Inspections and Cleanroom

Reading assignment

- **Introduction to dynamic analysis**
  - Zhu, Hong, Patrick A. V. Hall, and John H. R. May, "Software Unit Test Coverage and Adequacy," ACM Computing Surveys, vol. 29, no.4, pp. 366-427, December, 1997

Manual Reviews

- Manual static analysis methods
- Most can be applied at any step in the lifecycle
- Have been shown to improve reliability, but
  - often the first thing dropped when time is tight
  - labor intensive
  - often done informally, no data/history, not repeatable

Different Kinds of Manual Reviews

- Reviews
  - author or one reviewer leads a presentation of the artifact
  - review is driven by presentation, issues raised
- Walkthroughs
  - usually informal reviews of source code
  - step-by-step, line-by-line review

Different Kinds of Manual Reviews

- Software inspections
  - formal, multi-stage process
  - significant background & preparation
  - led by moderator
  - Many variations of this approach
- Cleanroom
  - formal review process
  - Plus, statistical based testing

Software Inspections

- Developed by Michael Fagan in 1972 for IBM
- 3-5 participants
- 5 stage process with significant preparation

Inspections participants (4 to 6 people)

- MODERATOR - responsible for organizing, scheduling, distributing materials, and leading the session
- AUTHOR - responsible for explaining the product
- SCRIBE - responsible for recording bugs found
- PLANNER or DESIGNER - author from a previous step in the software lifecycle
- USER REPRESENTATIVE - to relate the product to what the user wants
- PEERS OF THE AUTHOR - perhaps more experienced, perhaps less
- APPRENTICE - an observer who is there mostly to learn

Inspection Process

- Planning
  - done by author(s)
    - Prepare documents and an overview
      - explain content to the inspectors
  - done by moderator
    - Gather materials and insure that they meet entry criteria
    - Arrange for participants
      - assign them roles
      - insure their training
    - Arrange meeting

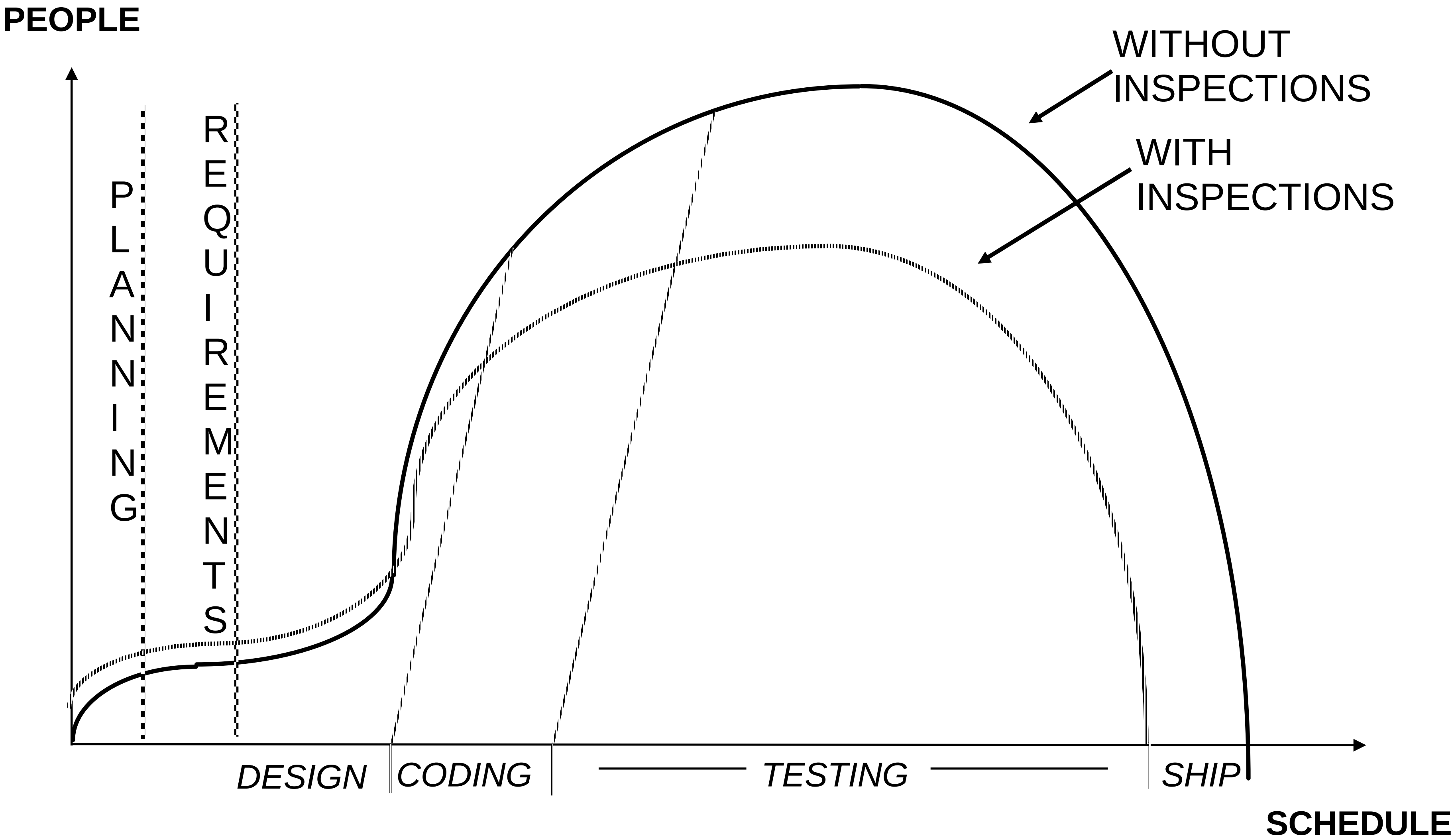
Fagan Inspection Process (cont.)

- Preparation
  - Participants study material
- Inspection
  - Find/report faults (Do NOT discuss alternative solutions)
- Rework
  - Author fixes all faults
- Follow-Up
  - Team certifies faults fixed and no new faults introduced

Fagan Inspection-general guidelines

- Distribute material ahead of time
- Use a written checklist of what should be considered
  - e.g., functional testing guidelines
- *Criticize product, not the author*

People Resource versus Schedule



* Fagan, 1986

Experimental Results

- software inspections have repeatedly been shown to be cost effective
- increases front-end costs
  - ~15% increase to pre-code cost
- decreases overall cost

IBM study

- doubled number of lines of code produced per person
  - some of this due to inspection process
- reduced faults by 2/3
- found 60-90% of the faults
- found faults close to when they were introduced
  - The sooner a fault is found the less costly it is to fix

Why are inspections effective?

- knowing the product will be scrutinized causes developers to produce a better product
  - Hawthorne effect
- having others scrutinize a product increases the probability that faults will be found
- walkthroughs and reviews are not as formal as inspections, but appear to also be effective
  - hard to get empirical results

What are the deficiencies?

- tend to focus on error detection
  - what about other "ilities" -- maintainability, portability, etc.
- not applied consistently/rigorously
  - inspection shows statistical improvement
- human intensive and often makes ineffective use of human resources
  - e.g., skilled software engineer reviewing coding standards, spelling, etc.
  - Lucent study .5M LOCS added to 5M LOCS required ~1500 inspections, ~5 people/inspection
  - No automated support

Experimental Evaluation

- There have been many studies that have demonstrated the effectiveness of inspections
- Indirect effect--Developers involved in inspections improve their skills by observing superior artifacts and skilled reviewers
- Recent studies trying to determine what aspects of inspections are effective
  - Provide insight into
    - Ways to improve the process
    - Ways to reduce the cost

Experimental evaluation of inspections

- Adam Porter, Harvey Siy, Audris Mockus, Lawrence G. Votta, *Understanding the Sources of Variation in Software Inspections*, UMd Technical Report, Jan 1997
- A. Porter,H.P. Siy, C.A. Toman, L.G. Votta, *An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development*, IEEE Transactions on Software Engineering, 1997 23(6): 329-346, June 1997.

Experimental Design

- Lucent compiler project for 5ESS telephone switching system, 1994
  - 55K new lines; 10K reused lines
- Inspectors chosen from 11 professionals
  - At least 5 yrs. experience
  - Inspection training
- Modified inspection process and measured effect
  - Defects found
  - Interval: time from when artifact is ready to be reviewed until it is repaired
- 88 inspections overall

Variants to consider

- Team size
  - Difference between teams of 1, 2, or 4 on # defects found
- Inspection interval
  - Calendar time to complete an inspection
- Single or multi-session inspections
  - N-fold --N teams doing N independent inspections
  - Multiple phased inspections focus on different concerns at each phase
- Individual or group centered
  - Is it necessary to actually have a meeting?

Alternatives

- N sessions, with M people, repairing defects (R) between sessions or not (N)
  - Ns x Mp {R|N}

  - E.g., Considered
  - 1sX4p
  - 2sX2pN
  - 2sX2pR
  - 1sX2p
  - 2sX1pN
  - 2sX1pR

Hypotheses

- Large teams ==>
  - No increase in defects found
  - Increase in interval
- Multiple-session inspections ==>
  - Increase in defects found
  - Increase in interval
- Correcting defects between sessions ==>
  - Increase in defects found
  - Increase in interval
    - Terminated this process early since it was too costly

Results from the experiment

- Can use 2 person teams
  - Can use a small team w/o jeopardizing the effectiveness
  - 1sX1p < 1sX2p, but 1sX2p = 1sX4p
- Number of sessions did not impact effectiveness
  - More sessions increase interval but not defects found
  - Can use one session
- Repairs between sessions did not significantly improve defect detection but did increase time interval

Use single sessions inspections with 2 person teams

Results from the experiment

- **Effort increases with the number of people, independent of the process (e.g., number of sessions)**

Results from the experiment--independent of the process used

- Only 13% of reviewer issues are real defects
  - Meetings suppressed 26% of the superfluous issues
- Meetings lead to the detection of 30% of all the defects
  - Others found by individuals before the meeting

Cleanroom: S/W development process

- Mills, Harlan D., Michael Dyer, and Richard C. Linger
- Originally proposed by H. Mills in the early 80's
- H. Mills had previously proposed the chief programmer team concept

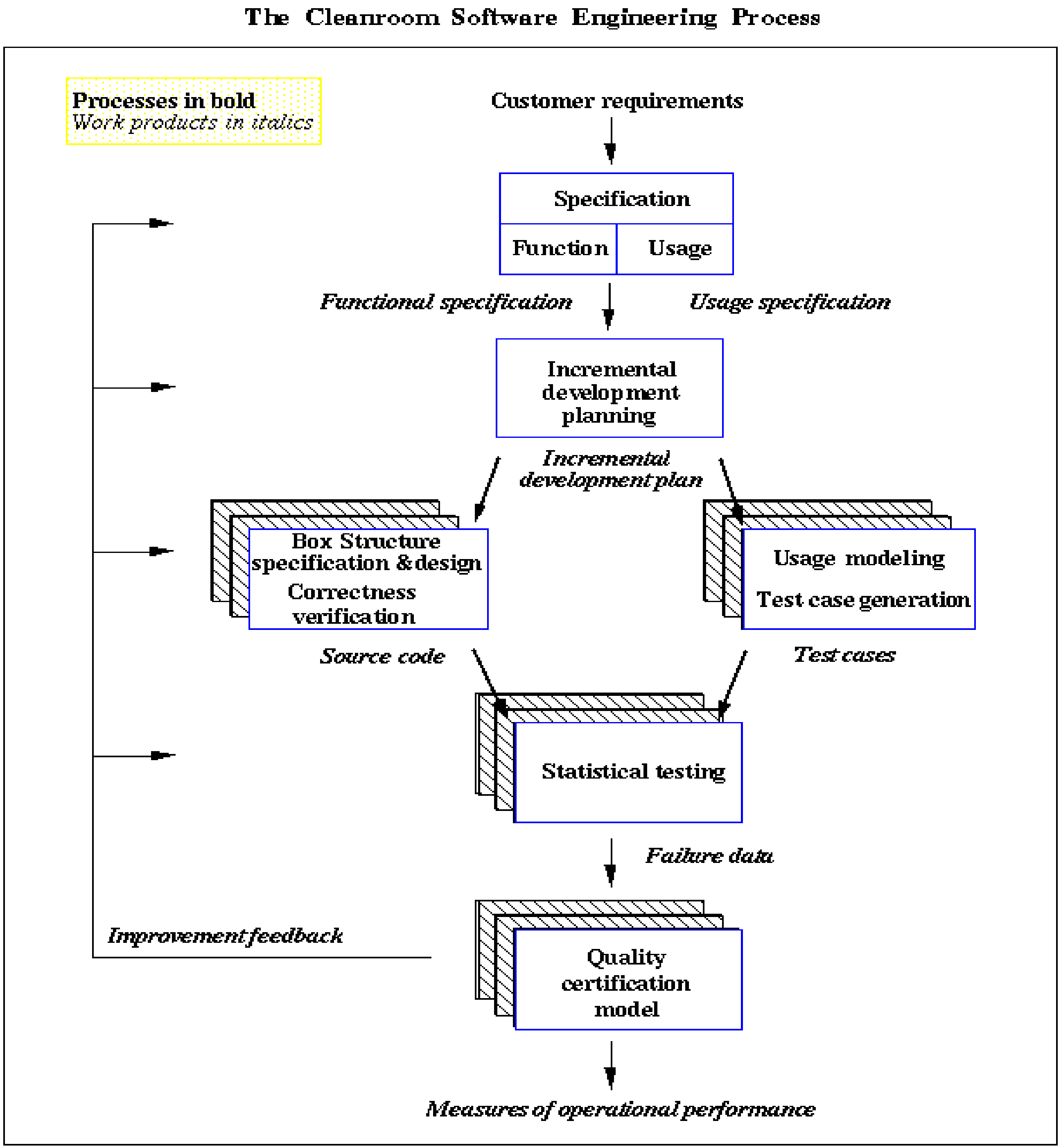
Major contributions

- Incremental development plan
  - Instead of a pure waterfall model
  - Incrementally develop subsystems
- Use formal models during specification and design
  - Structured specifications
  - State machine models
- Use informal verification instead of testing
- Independent, statistical based testing
  - Based on usage scenarios derived from state machine models

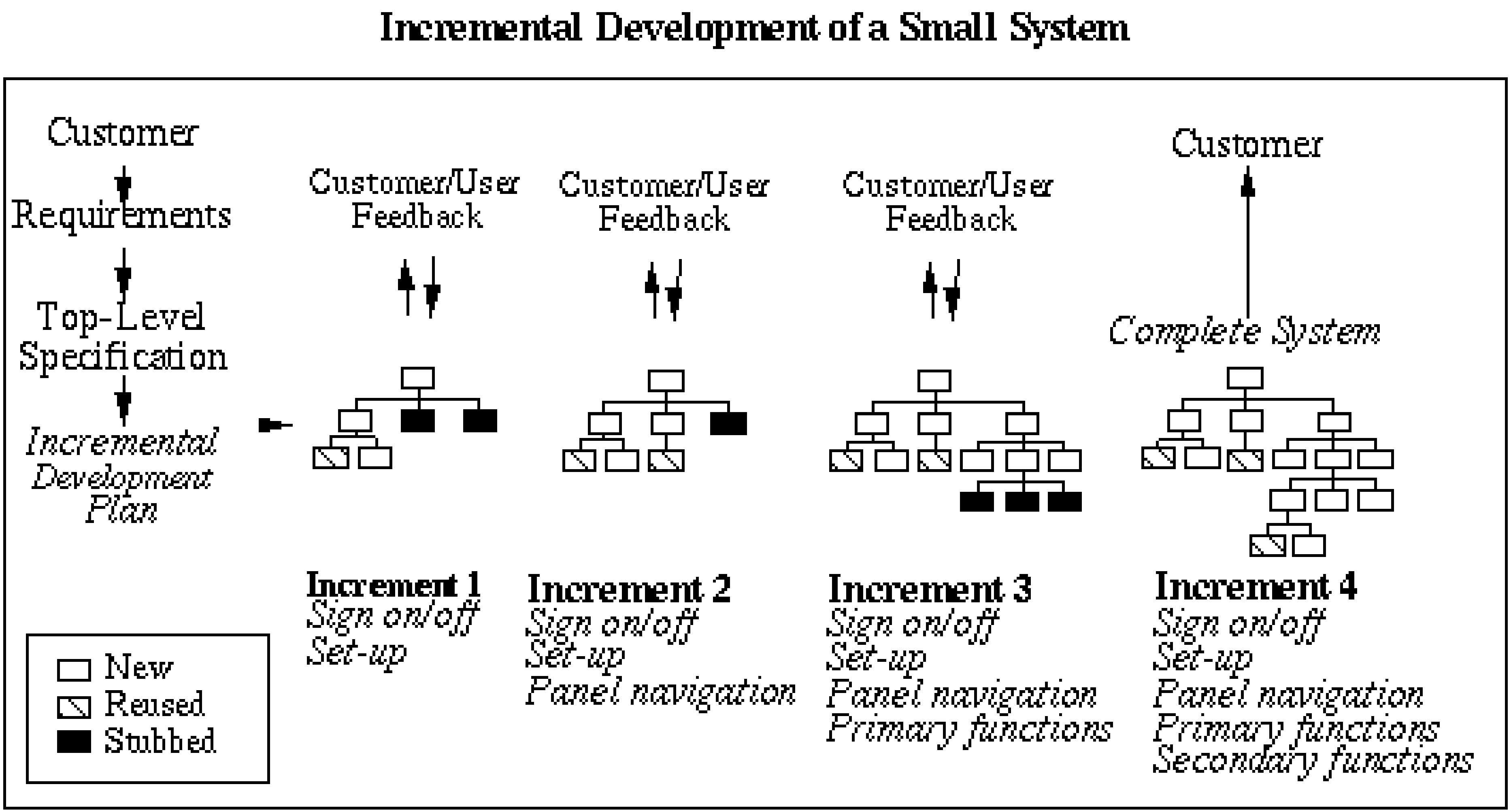
Cleanroom Process

- Incremental Planning
- Box Structure Specification and Design
- Usage Specification
- Correctness Verification
- Usage Modeling
- Statistical Testing
- Reliability Estimation
- Process Control and Improvement

Cleanroom


The Cleanroom Software Engineering Process
Processes in bold
Work products in italics
Customer requirements
Specification
Function
Usage
Functional specification
Usage specification
Incremental development planning
Incremental development plan
Box Structure specification & design
Correctness verification
Usage modeling
Test case generation
Source code
Test cases
Statistical testing
Failure data
Improvement feedback
Quality certification model
Measures of operational performance

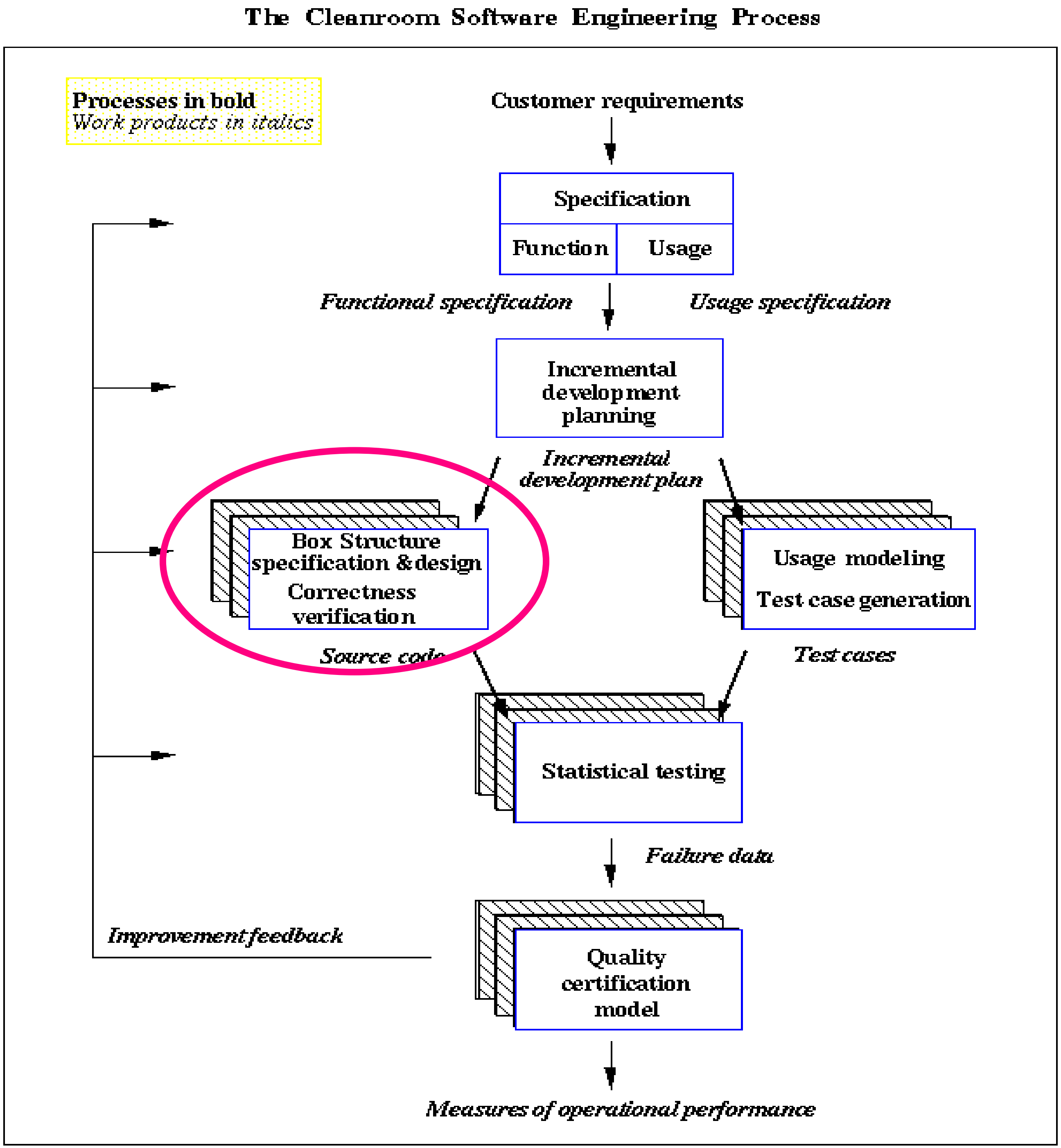
Cleanroom


Incremental Development of a Small System
Customer
Requirements
Top-Level Specification
Incremental Development Plan
Customer/User Feedback
Customer/User Feedback
Customer/User Feedback
Customer
Complete System
Increment 1
Sign on/off
Set-up
Increment 2
Sign on/off
Set-up
Panel navigation
Increment 3
Sign on/off
Set-up
Panel navigation
Primary functions
Increment 4
Sign on/off
Set-up
Panel navigation
Primary functions
Secondary functions
New
Reused
Stubbed

Benefits of Incremental Development

- Early feedback
  - on part of the system, at least
- Improves morale
  - Something tangible is working
- Improves chances of releasing on time
  - Incorporate high priority capabilities first
  - Low priority capabilities may miss release
  - Detect problems with high priority capabilities early
    - More time to react

Cleanroom



The Cleanroom Software Engineering Process
Processes in bold
Work products in italics
Customer requirements
Specification
Function
Usage
Functional specification
Usage specification
Incremental development planning
Incremental development plan
Box Structure specification & design
Correctness verification
Usage modeling
Test case generation
Source code
Test cases
Statistical testing
Failure data
Improvement feedback
Quality certification model
Measures of operational performance

Box Structure Specification and Design

- Refinement approach to developing the design
- Black Box
  - High level functional specification
    - Input and output specification
  - Interface specification of major components
- State Box
  - State transition diagram
  - Shows high level functioning of each component
- Clear Box
  - Low level design
    - Data structures and algorithms

Verification

- ensure that a software design is a correct implementation of its specification
- team verification of correctness takes the place of individual unit testing
- benefits
  - intellectual control of the process
  - motivates developers to deliver fault-free code
  - verification is a form of peer review
  - each person assumes responsibility for and derives a sense of ownership in the evolving product
- every person must agree that the work is correct before it is accepted -> successes are ultimately team successes, and failures are team failures

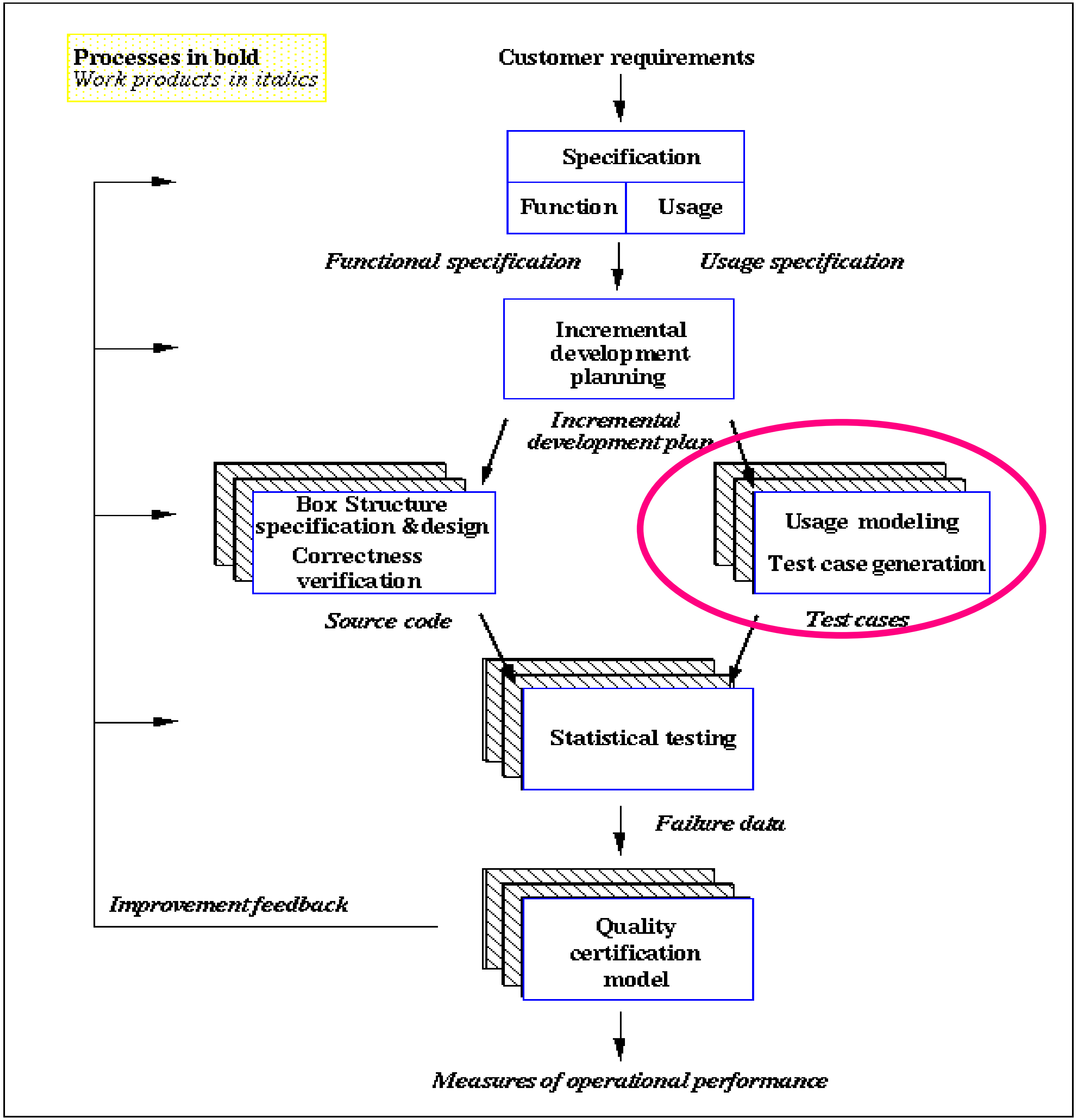
Verification

- team applies a set of correctness questions
- correctness is established by group consensus if it is obvious
- by "formal" proof techniques if it is not

- Form of inspection

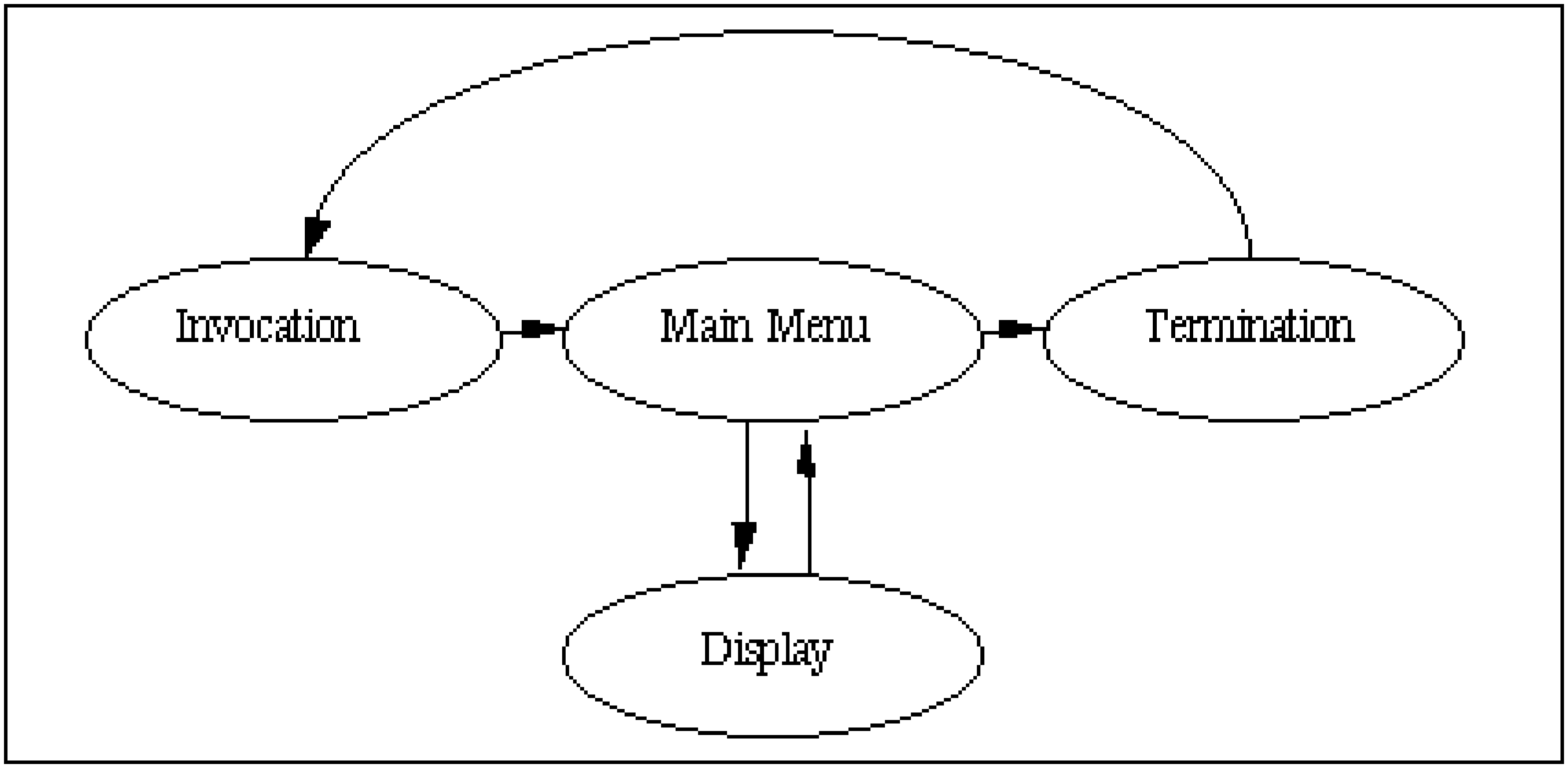
Cleanroom

The Cleanroom Software Engineering Process



Usage specification

Graphical Usage Model of a Simple System



Statistical Testing

- Generation of Test Cases
  - each test case is a walk through the usage model
    - invocation->termination
  - test cases constitute a "script" for use in testing
    - applied by human testers or used as input to an automated test tool
- Stopping Criterion for Testing
  - target level of estimated reliability are achieved
  - Usage coverage achieved

Experimental evaluation of cleanroom

Selby, R.W., V.R. Basili, and F.T Baker,. "CleanroomSoftware Development: An Empirical Evaluation," IEEE Transactions on Software Engineering, September 1987, pp.1027—1037

Not assigned

Experimental design

- 15 three person teams, developed the same software system
  - 88-2300 LOCs
  - 10 teams--cleanroom
  - 5 teams used ad hoc techniques

Experimental results (in a nutshell)

- Cleanroom
  - 6 of the 10 cleanroom teams completed ~90% of the project
  - Met requirements better
  - Had more operational test cases
  - Met milestones (compared to only 2 of the traditional teams)
  - 86% missed traditional testing and debugging
  - 81% claimed they would use the technique again

Comments on Experimental Results

- Not clear what aspects of cleanroom led to the observed improvements
- Need a more careful experimental evaluation

Case Studies

Project	**Application, Size**	**Quality** * (Errors/KLOC)	**Productivity**
Ericsson OS-32	OS for telephone switch, 350 KLOC	1	1.7 improvement in development 2X improvement in testing
Hewlett-Packard	Windows application, 3.5 KLOC	1.4	
IBM AO Expert	decision support, 107 KLOC	2.6	486 LOC/PM
IBM COBOL SF	language, 85 KLOC	3.4	5X improvement
IBM Tucson 3490E Model C SCSI-2	SCSI adapter for tape drive, 86 KLOC	1.2	
US Air Force STARS Demo Project	command and control, 332 KLOC	available 10/95	available 10/95
US Army Picatinny Arsenal I-MBC	mortar ballistics computer, 75 KLOC	0.8	4.8X improvement
US Naval Coastal Systems Station AN/KSQ1	amphibious assault directions system, 3.5 KLOC	2.5	

* Error rates are from first execution through completion of certification testing.

Remember

- Typical programmer produces about 30 LOCs a day
  - Ranges between 10-100 LOCs
- Faults/KLOC
  - Ranges between 3-10 faults/KLOC

Note: faults are hard to measure

- Each syntactic change
- Each misunderstanding

Comments on Cleanroom

- Very Visionary
  - Block structure design and usage scenarios supported by UML
  - Provides early visibility into the product
- Often misinterpreted to mean no testing,instead of systematic, careful testing
- Pure Cleanroom requires considerable discipline and is human intensive
- Some variant of cleanroom is often used in practice

Concluding remarks on Manual Reviews

- Some form of careful manual inspection seems to improve the quality of a s/w system and to improve productivity
  - Not clear if the benefits of cleanroom are from the inspection aspects of the process or other aspects or some combination
- When deadlines are tight, it is very hard to commit the resources for such a labor-intensive task
- Some automated support could help to reduce the manual effort involved
  - Would this be effective or counter-productive?