Software Analysis

Sandro Morasca and Dario Bertolino

Università degli Studi dell'Insubria

Dipartimento di Scienze Teoriche e Applicate

Via Ottorino Rossi 9 – Padiglione Rossi

I-21100 Varese, Italy

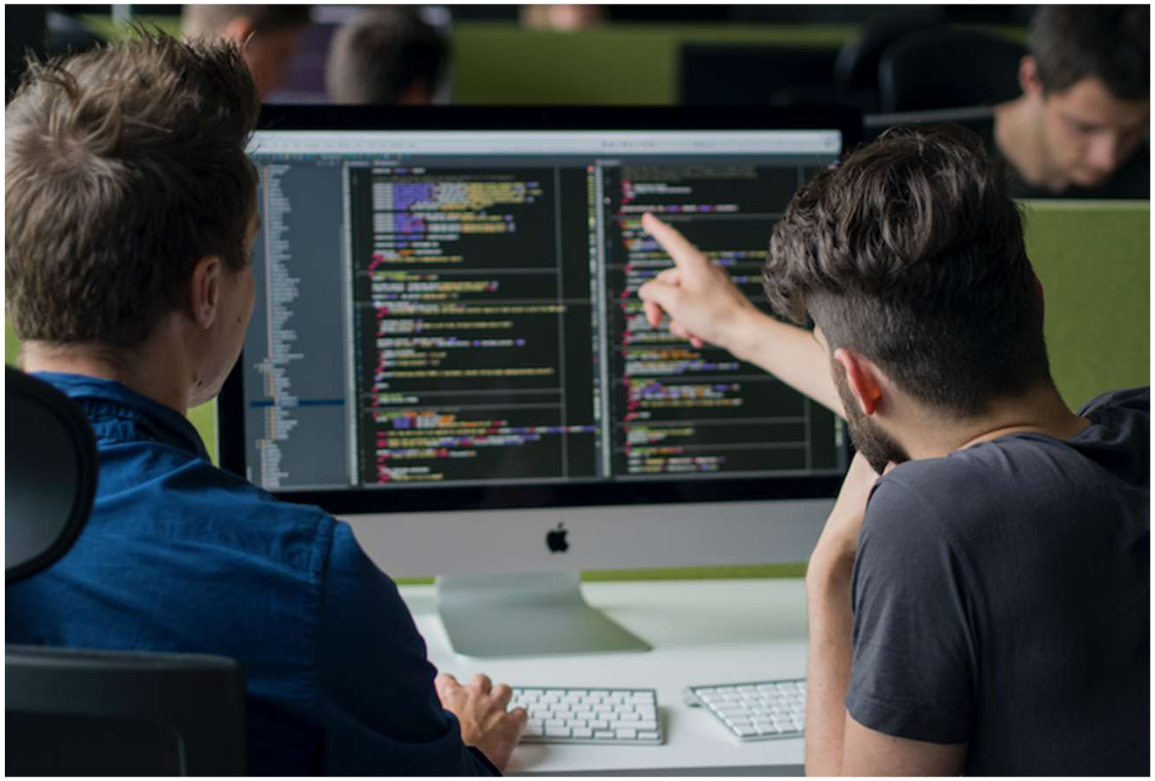
{sandro.morasca,dario.bertolino}@uninsubria.it

Characteristics

- Software analysis is for documents (artifacts)

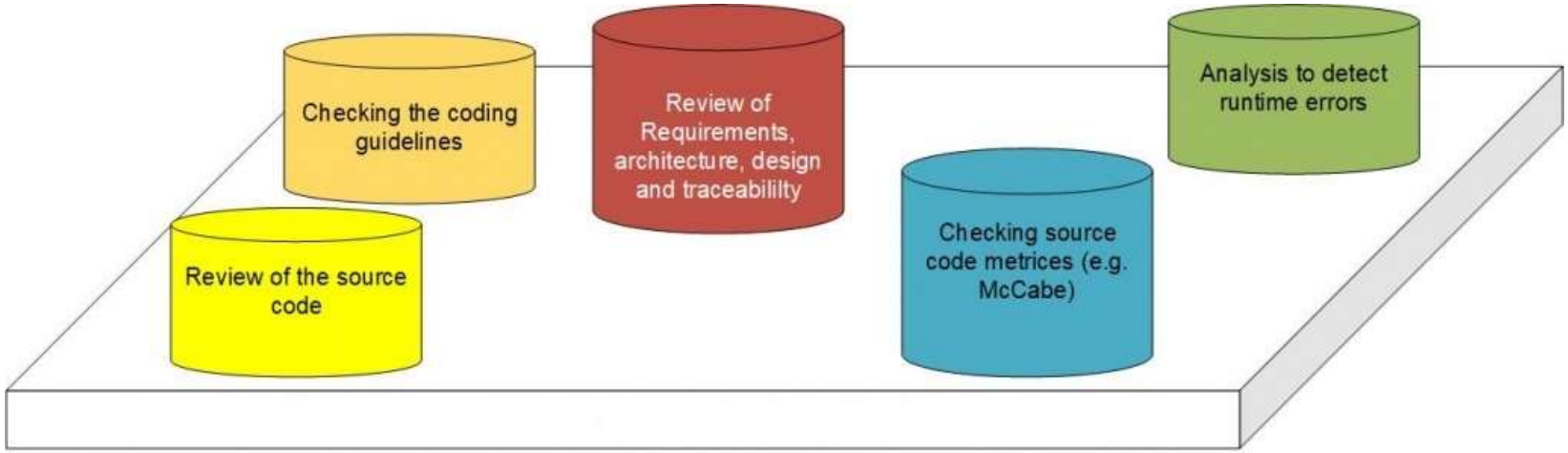
Characteristics

- Software analysis is for code (one of the artifacts)



Software Analysis Is a Set of Different Activities

- **Basic Concepts**
  Reviews
  Tools
  "Formal"
  Analysis



Characteristics

- Is carried out without executing software
- Finds defects
  - not failures
- Allows for early defect detection
- Different types of reviews exists
- **Complements software testing**
  - it is also carried out when debugging

Advantages

- **Inexpensive way of detecting and eliminating defects**
  - early defect detection and correction
  - focuses on root causes
    - not failures
- Can be performed well before testing
- All kinds of software artifacts
- Reviews can find omissions
- Knowledge exchange between people in reviews
- Structuring for presentation helps authors rethink their work
- Quality is a team's concern and not a single person's
- Data collected in reviews can be used for process improvement

Manual Techniques

- Reviews
  - inspection
  - (peer) reviews
  - walkthroughs
  - pair programming
  - pair desk-check
  - ad-hoc review

What is a Review?

- **A review is an evaluation of a product or project status to ascertain discrepancies from planned results and recommend improvements**
- Any software artifact can be reviewed, including
  - requirements
  - specifications
  - architectures
  - designs
  - source code
  - test plans, test specifications, test cases, test scripts
  - user guides
  - web pages

What is a Review?

- **The main manual activity is to examine an artifact and comment it**

We don't need this functionality!

Missing Requirement!

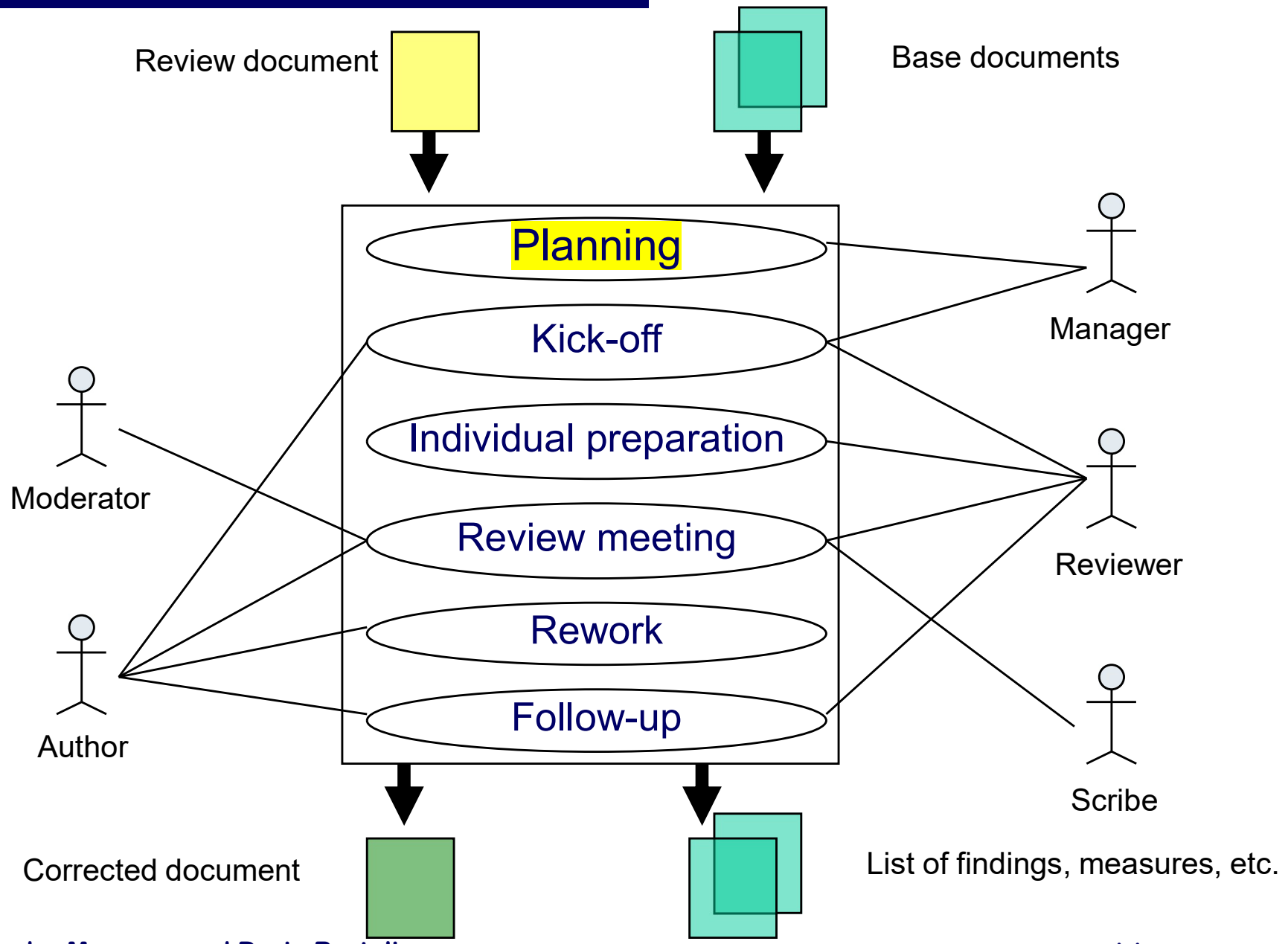
I can't understand the code

The algorithm should be faster

Reviews Goals

- **Typical defects that are easier to find in reviews**
  - deviations from standards
  - requirements defects
  - design defects
  - insufficient maintainability
  - incorrect specifications
- **The emphasis is on defect detection, not correction**

Review Activities

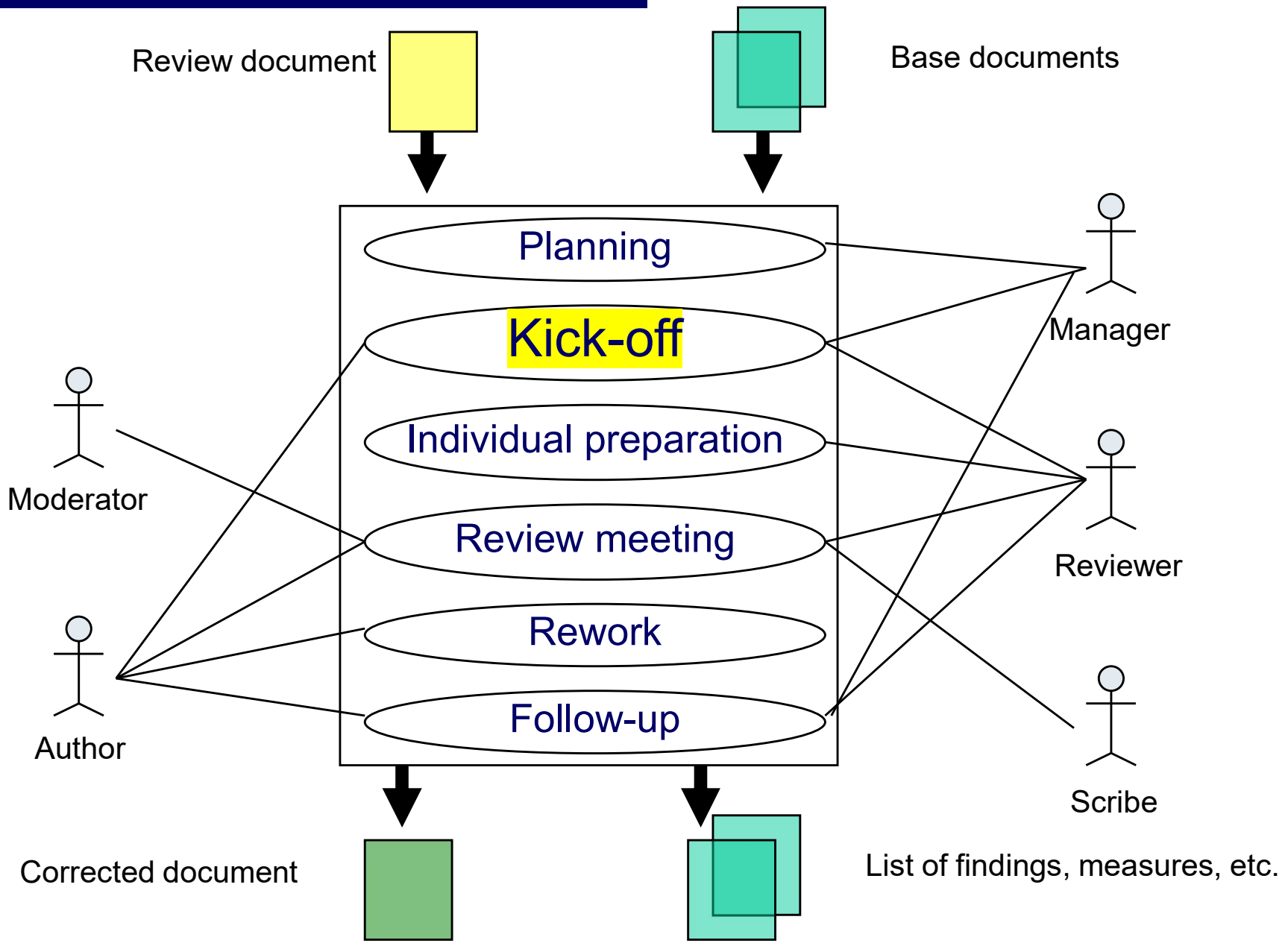


- Selecting the personnel
- **Allocating roles**
- **Defining the entry and exit criteria for more formal review types** (e.g., inspection)
- Selecting which parts of documents to look at

Planning: roles

- **Manager**
  - decides on the execution of reviews
  - allocates time
  - determines if the review objectives have been met

Activities



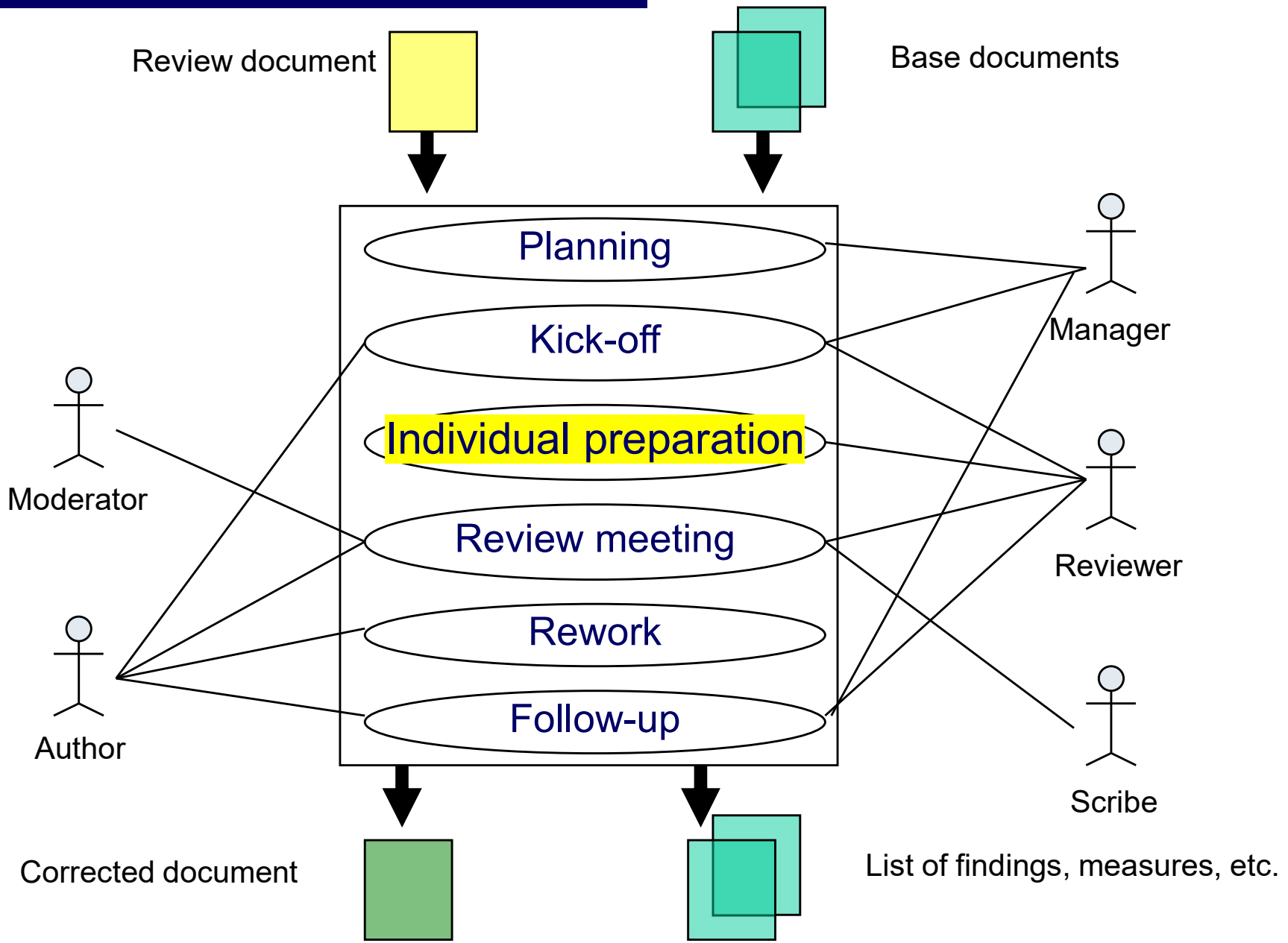
Kick-off

- Distributing the documents
- **Explaining the objectives, process, and documents to the participants**
- Checking the entry criteria for more formal review types (e.g., inspection)

Kick-off: Roles

- **Manager**
  - shows the plan to participants
- **Author**
  - has the responsibility for the document being reviewed
- **Reviewers**
  - have specific technical or business backgrounds
  - identify and describe faults
  - should represent different perspectives and roles

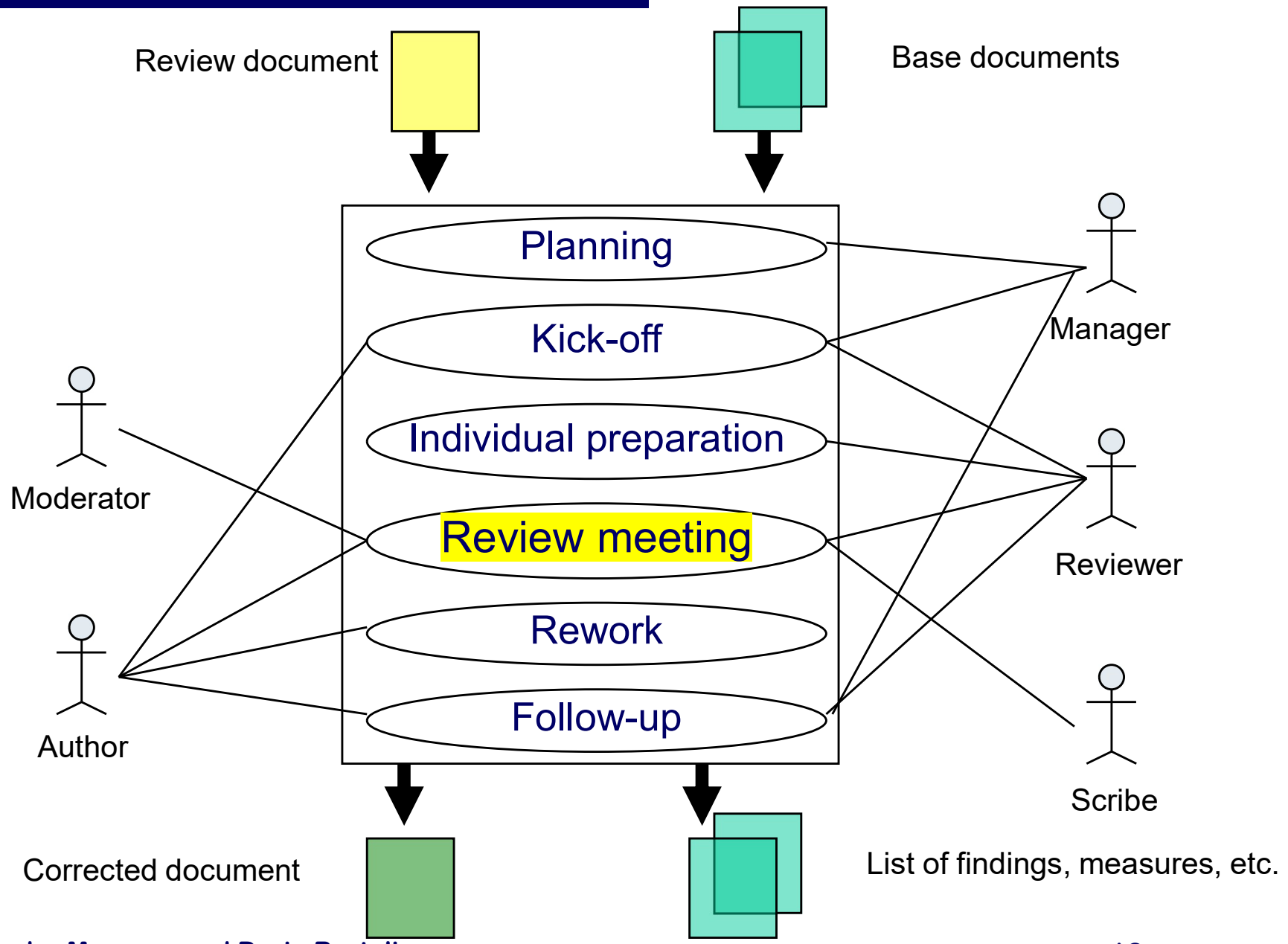
Activities



Individual Preparation

- **Work done by each of the participants on their own**
  - before the review meeting
- Outcomes
  - potential defects
  - questions
  - comments
  - suggestions for improvement
  - …

Activities



Review Meeting

- **Discussion**
  - documented results
  - minutes (for more formal review types)
- **Outcomes**
  - defects
  - recommendations
  - decisions

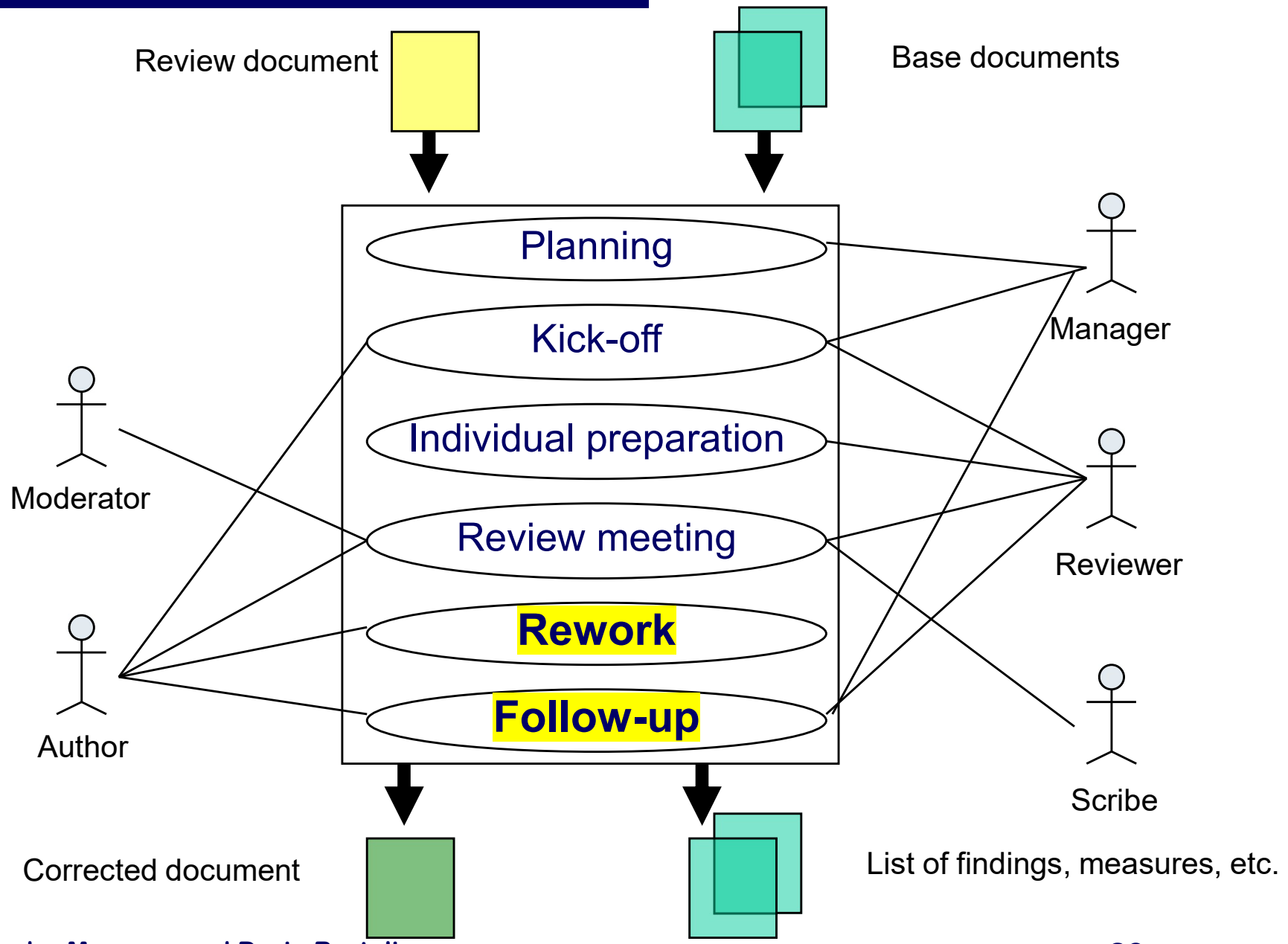
Review Meeting: Basic rules

- **Not a big ego fight**
  - authors: keep an open mind, avoid arguing or defending yourselves
  - reviewers: it's not about showing how smart you are
- **Keep the review team small**
  - 3 – 7 participants
- Limit review meetings to two hours
- **Only find problems in reviews**
  - **do NOT try to solve them at the meeting**

Review Meeting: roles

- **Moderator**
  - leads the review
    - planning
    - running the meeting
    - follow-up
  - tries to find an agreement among different viewpoints
  - probably the central figure for the review
- **Scribe**
  - documents
    - issues
    - problems
    - open points

Activities



Rework and Follow-up

- **Rework**
  - fixing defects
  - done by author
- **Follow-up**
  - checking that defects have been addressed
  - gathering measures
  - checking satisfaction of exit criteria (for more formal review types)

Data on Results

- **Reading rates**
  - source code: 100 – 150 NCLOC/hour during individual preparation
  - text documents: 1 – 5 pages/hour
  - proofreading: 7 – 25 pages/hour
- **Productivity**
  - inspections (formal reviews): 16 – 20 faults/KLOC
  - informal reviews: 3 faults/KLOC
- **Totals**
  - code inspections: about 60% of faults
    - higher than other techniques
  - design + code inspections: 70% - 85% of faults

Degree of Formality

formal ⟷ informal

	Inspection	Team review	Walkthrough	Pair programming	Peer desk-check	Ad-hoc review
Planning	YES	YES	YES	YES	NO	NO
Preparation	YES	YES	NO	NO	YES	NO
Meeting	YES	YES	YES	Continuous	YES	YES
Correction	YES	YES	YES	YES	YES	YES
Verification	YES	NO	NO	YES	NO	NO

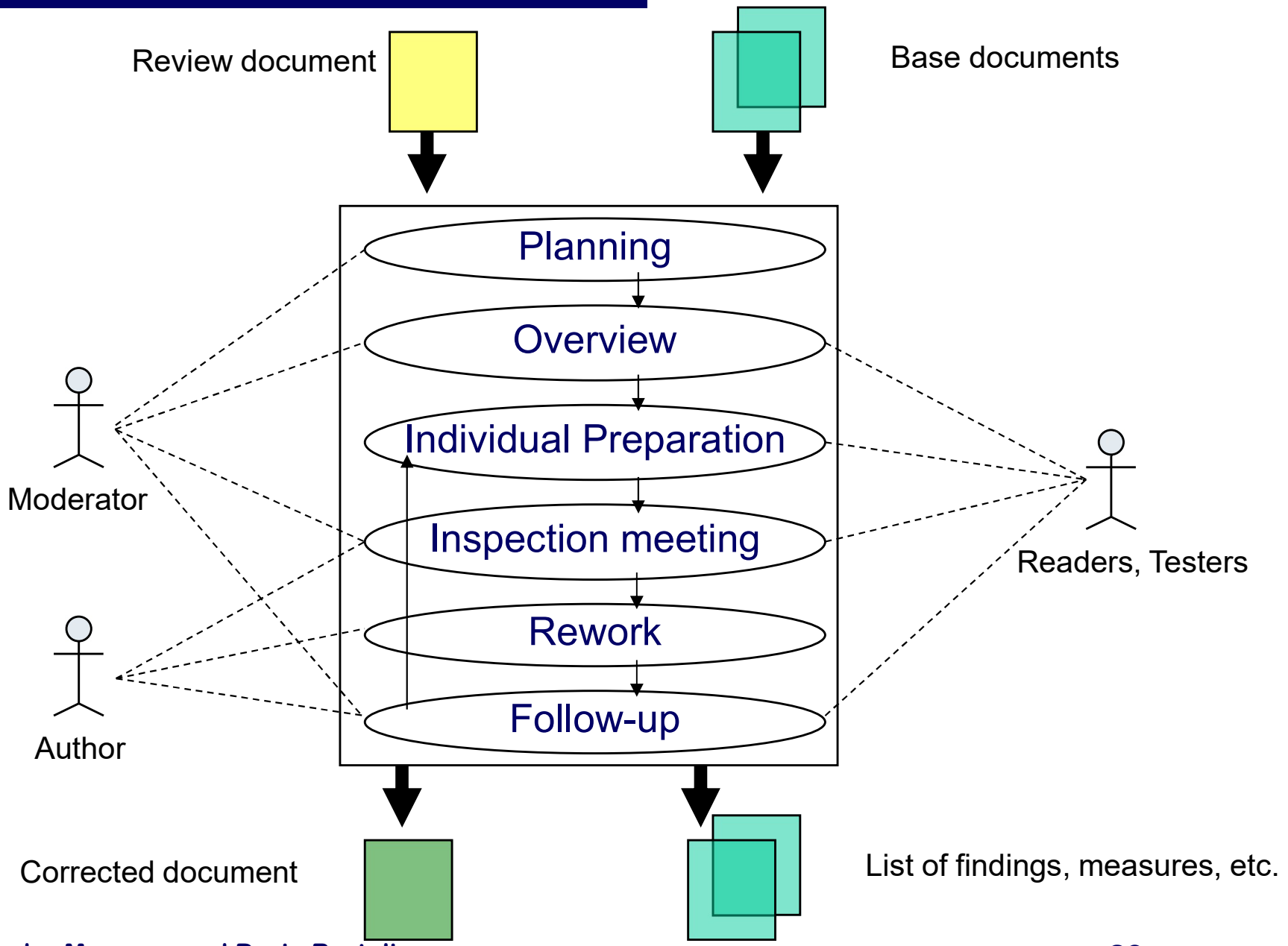
Inspection

- **It is the most systematic and rigorous kind of review**
  - documented procedure
  - Led by a trained moderator—NOT the author
  - Usually peer examination, it is repeated until all objectives are achieved
  - Defined roles: author, reviewers, moderator, scribe
  - Includes measures about the productivity of the reviewing process

Inspection

- **Formal process**
  - rules
  - pre-meeting preparation
  - checklists
  - entry and exit criteria
  - report and list of findings
  - Specific follow-up process
- **Main purpose: Find faults**

Fagan's Code Inspections



Fagan's Code Inspections

- Software Inspection Roles
  - **Moderator**
    - is typically borrowed from another project
    - chairs meeting
    - chooses participants
    - controls process
  - **Readers, Testers**
    - read code to group
    - look for flaws
  - **Author**
    - passive participant
    - answers questions when asked

Fagan's Code Inspections

- **Planning**
  - moderator checks entry criteria, chooses participants, schedules meeting
- **Overview**
  - provide background education, assign roles
- **Individual Preparation**
- **Inspection meeting** (see ahead)
- **Rework**
- **Follow-up** (& possible re-inspection)

In the Meeting

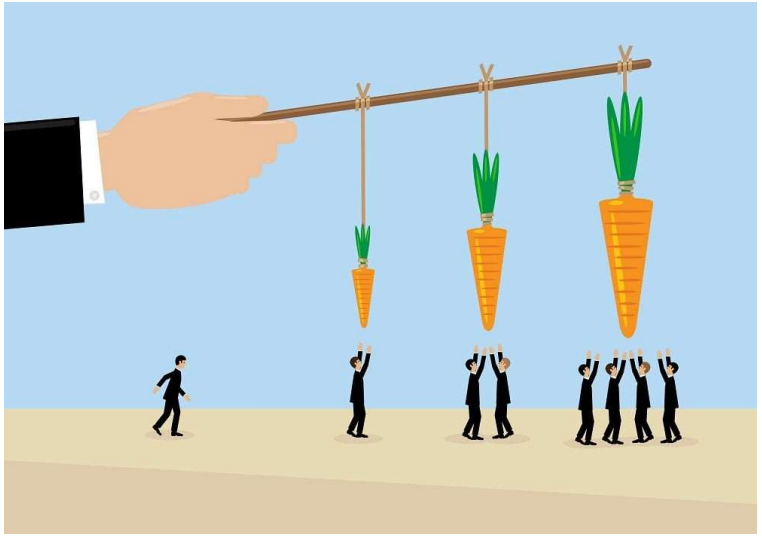
- **Goal: Find as many faults as possible**
  - max 2 x 2 hour sessions per day
  - approx. 150 source lines/hour
- **Approach: Line-by-line paraphrasing**
  - reconstruct intent of code from source
  - may also "hand test"
- **Find and log defects, but do not fix them**
  - moderator responsible for staying on track

Checklists — NASA example

- About 2.5 pages for C code, 4 for FORTRAN
  - Divided into: Functionality, Data Usage, Control, Linkage, Computation, Maintenance, Clarity
- Examples:
  - Does each module have a single function?
  - Does the code match the Detailed Design?
  - Are all constant names upper case?
  - Are pointers not typecast (except assignment of NULL)?
  - Are nested "INCLUDE" files avoided?
  - Are non-standard usages isolated in subroutines and well documented?
  - Are there sufficient comments to understand the code?

Incentive Structure

- **Faults found in inspection are not used in personnel evaluation**
  - programmer has no incentive to hide faults
- **Faults found in testing (after inspection) are used in personnel evaluation**
  - programmer has incentive to find faults in inspection, but not by inserting more

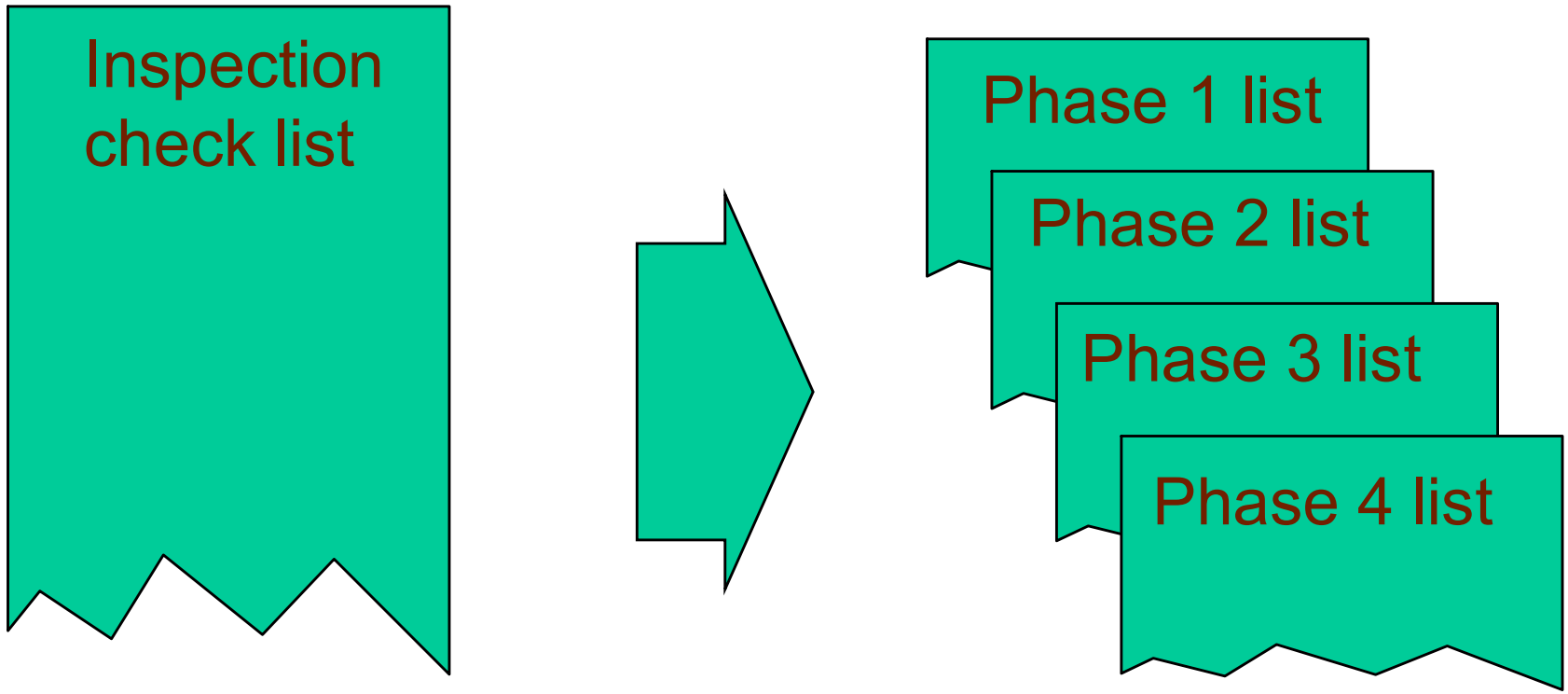


Variation: Active Design Reviews

- **Observation**:
  - an unprepared reviewer can sit quietly and say nothing
    - this must be avoided
- **Variant process**:
  - choose reviewers with appropriate expertise
  - several reviewers to look at different aspects
  - author asks questions of the reviewer
  - reviewer's job is to answer the questions

Variation: Phased Inspections

- Divide inspection into a series of smaller, focused "phases" in a definite order



Variation: Phased Inspections

- **Single inspector for simple, unambiguous checks** (e.g., standards conformance)
  - may be technical writer or junior programmer
- **Multiple inspectors for complex checks** (e.g., correctness)
  - skilled, knowledgeable engineers
  - independent inspections in private
  - reconciliation meeting to compare results

Inspection Automation

- **Although a manual technique, many kinds of automated support are possible**:
  - automate trivial checks (e.g., formatting)
  - reference: checklists, standards with examples
  - focus (highlight, selection) on relevant parts
  - annotation & communication
  - process guidance and (partial) enforcement
    - e.g., InspeQ will not allow check-off until all relevant parts of a document have been observed

Inspection Advantages

- **The evidence says inspection is cost-effective**
  - detailed, formal process, with record keeping
  - checklists
  - self-improving process
  - social aspects of process, especially for author
  - consideration of entire input space
  - applies to incomplete programs

Inspection Disadvantages

- **Limitations**
  - scale
    - inherently a unit-level technique
  - non-incremental
    - what about evolution?

Degree of Formality

formal ⟷ informal

	Inspection	Team review	Walkthrough	Pair programming	Peer desk-check	Ad-hoc review
Planning	YES	YES	YES	YES	NO	NO
Preparation	YES	YES	NO	NO	YES	NO
Meeting	YES	YES	YES	Continuous	YES	YES
Correction	YES	YES	YES	YES	YES	YES
Verification	YES	NO	NO	YES	NO	NO

Team Review

- **Technical review**
  - a peer group discussion that focuses on achieving consensus on the technical approach to use
- **Planned and strucured**
  - but less formal than inspections
  - may range from very formal to very informal
- **Participants**
  - different skills, including
    - peers and technical experts
  - focus on different problems
  - learning opportunity

Team Review

- **Leader**
  - trained moderator, if possible
  - it can also be the author
- **Pre-meeting preparation required**
- Optional activities:
  - checklists, review report, list of finding
- Productivity
  - about 2/3 of inspection productivity
- Main goals
  - discuss, make decisions, evaluate alternatives, find defects, solve technical problems, check conformance to specifications and standards

Team Review

Degree of Formality

formal ⟷ informal

	Inspection	Team review	Walkthrough	Pair programming	Peer desk-check	Ad-hoc review
Planning	YES	YES	YES	YES	NO	NO
Preparation	YES	YES	NO	NO	YES	NO
Meeting	YES	YES	YES	Continuous	YES	YES
Correction	YES	YES	YES	YES	YES	YES
Verification	YES	NO	NO	YES	NO	NO

Walkthrough

- **Informal review in which the author describes the product to a group of peers and solicits comments**
  - example: going through a specific user scenario
- **Meeting led by authors, while other roles are usually not defined**
  - author selects and presents the material
  - author usually records the findings
- **Open-ended session**
- Optional: pre-meeting preparation of reviewers
- Productivity: 50% of defects found in inspections (according to a case study)
- Purposes
  - learning, gaining understanding, brainstorm alternative solutions, find defects

This My C++ Class


```
class CoinBox
{
    int totalQtrs; // total quarters collected
    int curQtrs;   // current quarters collected
    boolean allowVend; // true=vending is allowed
public:
    CoinBox() {reset();}
    void addQtr(); // add a quarter
    void returnQtrs() {curQtrs=0;}
                       // return current quarters
    boolean isAllowedVend() {return allowVend;}
    void reset() {totalQtrs = 0; allowVend = false;curQtrs = 0;}
    void vend(); //if vending allowed, update totalQtrs and
curQtrs
};
```

What Do You Think about My Work?


```
void CoinBox::addQtr()
{
  curQtrs = curQtrs+1; //add a quarter
  if (curQtrs > 1)
             //if more than one quarter is collected,
     allowVend = false;  //then set allowVend
}
void CoinBox::vend()
{
  if (isAllowedVend()) //if allowVend
  {
      totalQtrs = totalQtrs+curQtrs;  //update totalQtrs
      curQtrs = 0;                    //curQtrs
      allowVend = false;              //allowVend
  }                    //else no action
}
```

What is an Informal Review

- Not based on a formal and documented process
- Usefulness varies depending on the reviewer
- **Main purpose: inexpensive way to get some benefit**
- Typical informal reviews

Degree of Formality

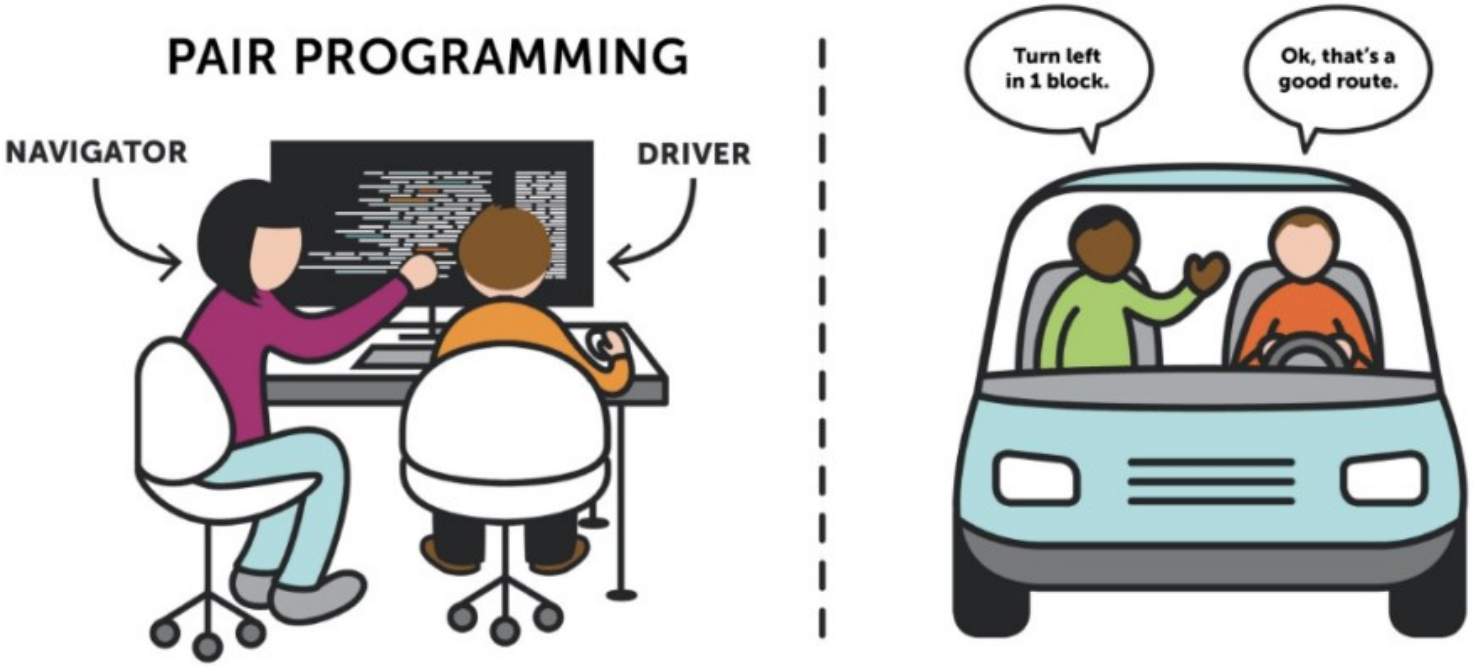
formal ⟷ informal

	Inspection	Team review	Walkthrough	Pair programming	Peer desk-check	Ad-hoc review
Planning	YES	YES	YES	YES	NO	NO
Preparation	YES	YES	NO	NO	YES	NO
Meeting	YES	YES	YES	Continuous	YES	YES
Correction	YES	YES	YES	YES	YES	YES
Verification	YES	NO	NO	YES	NO	NO

Pair Programming

- **Two developers work together on a single workstation**
- **Advantages**
  - communication
  - continuous, incremental reviews
  - continuous psychological pressure
  - more fun
  - fewer defects
- **Disadvantages**
  - more expensive

Pair Programming



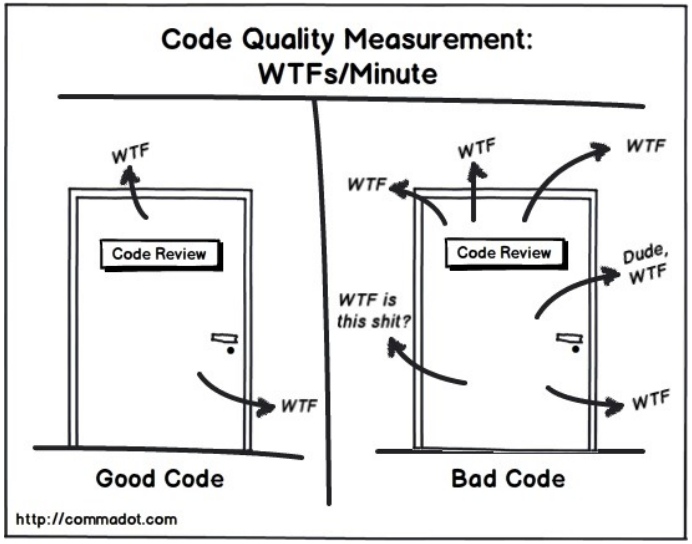
Degree of Formality

formal ⟷ informal

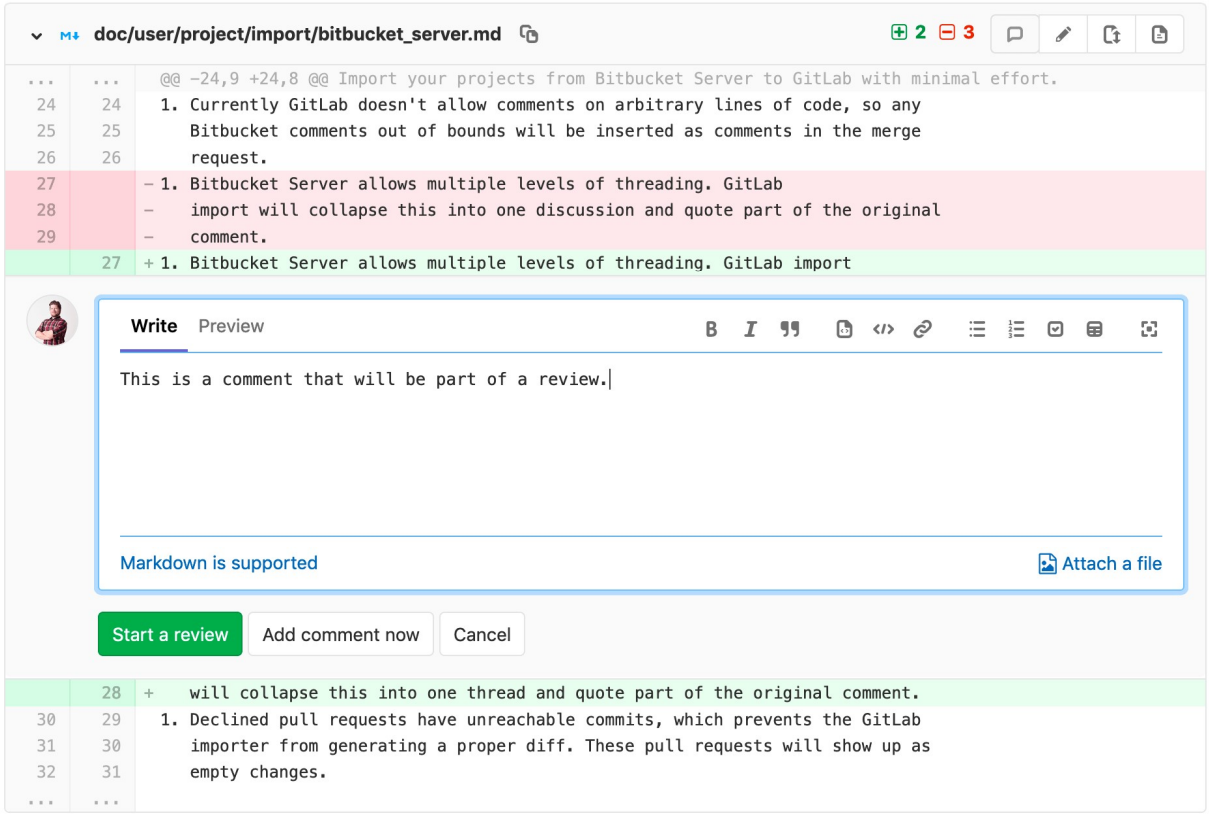
	Inspection	Team review	Walkthrough	Pair programming	Peer desk-check	Ad-hoc review
Planning	YES	YES	YES	YES	NO	NO
Preparation	YES	YES	NO	NO	YES	NO
Meeting	YES	YES	YES	Continuous	YES	YES
Correction	YES	YES	YES	YES	YES	YES
Verification	YES	NO	NO	YES	NO	NO

Peer Desk-check

- **In a peer desk-check**
  - reviewer else looks at the source listings alone
  - then, author and reviewer discuss the findings
- **Pass-around**
  - multiple, concurrent desk-check
  - involves several people
  - at the end, the author puts together the results



Interactive Tools for peer Desk-check in every version control software of the market (e.g. github, gitlab)



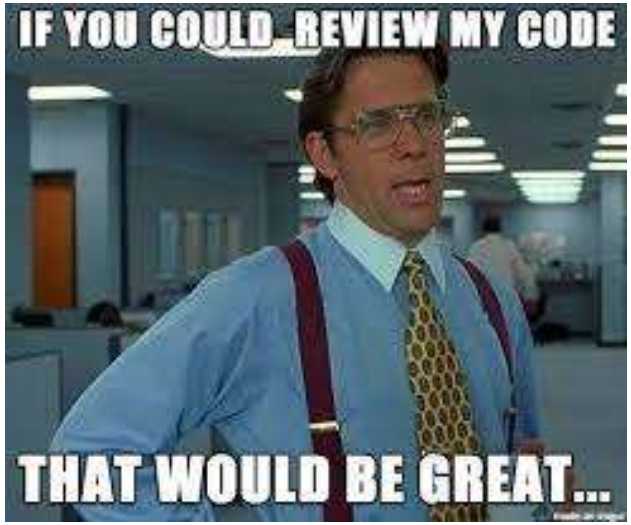
Degree of Formality

formal ⟷ informal

	Inspection	Team review	Walkthrough	Pair programming	Peer desk-check	Ad-hoc review
Planning	YES	YES	YES	YES	NO	NO
Preparation	YES	YES	NO	NO	YES	NO
Meeting	YES	YES	YES	Continuous	YES	YES
Correction	YES	YES	YES	YES	YES	YES
Verification	YES	NO	NO	YES	NO	NO

Ad-hoc Review

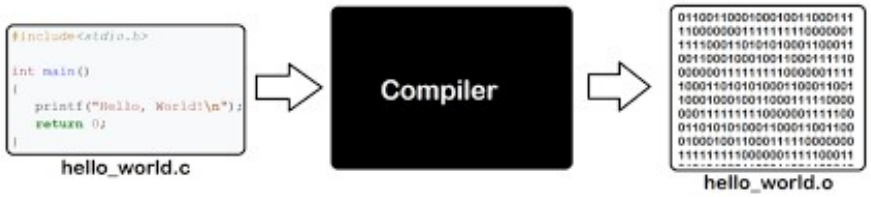
- **A programmer asks another to get a different perspective**
- Goal is to find problems that the programmer cannot usually see alone

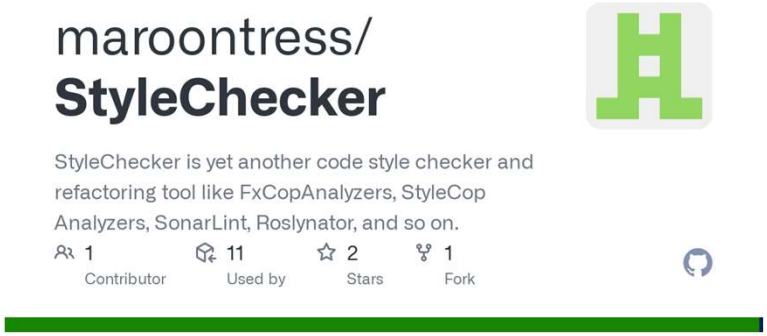


Choosing a Review Approach

	Inspection	Team review	Walkthrough	Pair prog.	Desk-check	Passaround
Find defects	X	X	X	X	X	X
Check conformance	X	X			X	X
Check completeness and correctness	X		X		X	X
Assess understandability and maintainability	X	X		X		X
Demonstrate quality of critical or high-risk components	X					
Collect data for process improvement	X	X				
Measure document quality	X					
Educate other team members		X	X	X		X
Reach consensus on an approach		X	X	X		
Ensure compatibility of changes		X	X		X	
Explore alternative approaches			X	X		
Simulate execution of a program			X			
Minimize review costs					X	

- compiler
- code style checker
- bug pattern detector
- design/architecture analyzer
- data/control flow analyzer
- formal verification tools





Automated Static Analysis

- **Static analyzers are software tools that scan source code to detect possible faults**
  - no code execution is carried out
- **Potential problems**
  - control flow problems: loops with multiple entry or exit points, unreachable code
  - data use problems: uninitialized variables, variables that are never used
  - interface problems: inconsistency of routine and procedure declarations and use
  - other problems: common security holes, duplicate blocks, style conventions and standards

Typical Defects Detected

- Syntax rule violations
- Violations of coding and naming conventions
- Unreachable code
- Undeclared variables
- Uninitialized variables
- Reference to an undefined variable
- Variables that are never used
- Methods that are never called
- Function results are never used
- Duplication of code blocks

Typical Defects Detected

- Array out of bounds
- Parametery type mismatch
- Inconsistent interfaces
- Security vulnerabilities
- Null pointer dereferencing
- Misuse of pointer
- Division by zero
- Deadlocks
- Performance problems
- Lack of localization
- …

False Positives vs. False Negatives

- False positives
  - problems are reported that will never occur
    - useless work to fix them
    - effort diverted from real problems
- False negatives
  - problems are not reported
    - faults are left in the shipped product

- **Different warning levels**
  - some levels may be disabled
    - balance between false positives and false negatives
  - syntax checks
  - type checks
  - cross references
    - undeclared variables
    - unused return values
    - unreachable code
  - exception handling
  - …

Code Style Analyzers

- **Using a consistent coding style**
  - makes code more understandable
  - may prevent specific faults
    - putting constants on left-hand side of comparisons
    - avoiding assignments in conditions
- **Typical defects detected**
  - naming conventions violated
  - missing Javadoc comments
  - duplicated code blocks
  - …
  - coding and design flaws (e.g., equals and hashCode())

- Initial situation
  - complex system, no documentation, degraded architecture, unavailable developers, etc.
    - has the agreed-on design been actually implemented?
    - does the software system comply with the standards?
    - is it possible to extend or test the software with reasonable effort?
    - …

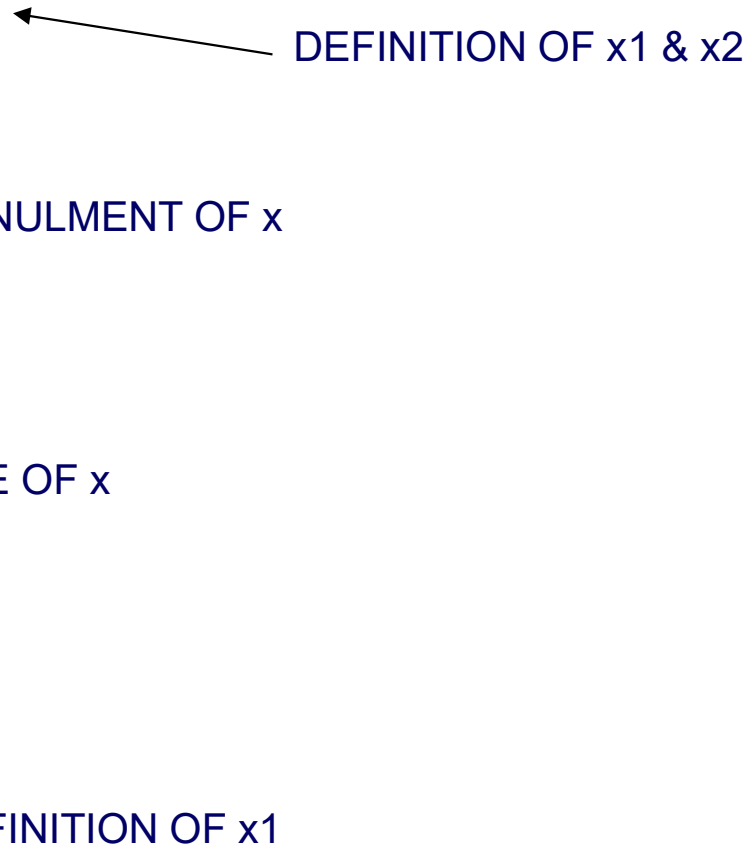
- Design/architecture review based on measures and static analysis, to identify potential problems
  - size limits
    - big classes (e.g., with more than 100 methods)
  - OO misuse, e.g., direct access to member data, heavy use of type casts, overuse of inheritance, classes that reference subclasses, etc.
  - cycles and bottlenecks
    - changing them would affect a large part of the system
  - architecture conformance problems
    - e.g., upward references, references that skip one or more layers, interface violations

Data Flow Analysis Basic Ideas

- Analyze a program based on the events each variable goes through at each statement
- **Definition**: the statement gives the variable a possibly new value (e.g., variable x2 receives a possibly new value in x2 := x)
- **Use**: the statement "looks at" the variable's value (e.g., use of variable x in assignment x2 := x)
- **Annulment**: the variable's value no longer exists (e.g., annulment of variable x in x: integer;)

An Example


```
1.  procedure swap(x1, x2: integer)
2.  var
3.      x: integer;
4.  begin
5.      x2 := x;
6.      x2 := x1;
7.      x1 := x;
8.  end;
```



General Rules

- The use of a variable in a statement must always be preceded in every sequence that leads to that statement by a definition of the variable, without any annulments of the variable between that definition and that use.

- For x
  - a u u (problem!)

General Rules

- The definition of a variable in a statement must always be followed in at least one path from that statement by a use of the variable before a new definition of the variable or an annulment of the variable.

- For x2
  - d d d (problem!)

General Rules

- For x1
  - d u d

Other Applications

- The same kind of analysis can be carried out in other contexts, e.g., when dealing with files
- **Open**
- **Close**
- **Read**
- **Write**

Rules

- "read" or "write" must always be preceded by "open," without "close in between
- "open" must be followed by "close" before another "open"
- "read" (resp. "write") cannot be followed by "write" (resp. "read") without "close + open" in between
- "close" must always be preceded by "open"

Using a Regular Language

- Alphabet: {a, u, d, ε}
  - ε is the empty string
- Every symbol of the alphabet is a regular expression
- Given two regular expressions e1 and e2, the following ones are regular expressions
  - e1 e2 → SEQUENCE
  - e1 + e2 → DECISION
  - (e1)* → LOOP

Using a Regular Language

- Given
  - statement if C then S1 else S2
  - the regular expressions S1x and S2x associated with x in S1 andS2

  the regular expression S1x+S2x is associated with x
- Given
  - statement while C do S
  - the regular expression Sx associated with x in S

  the regular expression (Sx)* is associated with x

Another Example


```
program example(input, output)
var
        x, y, res: integer;
begin
        read(x);
        read(y);
        res := 0;
        if y < 0 then begin
                                x := -x;
                                y := -y
                        end
        while y > 0 do
        begin
                res := res+x;
                y := y-1
        end;
        write(res)
end.
```

- For x
  - ad(ud+ε)(u)*
- For y
  - adu(ud+ε)u(udu)*
- For res
  - ad(ud)*u

Another Example


```
program example(input, output)
var
        x, y, z: integer;
begin
        read(x);
        read(y);
        if x > y then x := x-1
                 else y := y-1;
        while x+y > 0 do
        begin
                x := y-z;
                z := x+y;
                y := y-x
        end;
        if z > 0 then z := x+y+z
                 else z := x-y-z;
end.
```

- For x
  - adu(ud+ε)u(duuu)*(u+u)
- For y
  - adu(ε+ud)u(uuudu)*(u+u)
- For z
  - a(ud)*u(ud+ud)