Software Engineering

mariaiuliana.dascalu@gmail.com

Politehnica University of Bucharest, Romania
Department of Engineering in Foreign Languages

Objectives of the current presentation

- continue with Quality Assurance Management

Defect discovery - Where do the defects come from?

- Techniques for discovering the defects
  - Inspections
  - Walkthroughs
- How to estimate how many defects remain undiscovered
  - Defect density
  - Capture-recapture

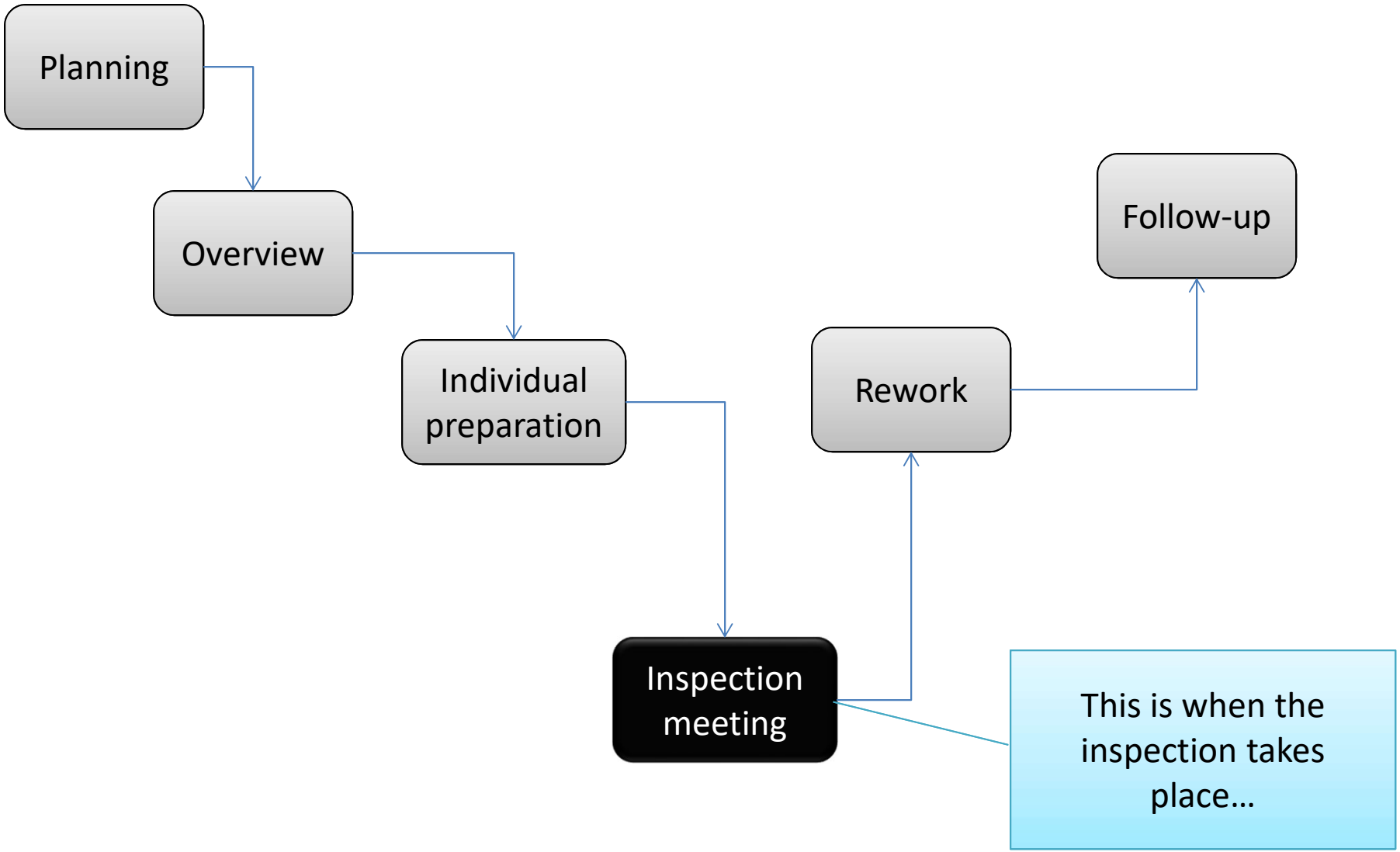
Software inspections

- Provide a way of structured reviews of software documents
  - Involve people examining the source representation with the aim of discovering anomalies and defects
  - Do not require execution of the system, so they may be used also before implementation
- Are means of verification and validation of software artifacts
- Can be used for all kind of software artifacts
  - Source code
  - Design documents
  - Requirements specifications
  - Test data

Roles in inspections

Role	Description
Author/owner	Person responsible for the inspected document and for fixing defects after the inspection
Inspector	Finding errors, omissions, inconsistencies in the documents; might identify broader issues which are outside of the scope of the inspection
Reader	Presents the document during the inspection meeting
Scribe	Records the results of the inspection meeting
Moderator	Manages the process and facilitates the inspection

Inspection process



Planning and overview

- Planning, when the moderator:
  - Selects the inspection team
  - Organizes the room
  - Ensures completeness of materials
- Overview, when the author:
  - Presents the documents to be inspected
  - Describes the intention of the documents and their purpose

Individual preparation and inspection meeting

- Individual preparation:
  - Each individual studies the documents and identifies defects
- Inspection meeting:
  - The team discusses
    - The defects found
    - The conformance to the standards
    - Quality of the document
  - The team should not
    - Propose solutions
    - Recommend changes to the components

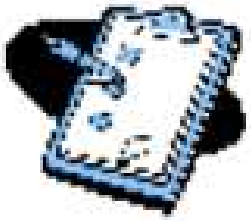
Finding defects during individual preparation

- Using guided reading (e.g. checklists)
- Checklists
  - Group common problems which the inspectors should look for
  - Contain list of types of defects that could be found in the documents (e.g. inconsistency between models, testability of the requirements)
    - The list is based on the examples from books and from historical data on defects
  - For code inspections, checklists vary depending on the programming language used

Example checklist

- Correctness

- ☐ Do any requirements conflict with or duplicate other requirements?
- ☐ Is each requirement written in clear, concise, unambiguous language?
- ☐ Is each requirement verifiable by testing, demonstration, review, or analysis?
- ☐ Is each requirement in scope for the project?
- ☐ Is each requirement free from content and grammatical errors?
- ☐ Is any necessary information missing from a requirement? If so, is it identified as TBD?
- ☐ Can all of the requirements be implemented within known constraints?
- ☐ Are any specified error messages unique and meaningful?



Outcome of the inspection

- Organized list of faults
  - Redundant faults are removed
  - Conflicting faults are resolved
  - All faults are clearly specified

Fault number	Design document	Ref.	Description	Checklist	Comments
F.1	SRS 1	P 3, FR1	FR1 is contradicting with FR2	SRS checkpoint 2	…

Rework and follow-up

- Rework
  - The author makes changes to the document to remove the defects
- Follow-up
  - The moderator decides whether a re-inspection is required
    - If the re-inspection is not required then the moderator approves the document to be the final release

Pre-conditions for inspections

- A precise specification must be available
  - The process of inspections should be prepared beforehand
- Team members must be familiar with the appropriate standards
  - Team members should know what is a good/bad documents (e.g. coding standards, architectural styles…)
- Syntactically correct documents must be available
  - The document to be inspected should not be a draft version, but a final one
- A checklist should be prepared
  - To guide the reviewers

Efficiency and costs of inspections

- Efficiency
  - 500 statements/hour during overview
  - 125 source statement/hour during individual preparation
  - 90-125 statements/hour can be inspected during inspection meeting
- Costs
  - Inspecting 500 lines costs about 40 man/hours effort=> inspection is therefore an expensive process

- Still, some defects remain undiscovered
- How can we estimate the number of undiscovered defects?

Defect density: a measure of how good the software is?

- Do you think that the number of defects discovered is a good measure of a quality of a program/module/component?
  - How about small and large components?

- What about defect density?
  - Do you think this is a better measure?

  DD = no_of_defects/LOC

  (number of defects/lines of code)

Capture recapture

- Capture-recapture is a method for estimating how many defects a given artifact might have:
  - Based on statistics
  - Can be applied to any software artefact which can be inspected, tested, etc.

- The basic idea is that:
  - we have two independent reviewers
  - we perform the QA activity and check how many of the found faults overlap
  - based on the overlap we can estimate how many faults remain undetected after the QA activity

Capture-recapture: basics from the "theory"

- Suppose we perform counting in the population of size N (which is unknown and we wish to estimate it)
- Suppose we find M organisms, we tag them and release to the population
- After some time we capture n organisms and see that m of them have been marked
- We assume that the proportion of the marked organisms is the same as the proportion of the recaptured organisms: http://home.comcast.net/~sharov/PopEcol/lec2/caprecap.html
- N/M=n/m = > N=n*M/m

Restrictions

1. The population is closed,
2. animals do not loose their marks during the experiment,
3. all marks are correctly noted and recorded at each trapping occasion, and
4. each animal has a constant and equal probability of capture on each trapping occasion. This also implies that capture and marking do not affect the catchability of the animal.

Example of Applying Capture-Recapture in Software Inspections

The size of the overlap between the sets of faults found by reviewers indicates the number of faults left: if overlap increases, the number of defects left decreases.

http://leansoftwareengineering.com/2007/06/05/the-capture-recapture-code-inspection/

$$N = \frac{n1n2}{m}$$

where:

N = Estimate of total defect population size
n1 = Total number of defects discovered by first reviewer
n2 = Total number of defects discovered by second reviewer
m = Number of common defects discovered by both reviewers

Restrictions

1. Once the document is issued for inspection, it must not be changed; and the performance of the reviewers should be constant, i.e. given the same document the reviewers should find the same faults.
2. Reviewers must not reveal their proposed faults to other reviewers.
3. Reviewers must ensure that they accurately record and document every fault they find.
4. All reviewers must be provided with identical information, in terms of source materials, standards, inspection aids, etc. and this material must be available to them at all times.

Testing

- The process of executing a program with intention of finding errors
- Operating a system/component, under specified conditions, observing/recording the results and making an evaluation of some aspect of the system/component (IEEE)
- Code inspection != testing
- Testing strategies: test entirely (“big bang theory”) vs. incremental testing (uses stubs)

Testing Methods

- Black-box testing (functional testing) vs. white-box testing (structural testing)
- Integration testing: top-down, bottom-up
- Interface testing: takes place when modules are integrated
- Stress testing:
  - load tests (functional performance is tested under maximum operational load)
  - durability tests (tests are carried out in physically extreme operating conditions such as high temperature, humidity etc)
- Security testing: access control, backup of files and recovery in case of system failure, logging of transactions etc)
- Alpha site testing vs. beta site testing
- Manual vs. automated testing with specialized tools (e.g. for unit testing: JUnit, NUnit, CppUnit…)

What would be some valid test cases for the black-box testing?

- procedure search(Key:ELEM, T:SEQ of ELEM, Found:BOOLEAN, L:ELEM_INDEX)
- Pre-condition: T.first()<=T.last()
- Post-condition:
  - Found and T[L]=Key
  - not Found and not (exist i, T.first() <= i <= T.last(), T[i]=Key)
- Equivalence classes:
  - Dates in which the key element is in the sequence (first element, last element, middle element)
  - Dates in which the key element isn't in the sequence
  - The sequence has one value
  - The sequence has more values

Black-box testing example

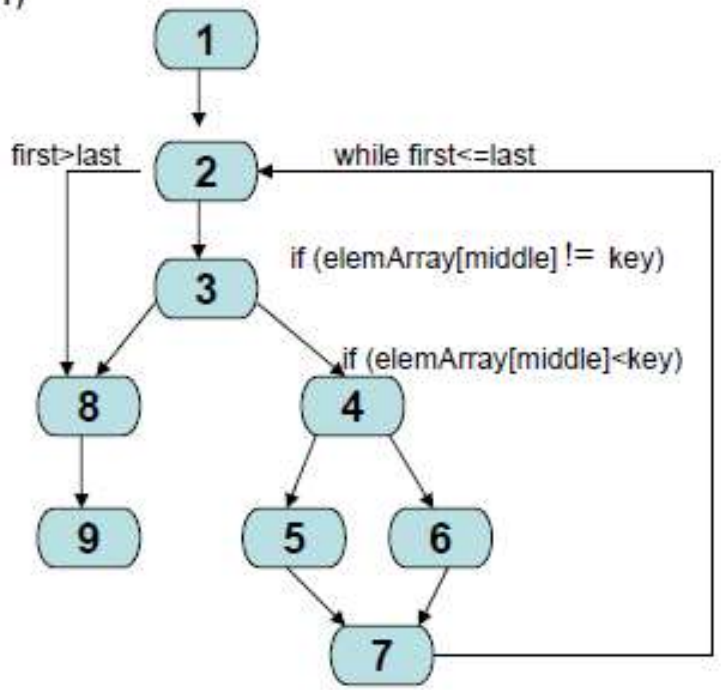
Input sequence	key	Expected results (found, L)
17	17	true,1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

What are the independent execution paths for the white-box testing?

```
class Result{
      public boolean found;
      public int index;
}
class BinSearch{
 public static void search(int key, int[] elemArray, Result r)
 {
      int first = 0;
      int last = elemArray.length-1;
      int middle;
      r.found = false; r.index=-1;
      while (first<=last) {
                middle = (first+last)/2;
                if (elemArray[middle] == key){
                          r.index = middle;
                          r.found = true;
                          return;
                } else if (elemArray[middle]<key)
                          first = middle+1;
                     else
                          last = middle-1;

                }
      }//while
  }//search
}
```

Execution graph



Absence of defects of a product = good quality?

- What is a quality process?
- How would you recognize a quality process in a restaurant?
- How about software development?
- Does the following environments say something about the quality of the product produced (food, software)?

The bottom line is…

- Clean kitchen does not necessarily produce tasty food, but
  - we'd rather eat a so-so tasty food from a clean kitchen.

- Good processes does not necessarily develop quality software, but
  - we'd rather fly in Airbus with a software from a well-known company than a small software development house.

- So, there is a need for quality processes, since they decrease the risk of companies producing low quality products.