, single-line, trailing, …
- Number of declarations per line: same types, different types,..

Coding Convention
Some Common Standards 2/2

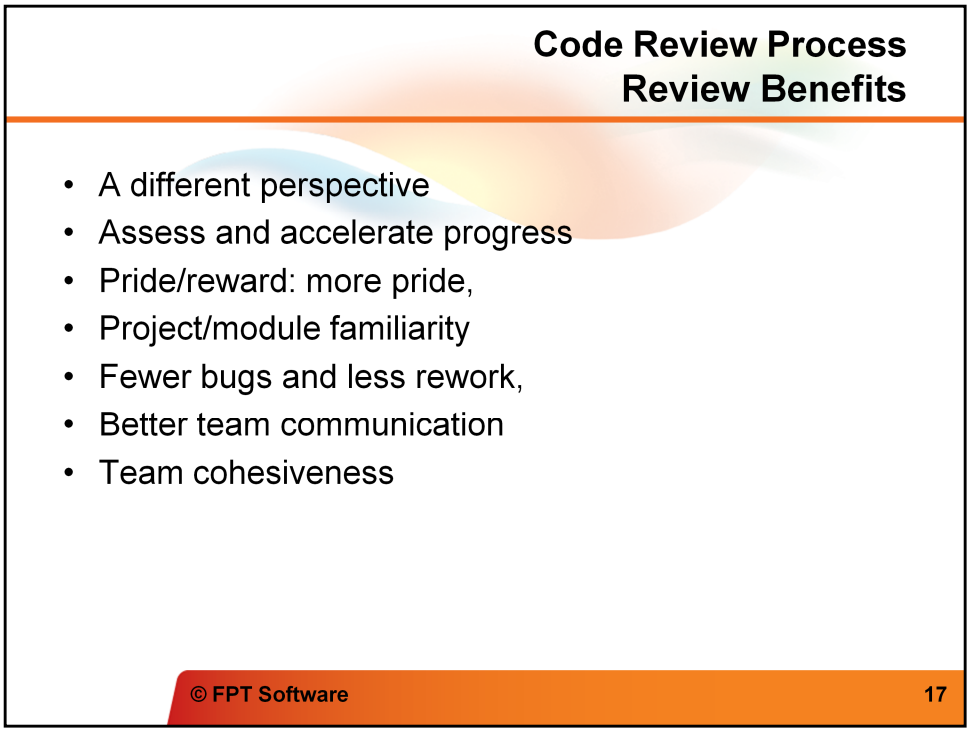
- Blank Lines improve readability by setting off sections of code that are logically related
  - Two blank lines should always be used:
    - Between sections of a source file
    - Between class and interface definitions
  - One blank line should always be used:
    - Between methods
    - Between the local variables in a method and its first statement
    - Before a block or single-line comment
    - Between logical sections inside a method
- Blank spaces should be used in the following circumstances
  - A keyword followed by a parenthesis should be separated by a space
  - A blank space should appear after commas in argument lists
  - All binary operators except . should be separated from their operands by spaces

Coding Convention
Naming Conventions

- General naming rules:
  - Should be functionally meaningful, & indicate identifier's purpose
  - Use terminology applicable to the domain
  - Identifiers must be as short as possible (<=20 characters)
  - Avoid names that are similar or differ only in case
  - Abbreviations in names should be avoided, etc.
- Use a noun or noun phrase to name a class or code module
- Variables names must start with lowercase
- Constants: named in uppercase letters, might have underscore
- Method names must start with lowercase letter, usually use "active verb" as the first word of method name
- Instance /object names follow rules of variable names

Code Review Process
Code Review

- A thorough technical & logical line-by-line review of a code module (program, subroutine, method, ..)
- In a code review, the team would examine a sample of code and fixes any defects in it
  - Codes is inspected to identify possible improvements and ensure that business requirements are met.
  - For example, it could be made more readable or its performance could be improved

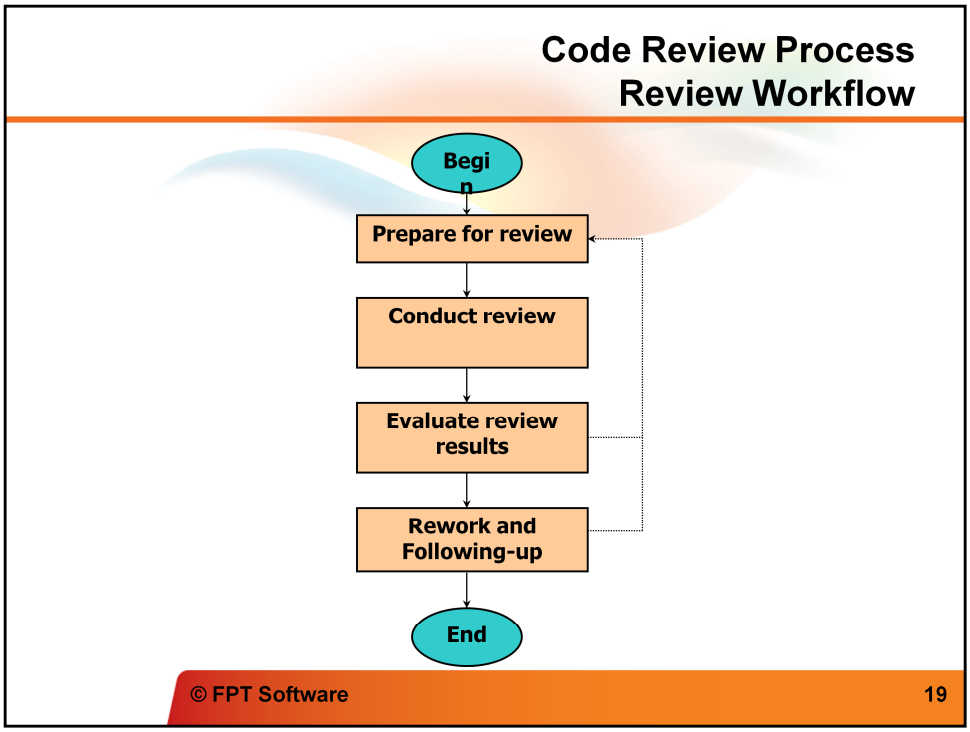


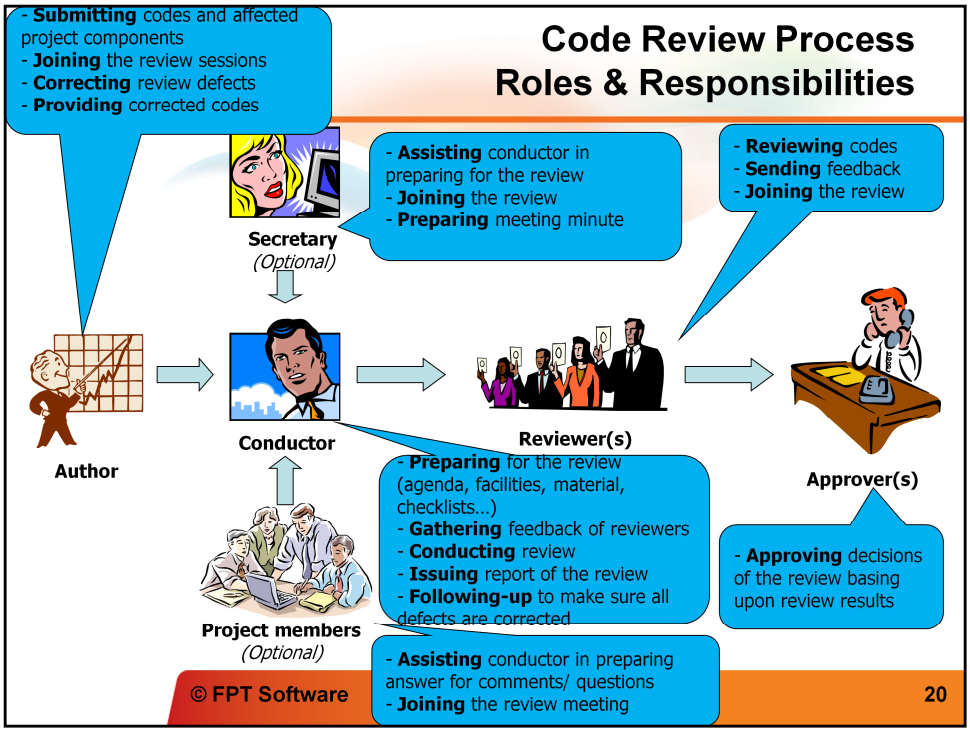
- ✓ A different perspective. “Another set of eyes” adds objectivity. Similar to the reason for separating your coding and testing teams, peer reviews provide the distance needed to recognize problems.
- ✓ The ability to assess and accelerate progress. Is the team producing the needed output at the needed rate?
- ✓ Pride/reward. Recognition of coding prowess is a significant reward for many programmers.
- ✓ Project/module familiarity. Everyone involved in the review becomes more familiar with the project and the module.
- ✓ Less rework. Do it right the first time. Changes cost more later in the life cycle. The peer review process catches many errors *before* they go to production.
- ✓ Fewer bugs. It’s better to discover your own problems than to have someone (like a user) point them out to you. For the same reason, you may find that many bugs will be eliminated before the code comes to be reviewed by peers.
- ✓ Improved communication. More opportunities for interaction tend to lead the team toward improved communication.
- ✓ Team cohesiveness. Working together helps draw team members closer. It also provides a brief respite from the isolation that coding often brings.

Code Review Process
Review Candidates

- A portion of the software that only one person has the expertise to maintain
- Code that implements a highly abstract or tricky algorithm
- An object, library or API that is particularly difficult to work with
- Code written by someone who is inexperienced or has not written that kind of code before, or written in an unfamiliar language
- Code which employs a new programming technique
- An area of the code that will be especially catastrophic if there are defects

Catastrophic: the tham


Code Review Process
Review Workflow
Begi
n
Prepare for review
Conduct review
Evaluate review
results
Rework and
Following-up
End
© FPT Software
19


Code Review Process
Roles & Responsibilities
- Submitting codes and affected project components
- Joining the review sessions
- Correcting review defects
- Providing corrected codes
Secretary
(Optional)
- Assisting conductor in preparing for the review
- Joining the review
- Preparing meeting minute
- Reviewing codes
- Sending feedback
- Joining the review
Author
Conductor
Reviewer(s)
Approver(s)
- Preparing for the review (agenda, facilities, material, checklists…)
- Gathering feedback of reviewers
- Conducting review
- Issuing report of the review
- Following-up to make sure all defects are corrected
- Approving decisions of the review basing upon review results
Project members
(Optional)
- Assisting conductor in preparing answer for comments/ questions
- Joining the review meeting
© FPT Software
20

Code Review Process
Reviewing Inputs

- Statement of objectives of the review
- Source codes to be reviewed
  - ✓ Must be completed, tested by developer
  - ✓ Ready for the next step in the development life cycle (developer's view)
- Other affected project components: documentation, test cases, a project schedule, or requirements changes, etc.
- Checklists to be used (optional)

Code Review Process
Prepare for review 1/2

- Preparing agenda and facilities for the review
- Contacting with PM/QA to identify list of reviewers
- Sending agenda along with all inputs to reviewers
- If a meeting is necessary, gathering all comments /questions of reviewers and prepare answers/solutions for each one

Code Review Process
Prepare for review 2/2

- Preparing notes:
  - Reviews are conducted as needed, usually based on the rate of code output.
  - The frequency of individual participation in a peer review depends primarily on the size of the programming team.
    - A team of 3 developers might include all three in every review.
    - Larger teams might be able to rotate participation based on experience, skill level, subject matter familiarity, ...
  - The review should include the programmer, two reviewers, a recorder, and a leader.
  - Other considerations for the size of the review team might be the scope of the project, workload, or training needs.

Code Review Process
Conduct Review 1/2

- Performing online review. Some questions reviewers might bring up:
  - ✓How does this module actually satisfy the stated requirement?
  - ✓How does the output affect the previously base-lined interface documentation?
  - ✓Wouldn't a *case* statement work here instead of a nested *if-then-else* structure?
  - ✓Etc.
- Logging and sending to conductor all defects found and comments/questions

Code Review Process
Conduct Review 2/2

- Conducting the review meeting (if any).
  - The leader opens with a short discussion of the goal of the meeting and lays out any ground rules.
  - Prioritising all defects and comments/ questions during the meeting
  - The developer goes through and explains his/her code.
  - The reviewers raise, discuss on the issues, comments, questions
  - The developer addresses those issues and explains the logic, problems, and choices that resulted in this code
- Logging all new defects or unsolved questions found during the review meeting

Code Review Process
Evaluate review & Follow up

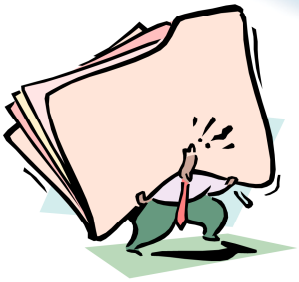
Evaluating review results

- Preparing and issuing Review report to all attendees
- Making approval or reject basing upon the review's results

Rework and Follow-up

- Fix code-review defects
- Monitoring to make sure all defects are closed as planned
- If necessary, preparing for the new review again.

Code Review Process
Review Outputs



- Review Report: reviewer list, defect list, statistic and analysis…
- Filled-up checklists
- Minute of meeting (if any)
- Approval or Reject of approver

Code Review Process
Self- Code Review 1/3

- What: developer to do self-code review while he/she do the coding, it is to make sure that:
  - Requirement logics are implemented correctly
  - No coding conventions or common defects existed
  - General programming practices are applied
- How:
  - Use code review tools
  - Use team-defined code review checklist

Code Review Process
Self- Code Review 2/3

Code Review Tools

- http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

➢ .NET
  - FxCop http://msdn.microsoft.com/en-us/library/bb429476%28v=vs.80%29.aspx
  - Resharper http://www.jetbrains.com/resharper/
  - StyleCop http://stylecop.codeplex.com/

➢ JAVA
  - ➢ CheckStyle (http://checkstyle.sourceforge.net/)

➢ C,C++
  - ➢ CPPCheck http://sourceforge.net/apps/mediawiki/cppcheck/

Code Review Process
Self- Code Review 3/3

Code Review checklist

- This is a team-defined coding checklist.
- Project developers are required to self review their codes following defined checklist items, filled the code review checklist as reviewing results
- Main checklist items
  - General coding conventions
  - Code module, class commenting
  - Source code details: modulation, code structure, loop, naming conventions, comments, etc.

Common Defects & Practices
Hard code constants

- Issue with giving a fixed value in codes, for example:

```
dgrView.PageSize = 10
strErr = "Error message here";
```

The problem occurs when you should change these values multiple times!!!

- Preventive Action: define constants in the common constant module or in a configure files

Common Defects & Practices
Array Index Start from 0

- Issue with below C-Language codes?

```
int i, a[10];
for (i=1; i<=10; i++) a[i] = 0;
```

This made the loop into an infinite loop!!!

- Cause: A C array with n elements does not have an element with a subscript of n, as the elements are numbered from 0 through n-1.
- Preventive: programmers coming from other languages must be especially careful when using arrays.

Common Defects & Practices
The Dangling else Problem

- Issue with below C-Language codes?

```
if (x == 0)
    if (y == 0) error();
else {
    z = x + y;
    f (&z);
}
```

Confused on the else using!!!

- Cause: else is always associated with the closest unmatched if.
- Preventive: use appropriated braces ({)

Common Defects & Practices
Null Pointer Exception

- Issue: the developer got Null-Pointer-Exception run-time error, while he/she did not detect that when compiling the codes

```
pPointer->member = 1;
strReturn = objDoc.SelectNodes(strName);
```

- Cause: the developer does not check null or think about null object before accessing object's value.
- Preventive: Should check null before accessing object or pointer before using its member

```
If ( pPointer != NULL ) pPointer->member = 1;
If (objDoc != NULL)
   strReturn = objDoc.SelectNodes(strName);
```

Common Defects & Practices
Detect Common Defects Sample

```
public bool IsValidLogin(string userName, string password)       {
        SqlConnection con = null;
        SqlCommand cmd = null;
         bool result = false;
        try {
           con = new SqlConnection(DB_CONNECTION);
           con.Open();
           string cmdtext = string.Format("SELECT * FROM [Users] WHERE [Account]='{0}' AND
                                                                [Password]='{1}' ", userName,
password);
           cmd = new SqlCommand(cmdtext);
           cmd.Connection = con;
           cmd.CommandType = CommandType.Text;
           result= cmd.ExecuteReader().HasRows;
           cmd.Dispose();
           con.Dispose();
           return result;
        }
        catch (SqlException) {
            return false;
        }
}
```

Lack of checking for null value(1)

SQL Injection (1)

Hard code !!(1)

SQL Performance Issue !!(1)

Memory leak !! (2)

Common Defects & Practices
Programming Practices 1

- Issue with variables or create objects in Loop?

```
for (int i=0; i<dt.Rows.Count-1; i++)
{
  string strName;
  strName = dt.Rows[i]["Name"].ToString();
  //do something here
}
```

Impact to the application performance!!!

- Cause: memory is allocated repeatedly.
- Preventive:
  - Variables should be declared before the loop statement or inside for() statement
  - Determine objects before loop statement

Common Defects & Practices
Programming Practices 2

- Code redundant issues:
  - Create new Object while we can reuse the object in previous command:
    ```
    BeanXXX bean = new BeanXXX();
    bean = objectYYY.getBeanXXX();
    ```
  - Variables are declared in based class but it is not used
  - Un-used methods/functions are existing in the application
  - Break a complex method/function to more simple methods / functions with only one or two lines of code, and could not be re-use
- Preventive actions:
  - Should verify that the current design is possible and is the best by coding sample
  - Re-check unnecessary code to remove in coding review
  - Supervise and assign person to review code carefully before coding
  - Supervise strictly changing source code from team daily

Common Defects & Practices
Programming Practices 3

- Avoid using an object to access a *static* variable or method. Use a class name instead.

```
classMethod();                  //OK
AClass.classMethod();           //OK
anObject.classMethod();         //AVOID!
```

- Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.
- Avoid assigning several variables to the same value in a single statement.

```
fooBar.fChar = barFoo.lchar = 'c'; // AVOID!
```

Common Defects & Practices
Programming Practices 4

- Do not use the assignment operator in a place

```
if (c++ = d++) { // AVOID!
   ...
}
```

should be written as:

```
if ((c++ = d++) != 0) {
   ...
}
```

- Do not use embedded assignments in an attempt to improve run-time performance.

```
d = (a = b + c) + r;
```

Common Defects & Practices
Programming Practices 5

- File operations: file read operations must be restricted to a minimum
- Clear content of big structure after use: always clear() the content of Collection/Map objects after use
- Be economical when creating new objects
- In program language that has no garbage collector (i.e C, C++): free allocated memory after use:

```
{
    double* A = malloc(sizeof(double)*M*N);
    for(int i = 0; i < M*N; i++){
        A[i] = i;
    }
}
```

memory leak: forgot to call `free(A);`
common problem in C, C++

Common Defects & Practices
Programming Practices 6

- Use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems

```
if (a == b && c == d) // AVOID!
if ((a == b) && (c == d)) // RIGHT
```

- Try to make the structure of your program match the intent, for example:

```
if (booleanExpression) {
   return true;
} else {
   return false;
}
```

should instead be written as

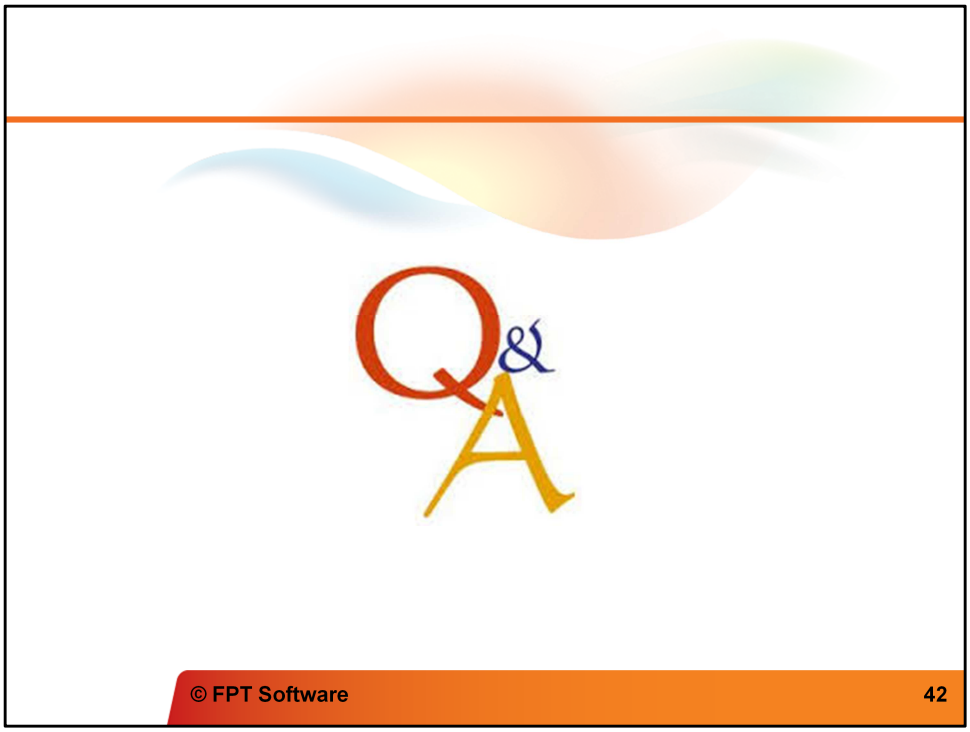
```
return booleanExpression;
```


Similarly,

```
if (condition) {
var = x;
} else {
var = y;
}
```

should be written as

```
var = (condition) ? x : y;
```


Q&A
© FPT Software
42