Course "Softwareprozesse"

Cleanroom Software Engineering

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik

- Principles
- Empirical results
- Typical practices
- Stepwise refinement
  - box structures, verification
- Statistical testing
  - Usage modeling
  - Hints for practice
- Cleanroom and Agile

Learning objectives

- Cleanroom is primarily a "No defects!" attitude
- It depends on careful argumentation, checked in a small team
- It measures reliability by statistical testing
  - at least in principle – this is quite difficult in practice.
- It is combinable with agile development practices
  - if and where precise specifications are available.

Cleanroom classification and goals

- Proposed by Harlan D. Mills, IBM, since 1980
  - 'Cleanroom' stands for defect prevention instead of defect elimination

Harlan Mills

Goal:

- High, quantified reliability at low cost

Classification:

- Cleanroom is a development approach and a management approach
  - but specialized -- not an entire software process model

Context:

- Whenever precise specifications can be written early
  - For new development, maintenance, and reengineering
  - Independent of technology, but requires a mature process

Cleanroom principles

Cleanroom development principle:

- Development teams strive to produce products without any defects
  - by careful design and development
  - by verification and review
  - but *not* by testing

Cleanroom testing principle:

- The purpose of testing is *measuring* the reliability of the product
  - not improving the reliability

Cleanroom management principle:

- Team-based practices limit the scope of human fallibility and allow for continuous improvement

Empirical results (1): IBM Cobol SF

Freie Universität Berlin

- R.C. Linger, H.D. Mills: "*A Case Study in Cleanroom Software Engineering: The IBM COBOL Structuring Facility*",
  - *12th Intl. Computer Science and Applications Conf.*, Oct. 1988.

- Project developing "Cobol Structuring Facility" COBOL/SF
  - A program analyzer/translator (written in PL/1) for converting Cobol code with GOTOs into structured Cobol code
  - 52 KLOC modified/added to existing 40 KLOC base product
- Overall productivity: +400%
- Overall defect density: 3.4 defects/KLOC
- Field-testing defects: 10 (only 1 of them major)

- The defect reduction is the main reason for the huge improvement in productivity
  - Testing such a system is very laborious

Empirical results (2): Ellemtel/Ericsson OS32

- L.G. Tann: *"OS32 and Cleanroom"*
  - *1st Annual European Industrial Symposium on Cleanroom Software Engineering, Copenhagen*, Denmark, 1993, pp. 1-40.

- Project developing an operating system for telephone switching systems
  - 73 people staff, 33 months duration
  - 350 KLOC resulting software size (14 LOC/PM)
- Development productivity: +70%
- Testing productivity: +100% (tests per hour)
- Testing defect density: 1 defect/KLOC

- These are very big improvements, considering this was a mature development organization already.

Empirical results (3): Controlled experiment

Freie Universität Berlin

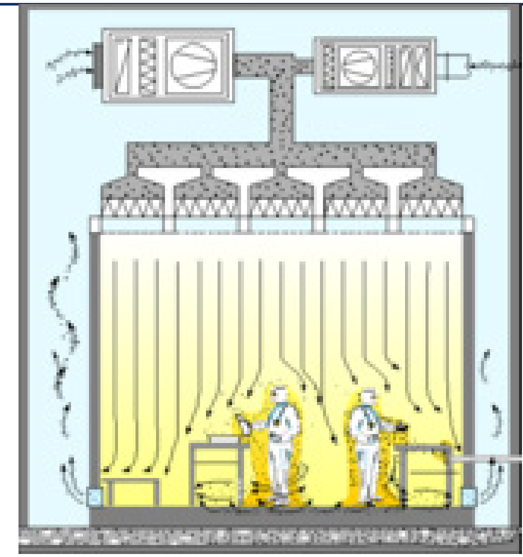
- R. Selby, V. Basili, F. Baker: "*Cleanroom Software Development: An Empirical Evaluation*"
  - IEEE Transactions on Software Engineering, 13(9), Sept. 1987
- A controlled experiment:
  15 teams (10 Cleanroom, 5 conventional) of 3 student developers (w. prof. experience). Each develops the same SW
  - electronic messaging system: duration 6 weeks, 4 milestones,
  - resulting size 800 to 2300 LOC of *Simple-T* code
- Results:
  - The Cleanroom teams developed more functionality
  - All Cleanroom teams kept all milestones,
    only 2 of the 5 others did
  - The Cleanroom programs were less complex (control flow)
    and had better annotation
  - The Cleanroom programs had significantly fewer test failures
  - 86% of the developers missed testing (quality was not affected)

Typical Cleanroom techniques

Freie Universität Berlin

Small teams

- High motivation, close cooperation, efficiency
  - **"Defects are not acceptable!"**
- Parallel development
  - Strict modularization has to be done at specification time
- Exact specification
  - All partial specifications are precise and self-contained



Physical cleanroom

Strict separation of development and testing

- Development teams
  - Development teams are strictly forbidden to perform any testing
- Test teams
  - Test teams never modify programs

Typical Cleanroom techniques (2)

Exact specification

- Defect prevention
  - Precise specifications help avoid ambiguity defects
- Verification
  - During development, defects are continually searched for by comparing with specification
- Specif. languages: Z, VDM, box method, special grammars

Stepwise refinement with the box method

1. Specification (black box)
   - Describes WHAT without HOW
2. State description (state box)
   - Specification as a state machine (not always useful)
3. Process description (clear box)
   - Partial HOW: "Implementation", but may use further black boxes

Typical Cleanroom techniques (3)

Freie Universität Berlin

The key point!

Review/verification

- Performed for each refinement
  - State box and clear box
- Grounded in mathematics, performed as team discussion
  - Convincing argumentation, rarely formal mathematical proof
- Argument is formulated and verified during an inspection

Incremental development

- Initially, only basic functionality is developed

Statistical testing

- Usage modelling
  - Test cases are a random sample according to usage model
- Quantitative statement on reliability (certification)

Typical Cleanroom techniques: Note

- First and foremost, Cleanroom development is an **attitude**
  - So none of the above techniques is absolutely mandatory:

They can be driven to extremes

- for instance developers may be prohibited to even compile their code

They can (and probably should) be relaxed

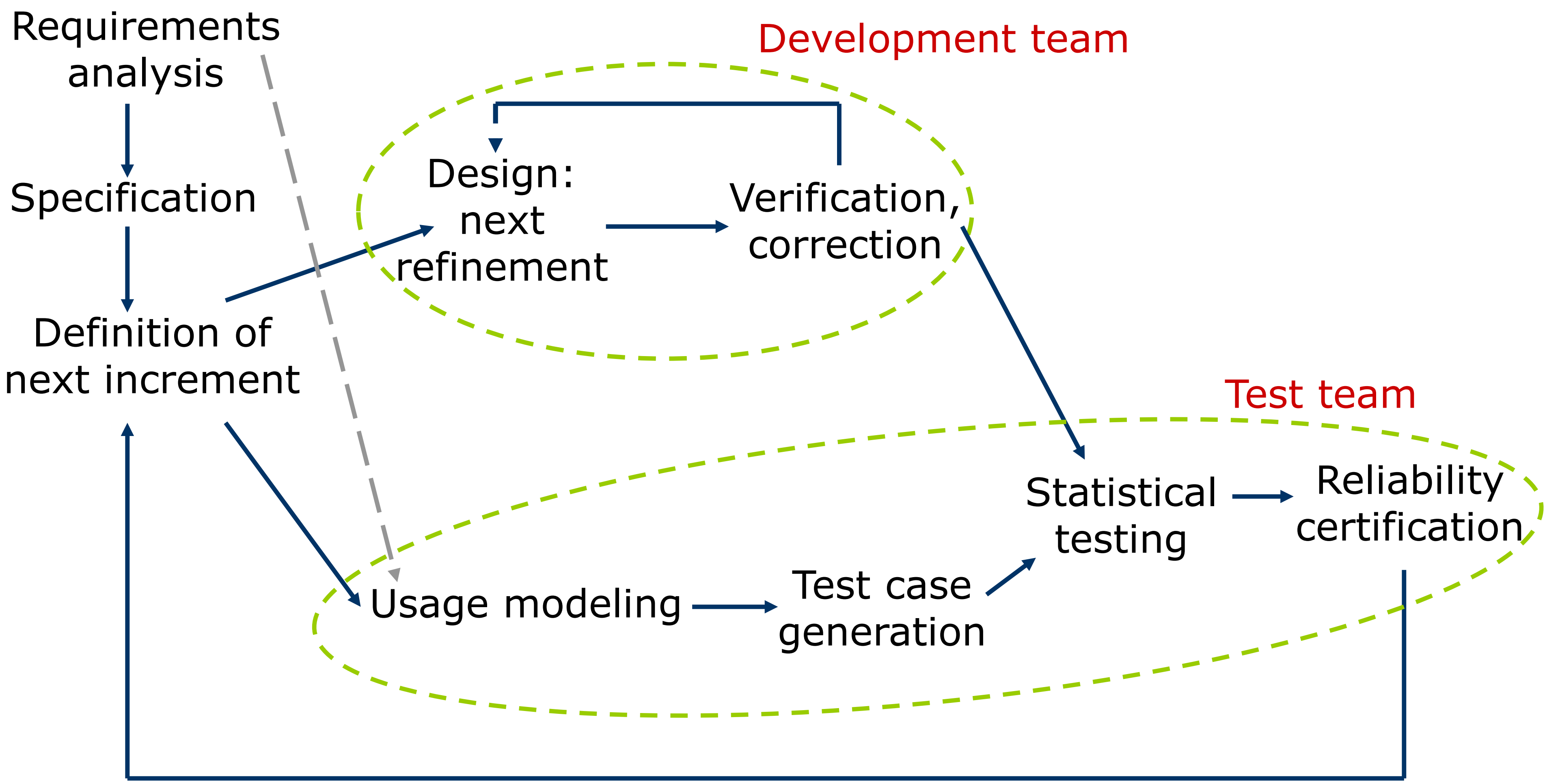
- for instance by performing defect testing before statistical testing

They can be exchanged for others

- for instance by driving development in some other way than by box refinement

M. Deck: Cleanroom Software Engineering Myths and Realities, 1997

Cleanroom process flow overview



Problems and Obstacles

Cleanroom is not suited if

- …formal specification is difficult
  - which is commonly the case for interactive systems
- …determining the correctness of test outputs is costly
  - but this is a problem for conventional development as well.
  - One could still do Cleanroom without reliability certification
    - by leaving out statistical testing

Necessary preconditions:

- Highly trained software engineers
  - Others cannot create reliable verification arguments
- Mature software process (well-defined, disciplined)
  - Immature processes will lack the necessary discipline and control

Specification and design with box structure

- Define black box:
  - define output based on input history

- Define state box [perhaps]:
  - define states for the representation of input history
  - reformulate black box (may introduce several new black boxes)
  - verify reformulation: state box must be equivalent to black box

- Define clear box:
  - define data abstraction for state data
  - reformulate state box
    (may introduce several new black boxes)
  - verify reformulation:
    clear box must be equivalent to state box

Continue with black boxes of the refinement

Trivial refinement example

Freie Universität Berlin

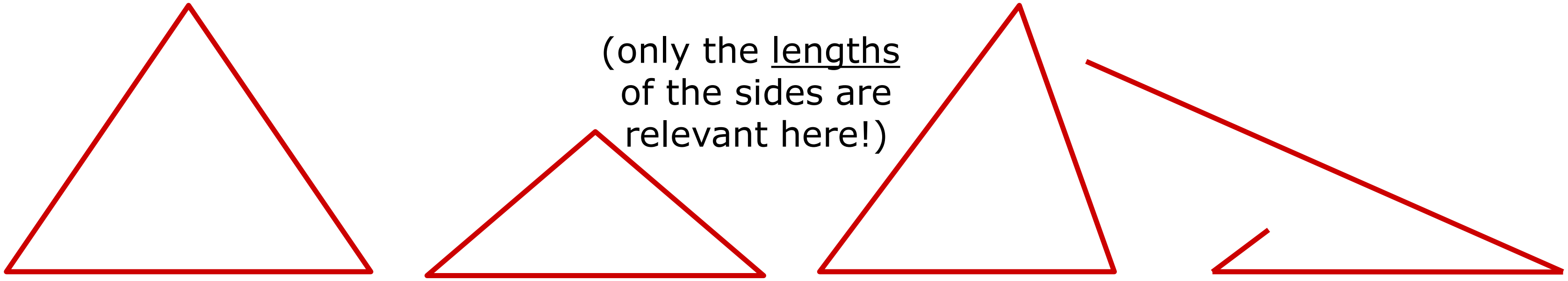
trivial example, for illustration only

- ***black box 1**:* triangleType(a, b, c)

precondition: a, b, c are positive, real numbers

postcondition:

return EQUILATERAL / ISOSCELES / OTHER / NO_TRIANGLE
⇔
the triple (a, b, c) is side lengths of an equilateral / non equilateral isosceles / non isosceles triangle / cannot be side lengths of a triangle



(only the lengths of the sides are relevant here!)

Refinement example (2)

(simple function example, so no state box is needed)

- *clear box 1*: triangleType(a, b, c)

IF allSidesSatisfyTriangleInequation(a, b, c)

THEN return trueTriangleType(a, b, c)

ELSE return NO_TRIANGLE

- ***black box 2***: allSidesSatisfyTriangleInequation(a, b, c)

precondition: a, b, c positive, real numbers

postcondition: True if each side is shorter than the sum of the other two; else False

- ***black box 3***: trueTriangleType(a, b, c)

precondition: (a, b, c) are the side lengths of a triangle

postcondition: …

Refinement example (3)

- *verification clear box 1:*

(a, b, c) can form triangle ⇔

the two shorter sides x, y together are longer than the longest side z.

Hence, $z < x + y$ (i.e., "*side z satisfies triangle inequation*") is sufficient for diagnosing a triangle.

"*All sides satisfy triangle inequation*" is a stronger condition, hence also sufficient.

Is "*All sides...*" also necessary? Yes: If $z < x + y$ holds, $x < z + y$ and $y < x + z$ will hold even more strongly

Hence, clear box 1 is correct.

Refinement example (4)

- ***clear box 2***: allSidesSatisfyTriangleInequation(a, b, c)

return (a < b + c AND b < a + c AND c < a + b)

- *verification clear box 2:*

3 different side lengths a, b, c are tested (→ "triangle"),
tests are connected by 'AND' (→ "all sides"),
each test compares one side to the sum of the two others,
each comparison is by 'less than' (→ correct inequation).
Hence, the implementation appears to be correct

Refinement example (5)

- ***clear box 3**:* <u>trueTriangleType(a, b, c)</u>

IF a = b= c THEN return EQUILATERAL
ELSE IF a = b OR a = c OR b = c THEN return ISOSCELES
ELSE return OTHER

- *verification clear box 3:*

'Equilateral' is a special case of 'isosceles' and must therefore be tested first*, this is done here.

The test for 'equilateral' is correct.

The test for 'isosceles' must check 3 different pairs (correct), only one needs to be equal (connection with 'OR', correct)

'Other' is the only remaining case, must be 'ELSE' part. Correct.

Therefore clear box 3 is correct.

*This is the most difficult spot of the argument.

Statistical Testing and Certification

- Most software processes use defect testing
  - Goal: Find as many defects as possible,
    with as few test cases as possible
  - Testing concentrates on 'difficult' cases.
- Defect testing makes almost no statement about reliability.

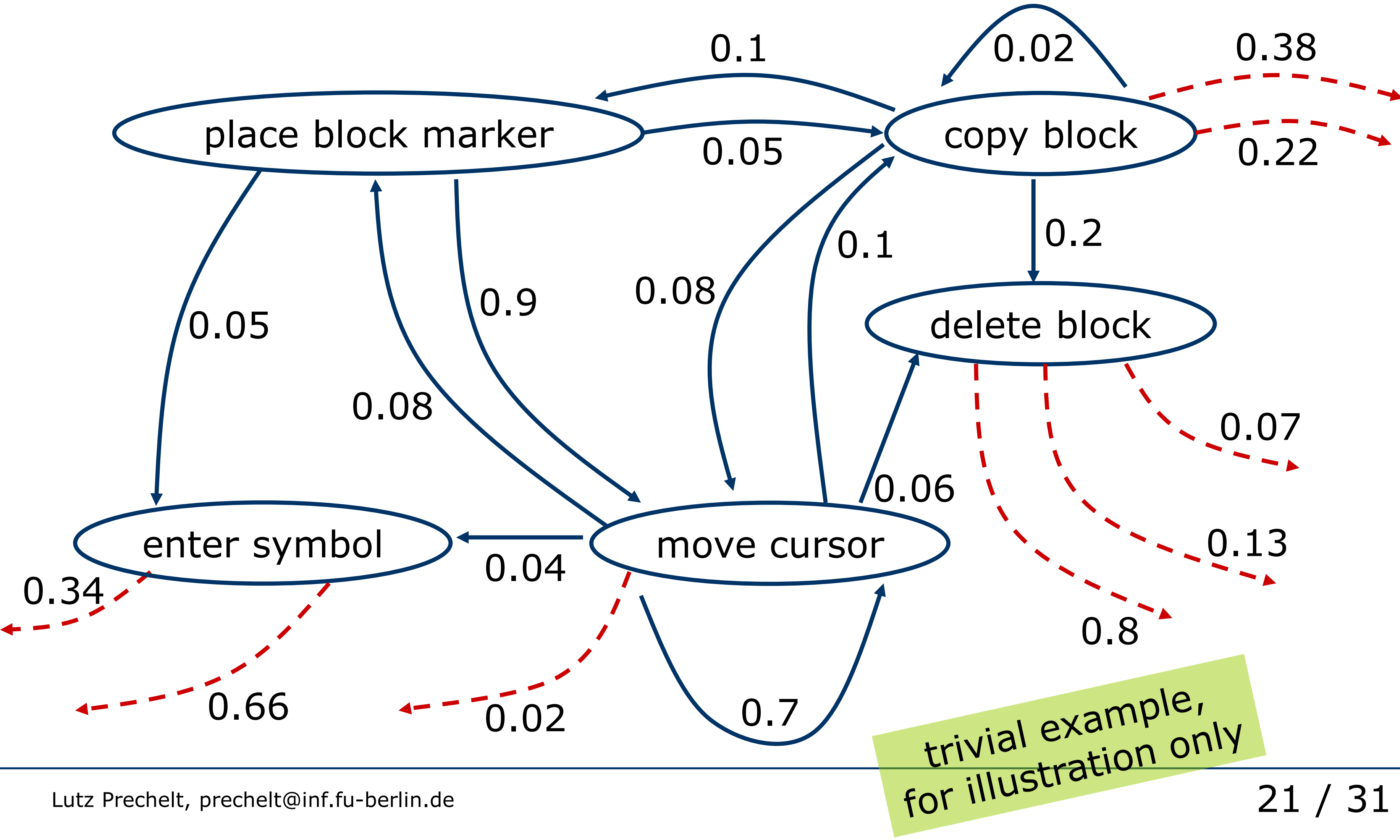
- In contrast, Cleanroom uses statistical testing
  - Goal: Quantify reliability; attitude like acceptance testing
  - Does <u>not</u> specifically aim to find defects
  - Testing reflects the frequency of 'typical' cases

- Basis: Usage modelling
  - Based on description of the usage profile (from requirements)
  - Mathematical description e.g. with Markov-chains
    (finite state space, discrete events)

Example: Excerpt from the usage model for a text editor

Freie Universität Berlin

Probabilistic state machine: States are actions, stochastic sequencing



Testing process

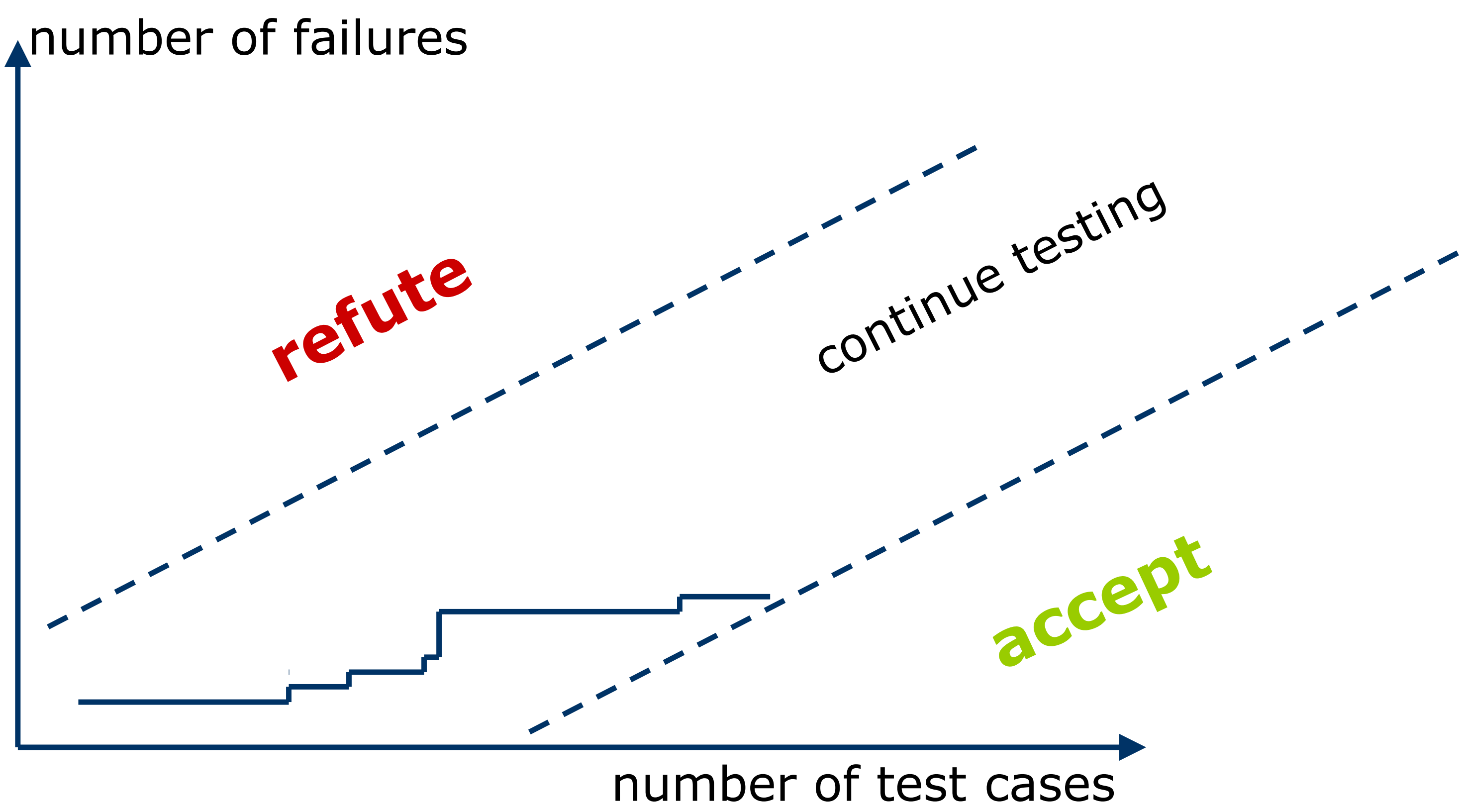
- Any number of test inputs can be generated automatically
  - based on the usage model
- Output correctness predicate?
    - like in machine learning for classification, labels are expensive!
  - Depends on application
  - Often only plausibility checking is possible
- Measure the intervals between failures
  - Terminate when sufficient reliability can be certified
  - Stop when insufficient reliability has been determined

Reliability certification

- The goal is a statement such as
  "MTTF(program) ≥ m with confidence K"
  - e.g. "With confidence 95% we can say that this program fails at most once every 2 000 000 steps"
  - MTTF: mean time-to-failure ("time" being the number of steps)
- Computed with statistical methods (binomial distribution)
- Problem:
  When I find and correct a defect,
  may I still use the data from the previous test runs?
  - Defect models and reliability growth models may allow this,
  - but then need to rely on assumptions
    - such as the non-introduction of new failures.
  - This is beyond the scope of this lecture.

Certification testing: basic idea

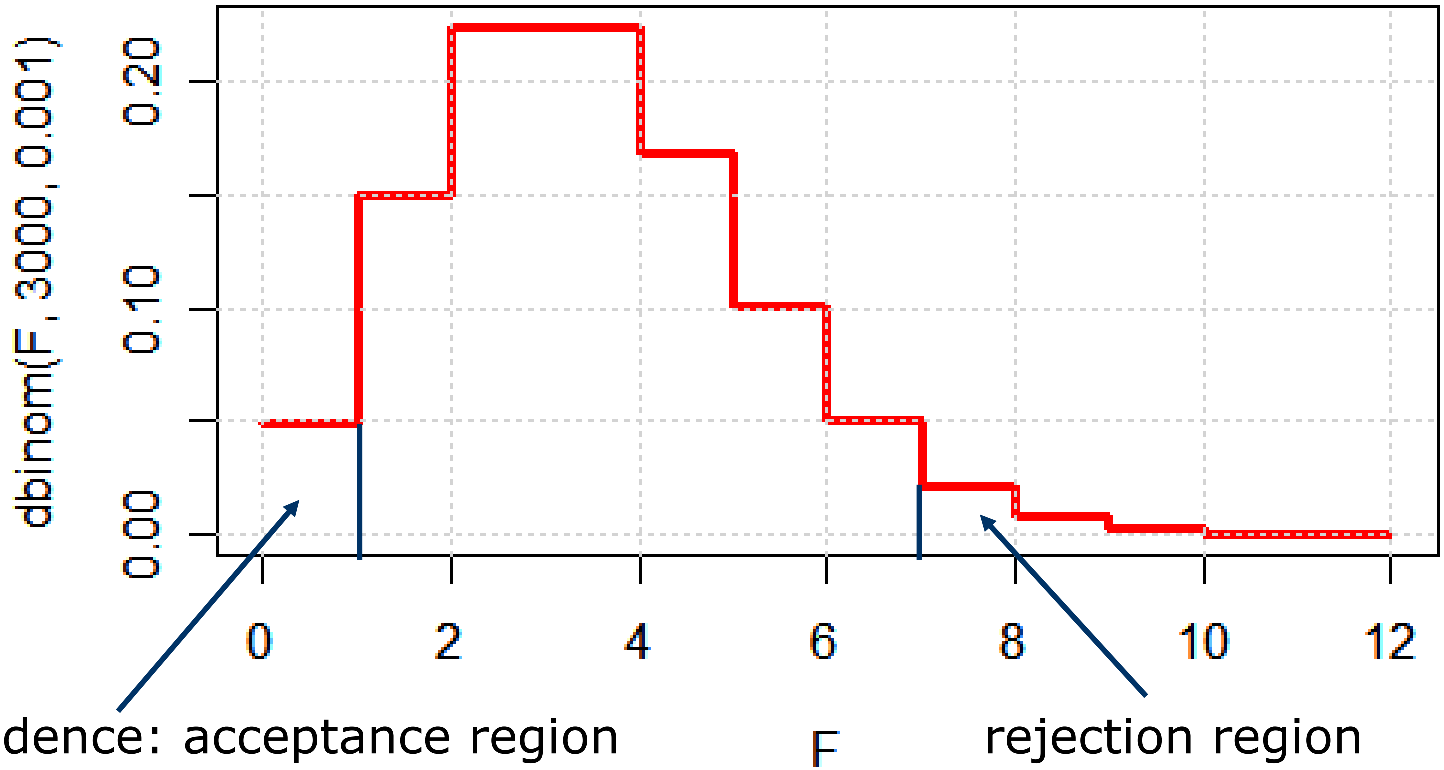
Schematic view! Details follow



Note that the up-steps are not vertical; they go 1 to the right as well.

Details: Binomial distribution

- Given an event (here: failure) with probability p (here: 0.001)
  - i.e. we want to certify 99.9% reliability (= 1-p)
- A binomial distribution describes the number F of failures to be expected during N runs (here: N=3000)



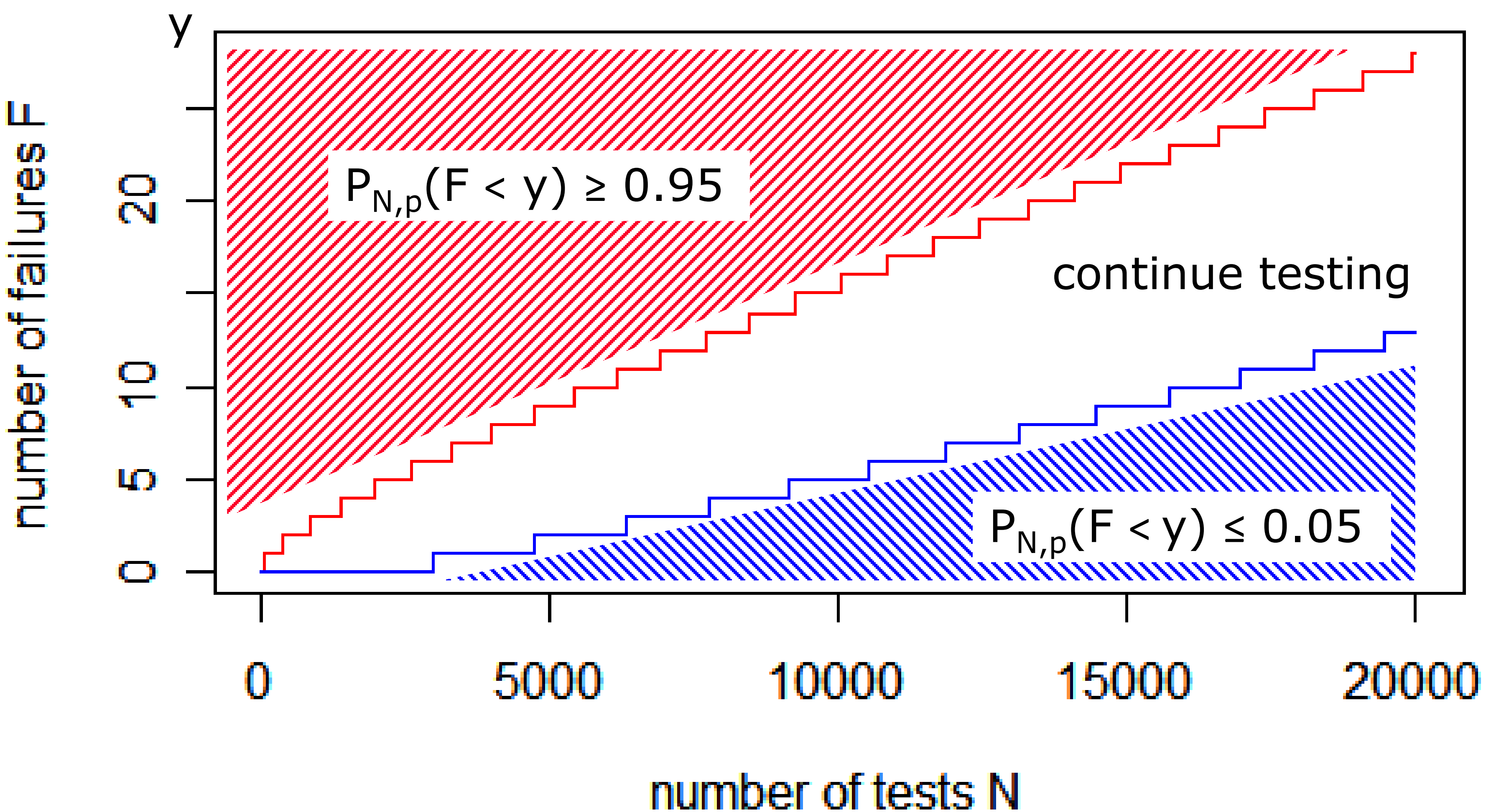
for 95% confidence: acceptance region

rejection region

http://mathworld.wolfram.com/BinomialDistribution.html

Certification testing

- Limit lines for binomial distribution (N trials, p=0.001)



Cleanroom testing in practice

M. Deck, J.A. Whittaker: "Lessons Learned from Fifteen Years of Cleanroom Testing", 1997

- One should integrate development and testing
  - Split has too much negative side-effects
    - adversarial thinking is bad, because collaboration helps
  - Cooperation adds value
    - e.g. operational profile helps SW design wrt real-time behavior
  - There *will* be some defects to be found and removed

- Statistical testing is very difficult
  - *huge* input spaces, so non-trivial usage models become very complicated

- One should adapt the techniques to the context
  - e.g. prototyping may be useful
  - e.g. coverage testing may be useful/required
  - regression testing is useful

Literature

* Richard Linger, Carmen Trammell:
  "Cleanroom Software Engineering Reference Model",
  Software Engineering Institute,
  Technical Report CMU/SEI-96-TR-022
  * detailled definition of the Cleanroom process

Cleanroom and Agile processes?

- Is Cleanroom SE compatible with an agile process?
  If no: Why not?
  If yes: When might the combination be useful?
  - Yes, there is no fundamental contradiction.
    - Although applying Cleanroom incrementally can be difficult.
  - May be useful for components with complex logic where reliability is critical.
- How to do it?
  - It may help to have only 1 refinement stage
    - Go from spec to final implementation in one step
    - and include all argumentation in the source code.

Summary: Cleanroom Software Engineering

- We studied Cleanroom for its ideas and basic attitude:
  "**Do not accept defects,**
  favor defect prevention over defect detection"
  - not as a software process to be used exactly as a whole;
  - useful where reliability matters a lot and specs are available

Key properties:

- Exact specification (important)
- Stepwise refinement with box-specification (replaceable)
- Verification during inspection (important, done by a team)
- Statistical testing based on usage model (difficult in practice)
- Reliability certification (ditto)

- **Result: very low defect rate, high productivity**

Thank you!

https://dilbert.com/strip/2002-06-11