

1 Overview	2 Life cycle components	3 Infrastructure components	4 Management components	5 Standards and Organizing
6 Static tesing	7 Dynamic testing	8 Test management	9 Tools	

Static techniques

Learning objectives

- Describe the objective of static analysis and compare it to dynamic testing
- Describe the phases, roles and responsibilities of a typical formal review
- List typical benefits of static analysis
- List typical code and design defects that may be identified by static analysis tools

References

- Dorothy Grahamet, Erik van Veenendaal, Isabel Evans, Rex Black. *Foundations of software testing: ISTQB Certification*
- FSOF course

1	2	3	4	5
6	7	8	9	

Contents

1. Static testing techniques
2. Reviews and the test process
 3. Review process
 4. Static analysis by tools
 5. Self- code review

1. Static testing techniques

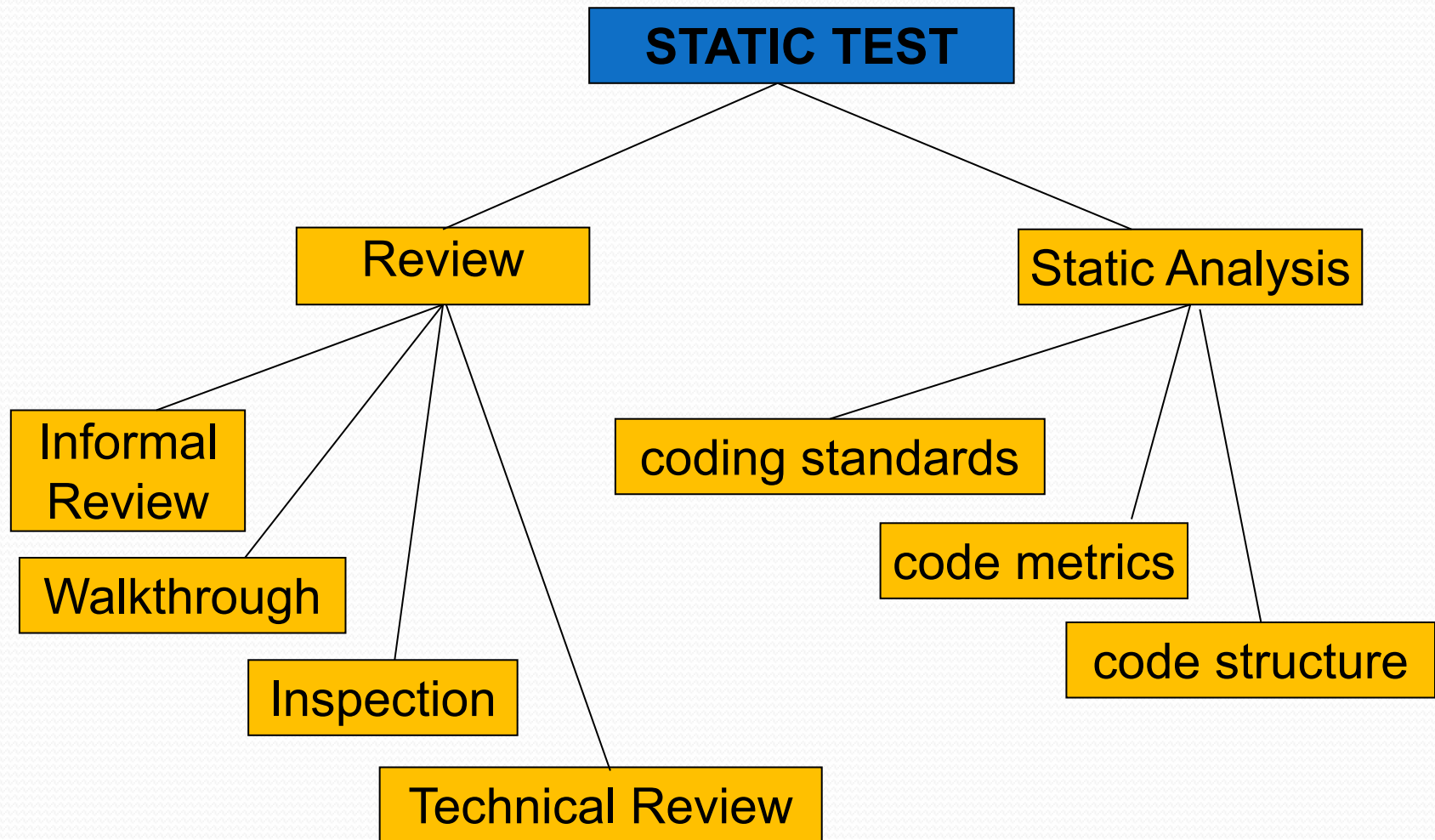
- Static testing techniques are those techniques that test a component or system at a specification or implementation level **without execution of the software**
- Types of defects that are easier to find during static testing are: deviations from standards, missing requirements, design defects, non-maintainable code, inconsistent interface specifications,...

1. Static testing techniques (cont.)

- Objectives

- to identify errors in any phase of SDLC as early as possible
- to verify that the components of software are in conformance with its requirements
- to provide information for project monitoring
- to improve the software quality and increase productivity

1. Static testing techniques (cont.)



1	2	3	4	5
6	7	8	9	

Contents

1. Static testing techniques
2. Reviews and the test process
3. Review process
4. Static analysis by tools
5. Self- code review

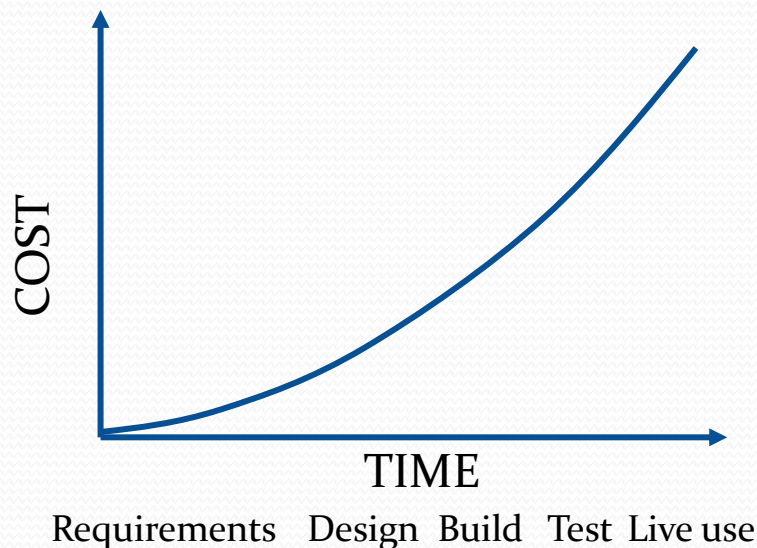
2. Reviews and the test process

- A review is a process or meeting for examination of a document by one or more people
- The objectives of reviews:
 - finding defects
 - informational, communicational and educational

2. Reviews and the test process

Benefits of reviews

- Early feedback on quality issues
- A cheap improvement of the quality of software products
- Development productivity improvement
- Reduced testing time
- A means of customer/user communication



2. Reviews and the test process

What can be reviewed?

- Anything written down
 - Requirement specifications
 - Design document
 - Code
 - Schedules
 - Test plans, test cases, defect reports
 - User manuals, procedures, training material
 - etc.

2. Reviews and the test process

Review inputs

- Statement of **objectives** of the review
- **Material** to be reviewed
- **Checklists** to be used
- **Report** templates

2. Reviews and the test process

Review deliverables

- Edits in review product
- Change requests for source documents
- Process improvement suggestions
 - to the review (Inspection process)
 - to the development process which produced the product just reviewed
- Metrics

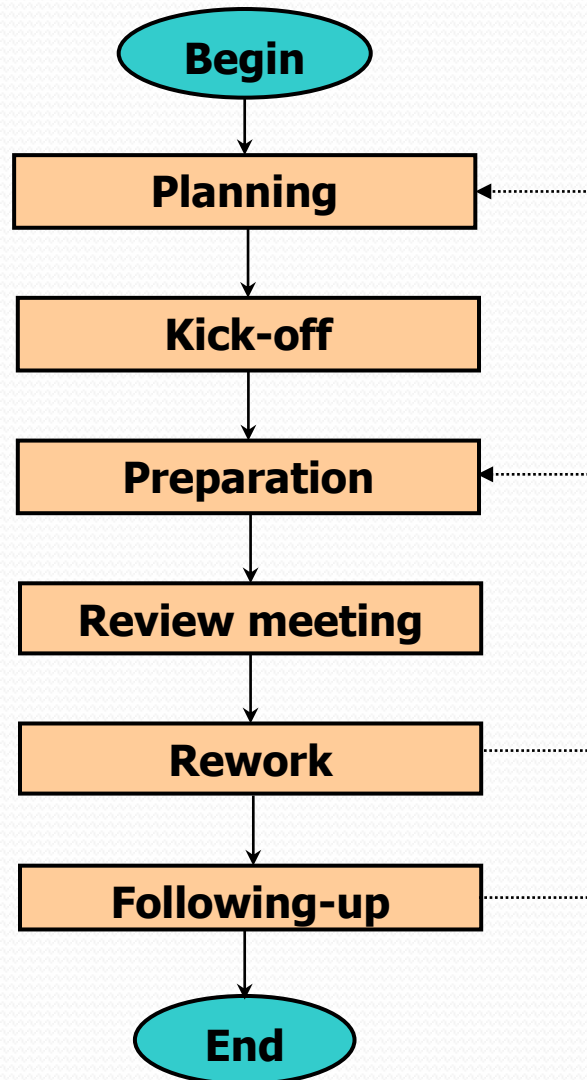
1	2	3	4	5
6	7	8	9	

Contents

1. Static testing techniques
2. Reviews and the test process
3. Review process
4. Static analysis by tools
5. Self-code review

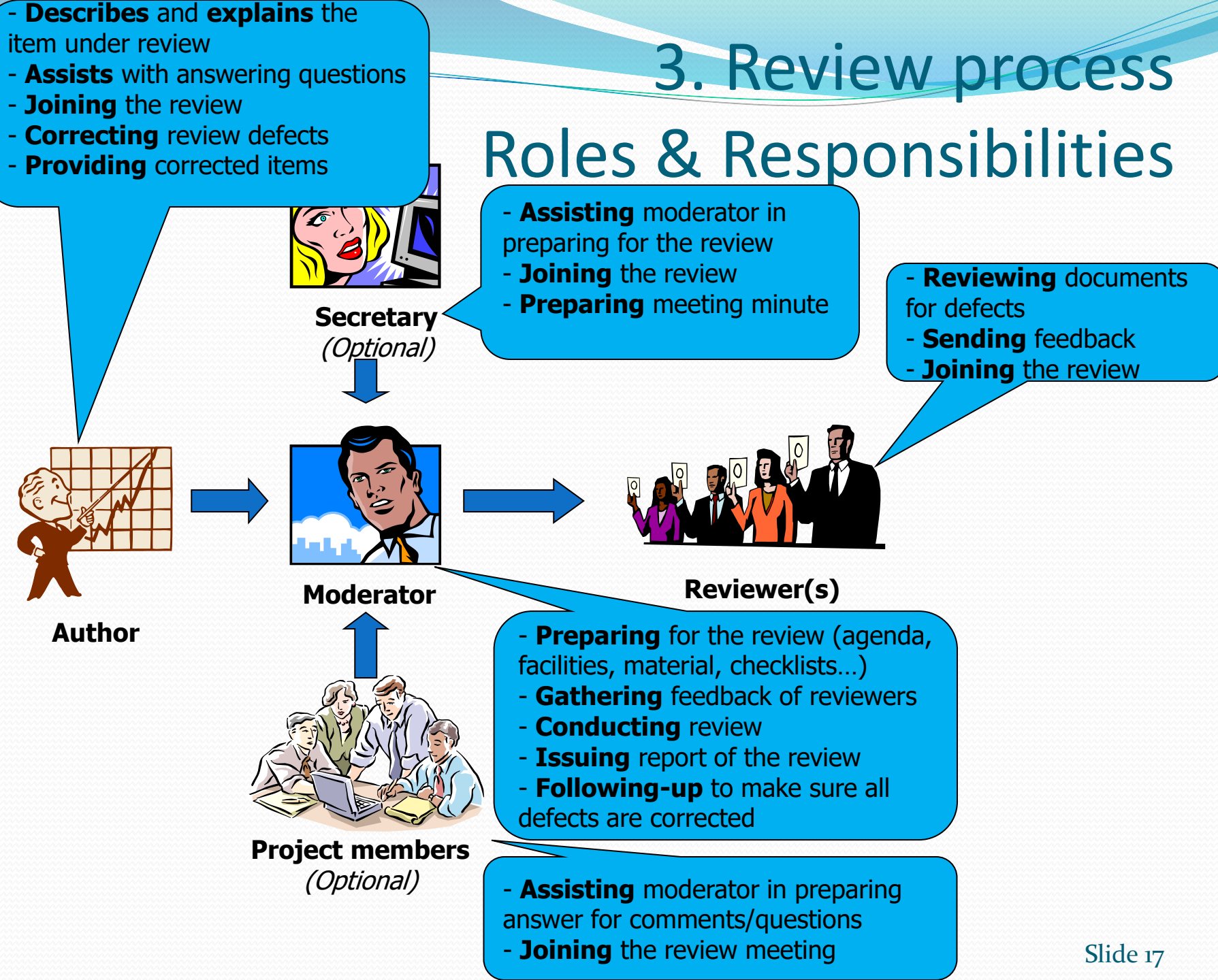
3. Review process

Phases of a formal review



3. Review process

Roles & Responsibilities



3. Review process

Example: Functional Design Document

F-48

Add a function to send an order via email

Feilds

- Email ID
- Recipient name

Buton

- Send

3. Review process

Step 1. Planning

- Begins with a 'request for review' to the **moderator** (review leader)
- **Entry check:** to ensure that *the reviewers' time is not wasted* on a document that is not ready for review
 - a short check of a product sample by the moderator (or expert) does not reveal a large number of major defects
 - e.g. 30 minutes , ≤ 3 major defects / page
 - the document available with line numbers
 - the document has been cleaned up
 - references needed for the inspection are **stable** and **available**

3. Review process

Step 1. Planning (cont'd)

- Decide **which part of the document to review**
 - maximum number of pages depends on the **objective**, **review type** and **document type**
- **Determines the composition of the review team**, normally consists of four to six participants with different roles
- Schedules the meeting venue and time

3. Review process

Step 2. Kick-off meeting

- An optional stage
- Highly recommended
- Goal is to get everybody *on the same wavelength* regarding the document under review
 - **distributing documents** (document under review, source documents and other related documentation)
 - **explaining the objectives, process and relationships** between the document under review and the other documents to the participants
- Can be run as a meeting or simply by sending out the details to the reviewers
- Beneficial for new or highly complex projects

3. Review process

Step 3. Preparation

- To **identify defects, questions** and **comments** to be asked during the review meeting
- **Done by each of the participants on their own** before the review meeting, by using the related documents, procedures, rules and checklists provided
- Using **checklists** can make reviews more effective and efficient
- All issues are recorded
- **Checking rate:** the number of pages checked per hour
 - usually in the range of 5-10 pages per hour, but may be much less for formal inspection

3. Review process

Step 4. Review meeting

- Led by a moderator
- The review meeting is limited to two hours
- Consists of (partly depending on the review type)
 - Logging phase
 - Discussion phase
 - Decision phase

3. Review process

Step 4. Review meeting - Logging phase

Scribe

Reviewer



- The **issues** are mentioned page by page, reviewer by reviewer and are **logged** by the scribe
 - Each defect is logged with a severity (critical, major, minor)
- No real discussion is allowed

3. Review process

Step 4. Review meeting – Discussion phase

Author

Moderator

Reviewer



- Participants can take part in the discussion by bringing forward their **comments** and **reasoning**
- The moderator takes care of people issues
 - intervene if the discussion is getting out of control

3. Review process

Step 4. Review meeting - Decision phase



- The participants have to **make a decision on the document**, sometimes based on **exit criteria**
 - the average number of critical and/or major defects found per page
- The moderator then closes the review meeting

Review process

Example: Functional Design Document

F-48

Add a function to send an order via email

Fields

- Email ID
- Recipient name

Button

- Send
- Reset

3. Review process

Step 5. Rework

- **Correcting** the defects
- Based on the defects detected, the author will make changes in the document, as per the action items of the meeting
- Changes on the document should be easy to identify during follow-up
- Not every defect leads to rework, judge if a defect has to be fixed
- If nothing is done, it should be reported

3. Review process

Example: Rework

- Example

#	Review comment	Status
1	Add Reset button	Done
2	Add detail about Menu	Done
3	Ask the client about facility to share via Social Networking sites	Not valid. Client does not need it

3. Review process

Step 6. Follow-up

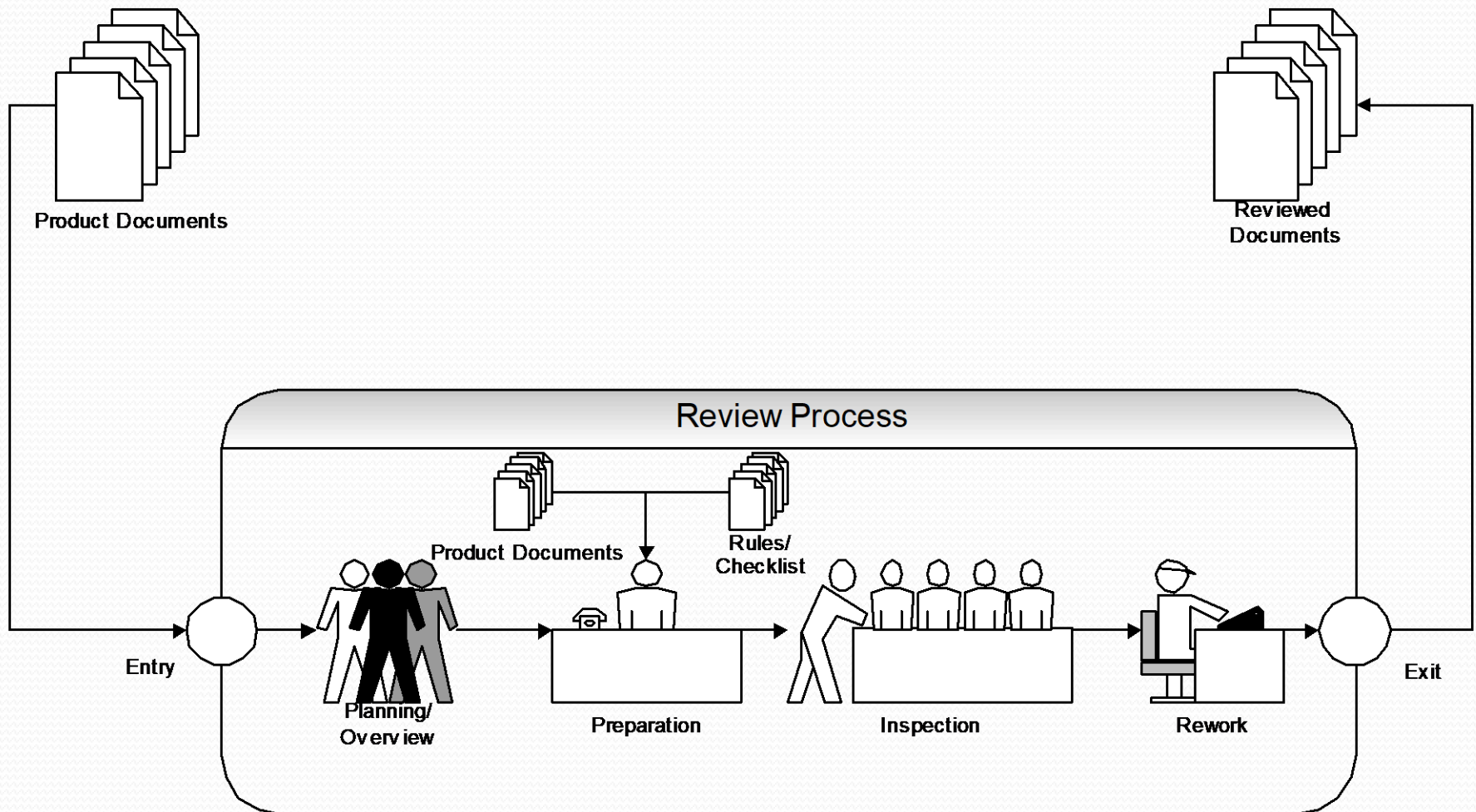
- When the rework is completed, **the author and the moderator meet once again** to review the results
- The moderator will
 - check that **the agreed defects** have been addressed
 - check the **exit criteria** to ensure that they have been met
 - decide if **document passes** review or if another review is necessary
 - **circulate** the reworked document all review participants and collects the feedback
 - gather **metrics**

3. Review process

Metrics of reviews (inspection)

- Review rate
 - the **amount of pages** reviewed in one hour
- Review effort
 - the **amount of time** required to review one page of text
- Defect finding rate
 - the number of defects found during **one hour**
- Defect finding effort
 - the number of defects found per **page**

3. Review process



3. Review process

Success factors for reviews

- Ebook page 70

1	2	3	4	5
6	7	8	9	

Contents

1. Static testing techniques
2. Reviews and the test process
3. Review process
4. Static analysis by tools
5. Self-code review

4. Static analysis

- Static analysis: “Analysis of a program carried out without executing the program” – BS 7925-1
- Goal: find errors
 - Unreachable code
 - Data flow anomaly
 - Infinite loops...
- Usually carried out by means of a supporting tool
- Types:
 - coding standards (coding convention)
 - code metrics
 - code structure

4. Static analysis

Coding convention

- 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.

4. Static analysis

Coding convention

- 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

4. Static analysis

Coding convention

- The reasons to use tool:
 - The number of rules in a coding standard is **usually so large** that nobody can remember them all;
 - Some context-sensitive rules that demand reviews of several files are **very hard to check by human beings**;
 - If people spend time checking coding standards in reviews, that **will distract them from other defects** they might otherwise find, making the review process less effective

4. Static analysis

Coding convention - Common standards

- 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,...

4. Static analysis

Coding convention - Common standards

- 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

4. Static analysis

Coding convention - Naming convention

- 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

4. Static analysis

Code metrics

- Information that can be calculated about **structural attributes** of the code, such as:
 - comment frequency
 - depth of nesting
 - number of lines of code
 - cyclomatic number

4. Static analysis

Code structure

- Control flow structure
- Data flow structure
- Data structure

4. Static analysis

Code structure - Control flow analysis

- Used to discover the **hierarchical flow of control** within a program
- To identify:
 - infinite loops
 - multiple entry to loops
 - unreachable (dead) code,...
- May use a control-flow graph (CFG) as representation
- The code metrics: number of nested levels or cyclomatic complexity

4. Static analysis

Code structure - Control flow analysis

- Example

```
a = 4;  
b = 15;  
z = 7;  
while b > z do  
  begin  
    writeln(z);  
    z++;  
    if a > b then  
      b = a; ← unreachable(dead) code  
  end
```

4. Static analysis


Code structure - Data flow analysis

- Focuses on occurrences of variables, following paths *from* the definitions (an assignment of a value to a variable) of a variable *to* its usages (a read of a variable's value)
 - variables that are never used
 - referencing a variable with an undefined value
 - assigning an incorrect or invalid value to a variable,...

x = y + z *x is defined, y and z are used*



IF a > b THEN read(S) *a and b are used, S is defined*



4. Static analysis

Code structure - Data flow analysis

- Example

n = 0

read(x)

n = 1

while x > y do

begin

read (y)

write(n*y)

x = x - n

end

*Data flow anomaly: n is
re-defined without being used*

*Data flow fault: y is used
before it has been defined*

4. Static analysis

Code structure - Data structure analysis

- The organization of the data itself
 - analyses of logical data structures
 - transformations of logical data structures
 - Example: list, queue, stack, ...
- Provides a lot of information about the difficulty in **writing programs to handle the data** and in **designing test cases**

4. Static analysis

Value of static analysis

- **Early detection** of defects prior to test execution
- **Early warning** about suspicious aspects of the code, design or requirements
- Identification of defects not easily found in dynamic testing
- Improved **maintainability** of code and design
- Prevention of defects

1	2	3	4	5
6	7	8	9	

Contents

1. Static testing techniques
2. Reviews and the test process
3. Review process
4. Static analysis by tools
5. Self-code review

5. Self-code review

- 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 ([example](#))
 - Use team-defined code review checklist

5. Self-code review

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/>

5. Self-code review

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.

Refer: [Code Review CheckList v1.0.xls](#)

5. Self-code review

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

5. Self-code review

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.

5. Self-code review

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 ({})

5. Self-code review

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);
```

5. Self-code review

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)

5. Self-code review

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

5. Self-code review

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

5. Self-code review

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!
```

5. Self-code review

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;
```

5. Self-code review

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++**

5. Self-code review

Programming Practices 6

- Use **parentheses** liberally in expressions involving mixed operators to avoid operator precedence problems
- Try to make the structure of your program **match the intent**, for example:

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

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

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

- should instead be written as
return booleanExpression;