N R Malotaux
Consultancy

Cleanroom software engineering

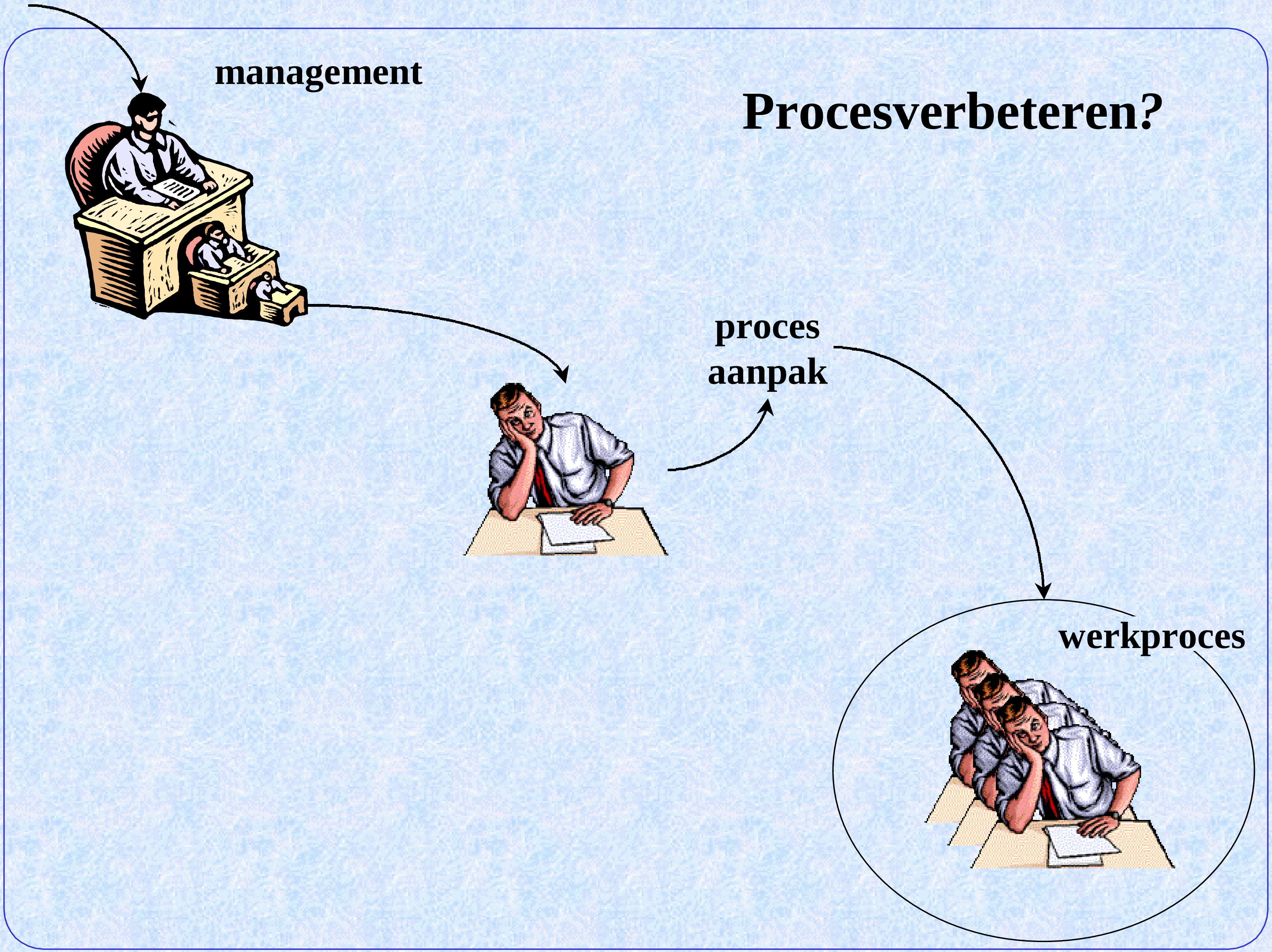
Ir. Niels Malotaux

030-228 88 68
niels@malotaux.nl
www.malotaux.nl/nrm

Niels Malotaux

- TU Delft – elektrotechniek 1974
- Meer dan 20 jaar ervaring in ontwikkeling van computers, embedded systemen en software
- Sinds 1998 “Kwaliteit op Tijd” adviseur
  - Begeleiden uitbesteder
  - Optimaliseren werkwijze R&D afdeling (DPI - HPI)
  - Optimaliseren werkwijze software afdeling (SPI)

Procesverbeteren?

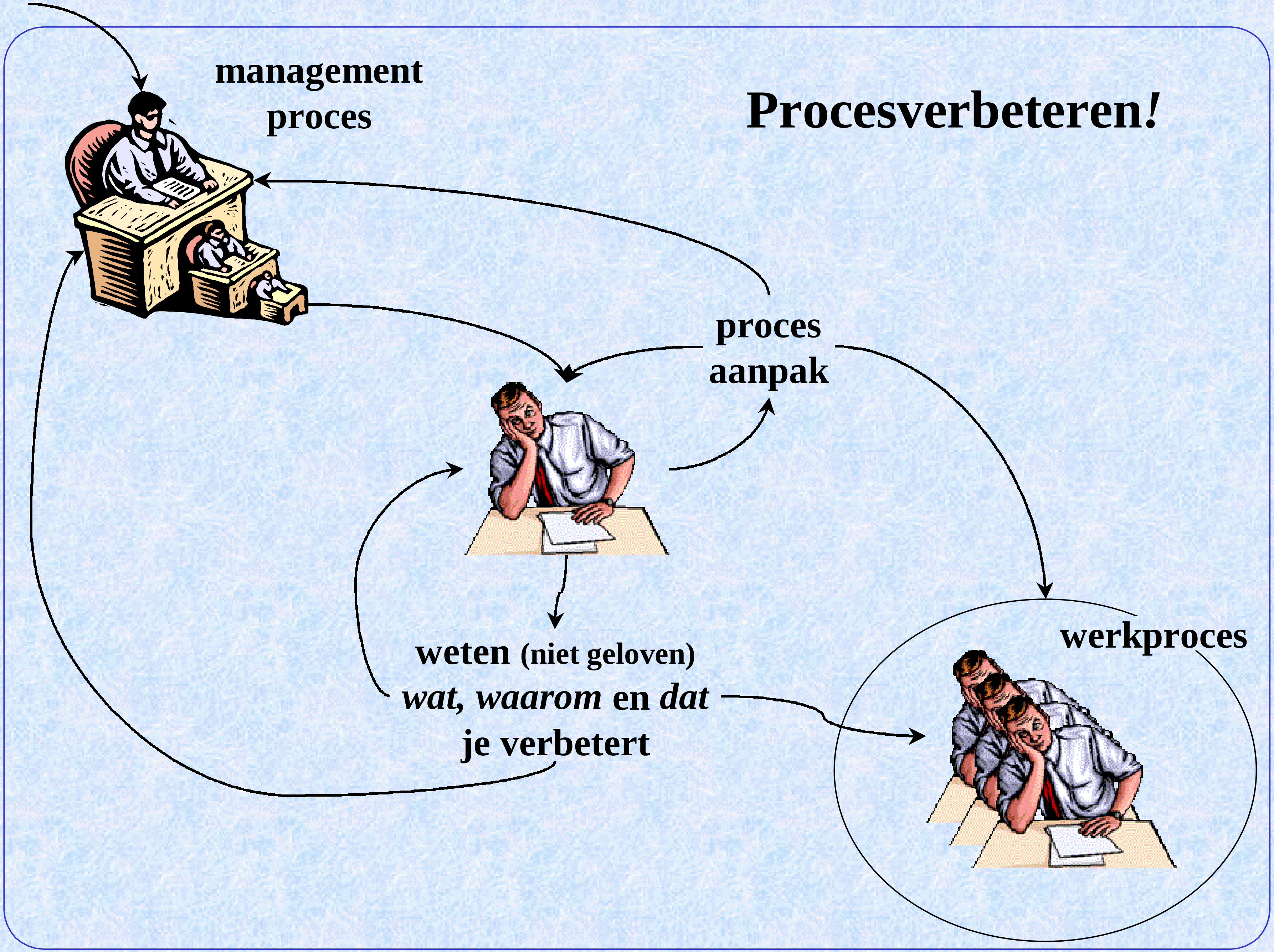


management

proces
aanpak

werkproces

Procesverbeteren!



management proces

proces aanpak

weten (niet geloven)
wat, waarom en *dat*
je verbetert

werkproces

Doelstelling

- Juiste kwaliteit
- Binnen afgesproken tijd en budget
- Op voor de medewerkers prettige wijze

Kwaliteit op tijd

Software processen

- CMM
  - Capability Maturity Model
- PSP → PPP
  - Personal Software Process
  - Personal Performance Process
- TSP
  - Team Software Process
- Cleanroom software engineering process

Inhibitors of high-quality results

- Poor project management
  - Inability to manage within constraints of
    - Cost
    - Schedule
    - Functionality
    - Quality
- Driving projects from schedule, not quality requirements
- Failure to control contents of requirements
- Failure to track defects and eliminate causes

The requirements paradox

- **Requirements should be stable**
- **Requirements always change**

Cleanroom: a SHIFT in practice from

- Individual craftsmanship to peer reviewed engineering
- Sequential development to incremental development
- Informal design to disciplined engineering specification and design
- Individual unit testing to team correctness verification
- Informal or coverage testing to statistical usage testing
- Unknown reliability to measured reliability

Cleanroom principles

- Incremental development
  - User verifyable increments
- Team organisation
  - 4~8 people
- Formal methods of specification and design
  - Level of formalism varies even within project
- Intense review
  - Mathematical proof of correctness
  - Verifying individual control structures
- No unit test
  - No testing of infinite number of paths
- Statistical testing as reliability measurement
  - Testing is not suitable for bug-hunting

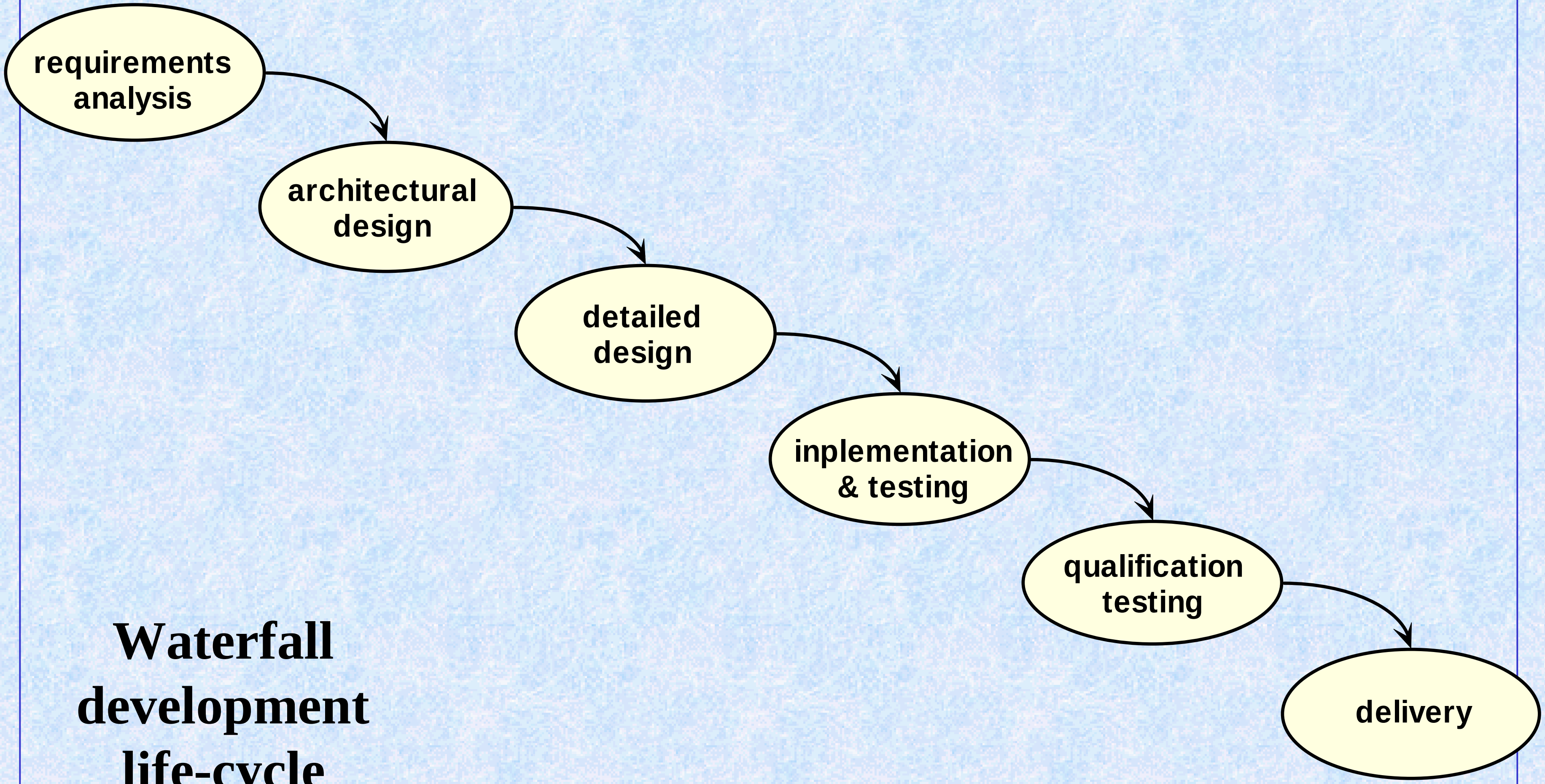
Cleanroom benefits

- **Zero failures in field use**
- **Short development cycles**
- **Long product life**

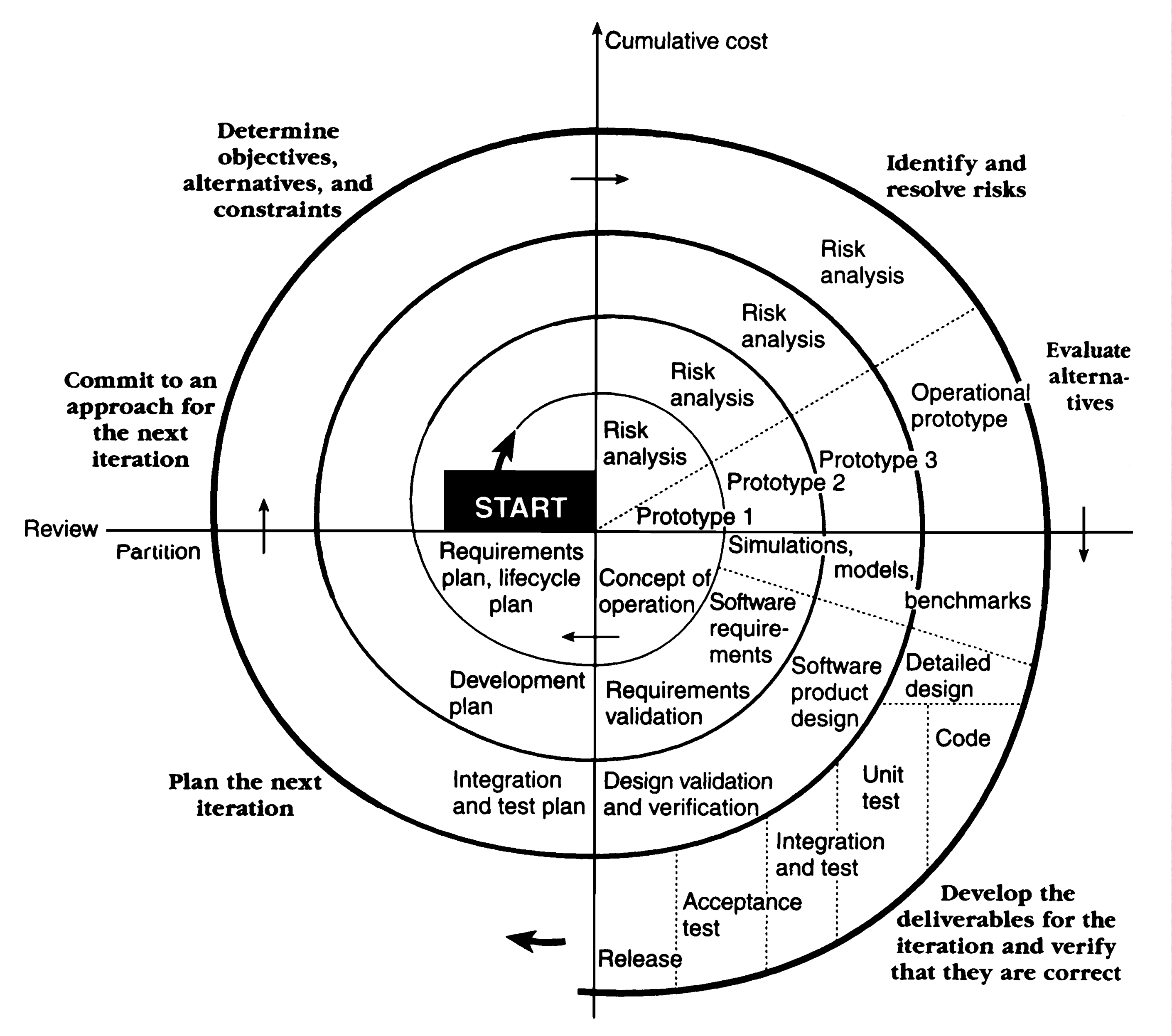
Quality is cheaper

Cleanroom principles

- Incremental development
  - User verifyable increments
- Team organisation
  - 4~8 people
- Formal methods of specification and design
  - Level of formalism varies even within project
- Intense review
  - Mathematical proof of correctness
  - Verifying individual control structures
- No unit test
  - No testing of infinite number of paths
- Statistical testing as reliability measurement
  - Testing is not suitable for bug-hunting



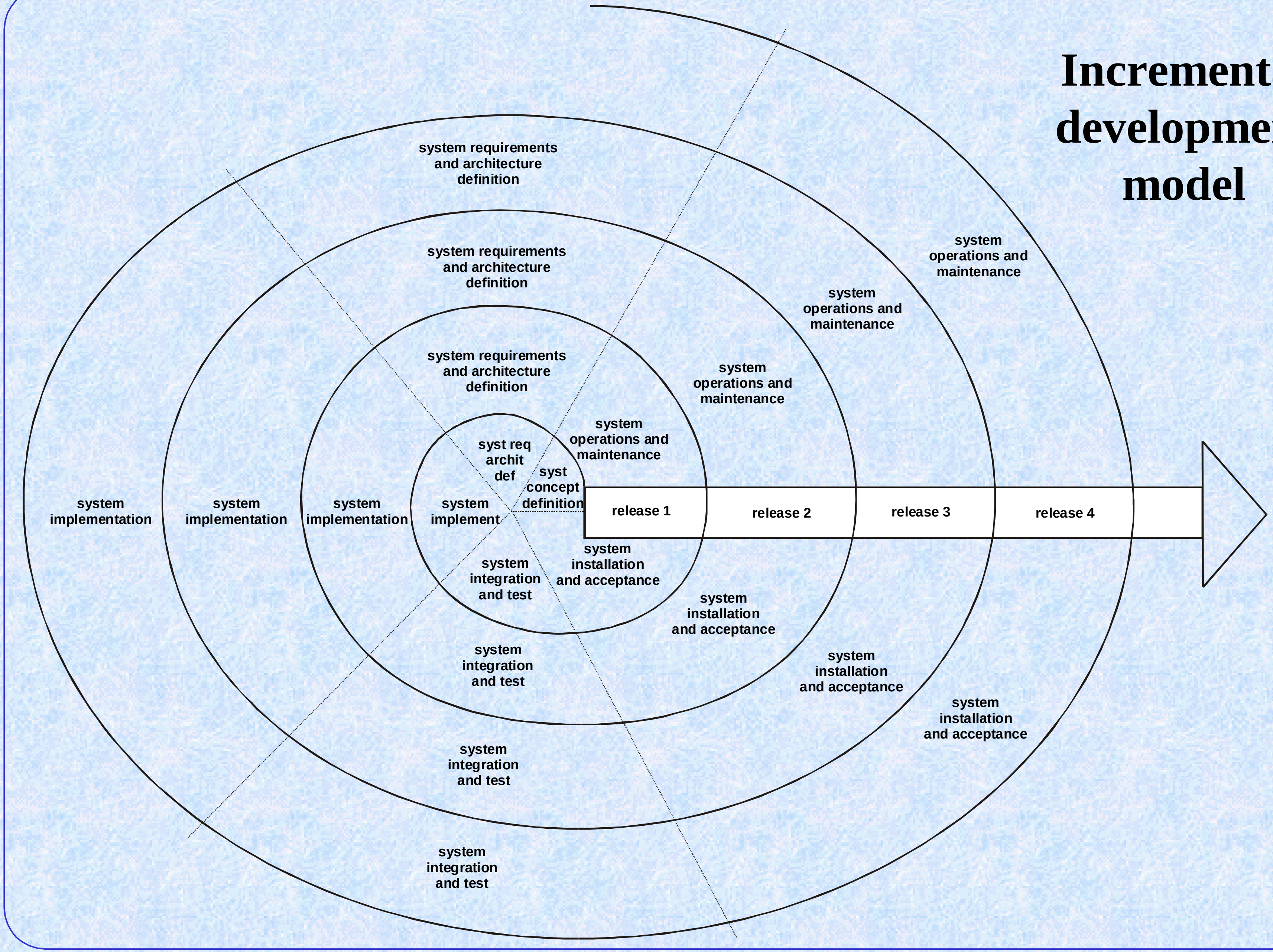
Waterfall development life-cycle model



Spiral Process model

(Boehm 88)

Incremental development model



system requirements and architecture definition
system requirements and architecture definition
system requirements and architecture definition
syst req archit def
syst concept definition
system operations and maintenance
system operations and maintenance
system operations and maintenance
system operations and maintenance
system implementation
system implementation
system implementation
system implement
release 1
release 2
release 3
release 4
system integration and test
system installation and acceptance
system installation and acceptance
system integration and test
system installation and acceptance
system installation and acceptance
system integration and test
system integration and test

Requirements Analysis	Design Engineering	Construction	Test (system, acceptance)

Waterfall development model

Complete Detailed Frozen	Complete Detailed Frozen	Build/test	Build/test	Build/test	Build/test	Build/test	
Requirements Analysis & specification	Design Spec	Step 1 →	Step 2 →	Step 3 →	Step 4 →	Step n →	Acceptance Test

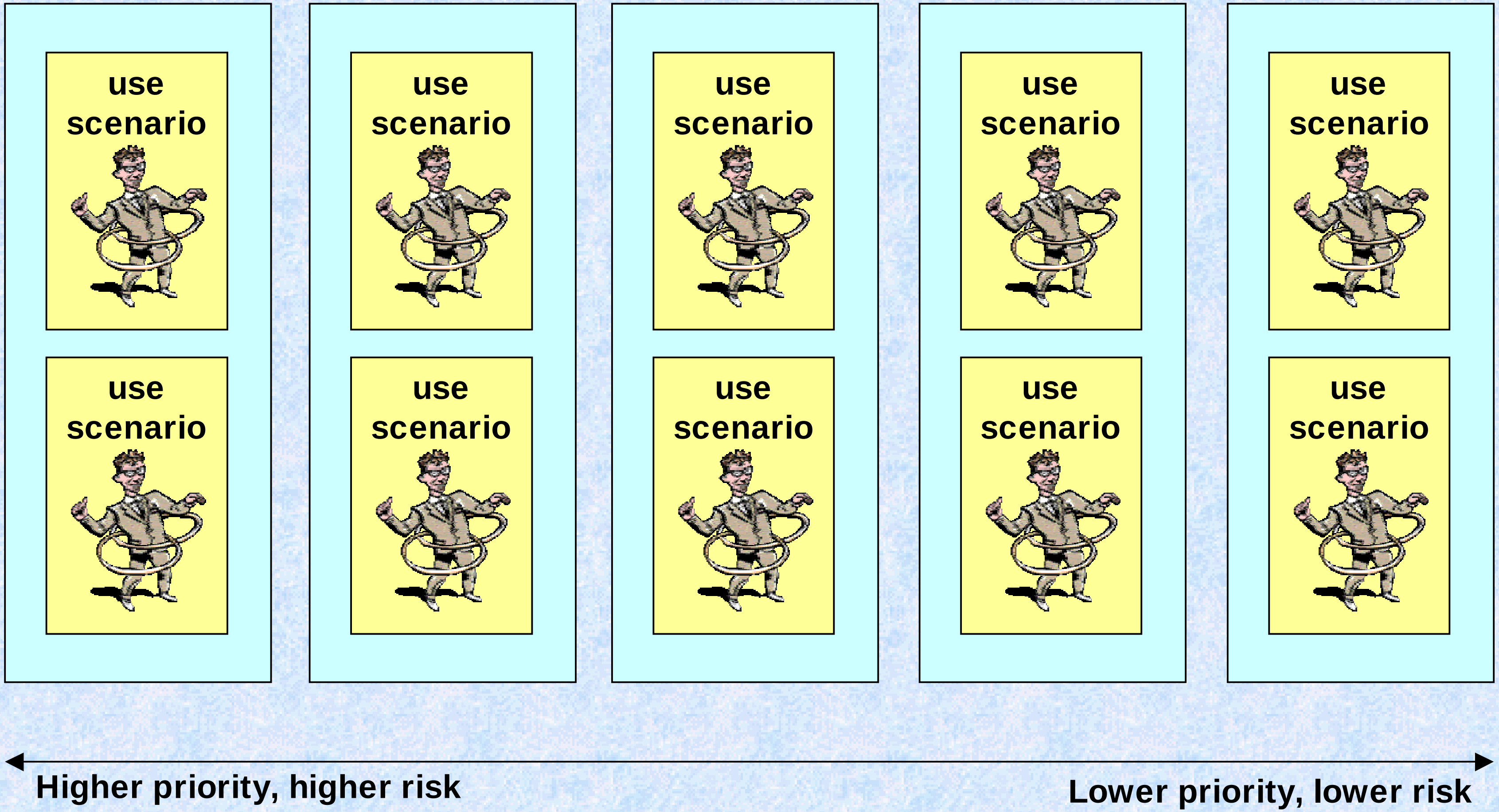
Incremental development model

Best guess Updated stepwise	Best Guess Updated stepwise	Requirements Design Build Test Use	Requirements Design Build Test Use	Requirements Design Build Test Use	Requirements Design Build Test Use	Requirements Design Build Test Use	
Requirements Analysis & specification (needs)	Design specs (ideas)	Step 1 →	Step 2 →	Step 3 →	Step 4 →	Step '50' →	Contract Acceptance Test

Evolutionary development model

Ref. Tom Gilb: Evo

Prioritize use scenarios



use scenario

use scenario

use scenario

use scenario

use scenario

use scenario

use scenario

use scenario

use scenario

use scenario

Higher priority, higher risk

Lower priority, lower risk

Sample two-week evo-cycle

Monday	Tuesday	Wednesday	Thursday	Friday
Final test of last week's build Review and enhance analysis models for new features	Release last week's build to users Create design models for new features Begin implementation of new features	Incremental build overnight		Weekend build from scratch

User use

Monday	Tuesday	Wednesday	Thursday	Friday
	All user feedback collected	Functionality freeze – no new features added beyond this point Incremental build overnight	Test new functionality Review feedback, determine changes for next release	Test new functionality Weekend build from scratch

Cleanroom principles

- Incremental development
  - User verifyable increments
- Team organisation
  - 4~8 people
- Formal methods of specification and design
  - Level of formalism varies even within project
- Intense review
  - Mathematical proof of correctness
  - Verifying individual control structures
- No unit test
  - No testing of infinite number of paths
- Statistical testing as reliability measurement
  - Testing is not suitable for bug-hunting

Box structure specification techniques

- **Black box**

  (current stimulus, stimulus history) → response

- **State box**

  (current stimulus, current state) → (response, new state)

- **Clear box**

  (current stimulus, current state) → (response, new state), by procedures

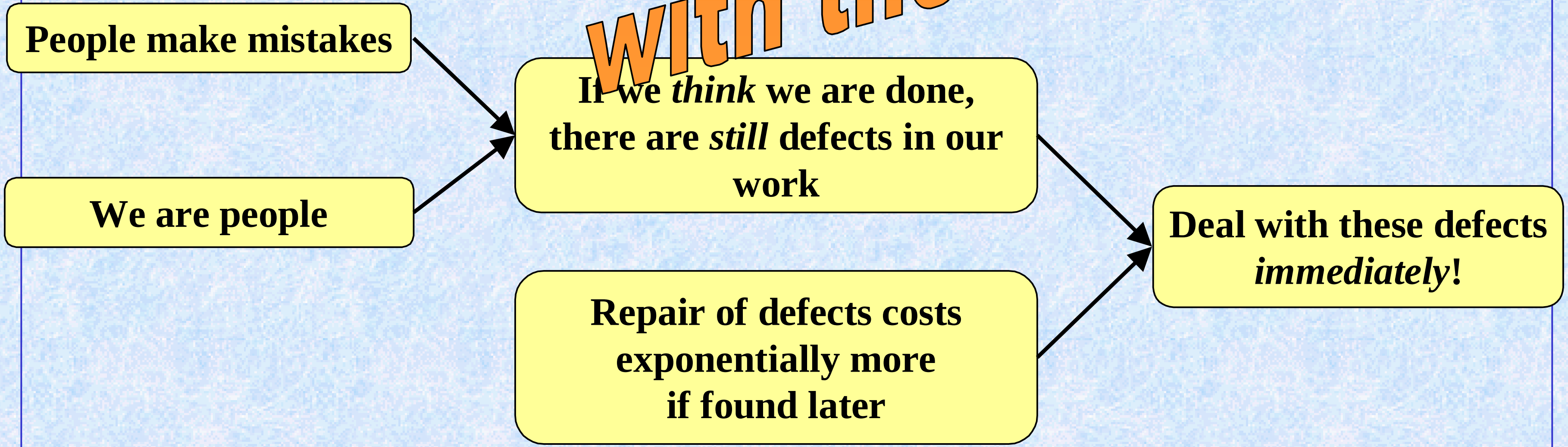
Generally, different parts of a software system require different specification techniques

Cleanroom principles

- **Incremental development**
  - **User verifyable increments**
- **Team organisation**
  - **4~8 people**
- **Formal methods of specification and design**
  - **Level of formalism varies even within project**
- **Intense review**
  - **Mathematical proof of correctness**
  - **Verifying individual control structures**
- **No unit test**
  - **No testing of infinite number of paths**
- **Statistical testing as reliability measurement**
  - **Testing is not suitable for bug-hunting**

Workprocess

So, when do we deal with these defects?



People make mistakes

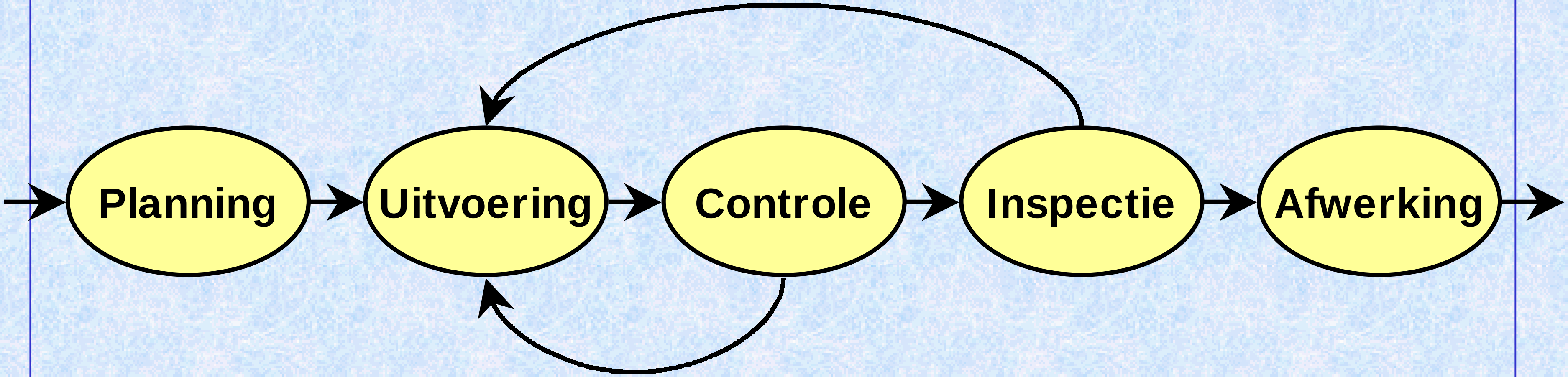
We are people

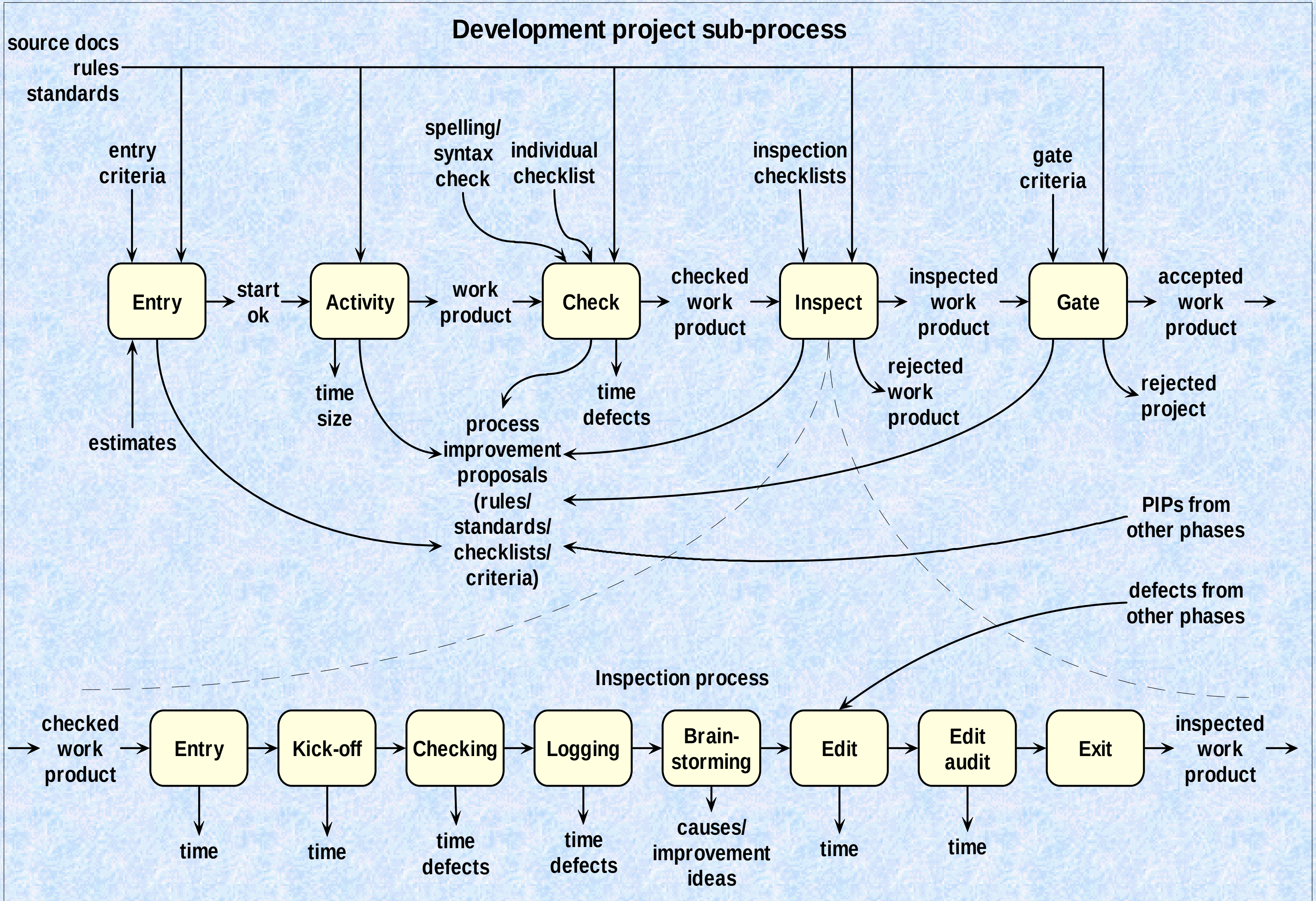
If we *think* we are done, there are *still* defects in our work

Repair of defects costs exponentially more if found later

Deal with these defects *immediately*!

Sub-fasen in elk projectonderdeel: PUCIA werkproces

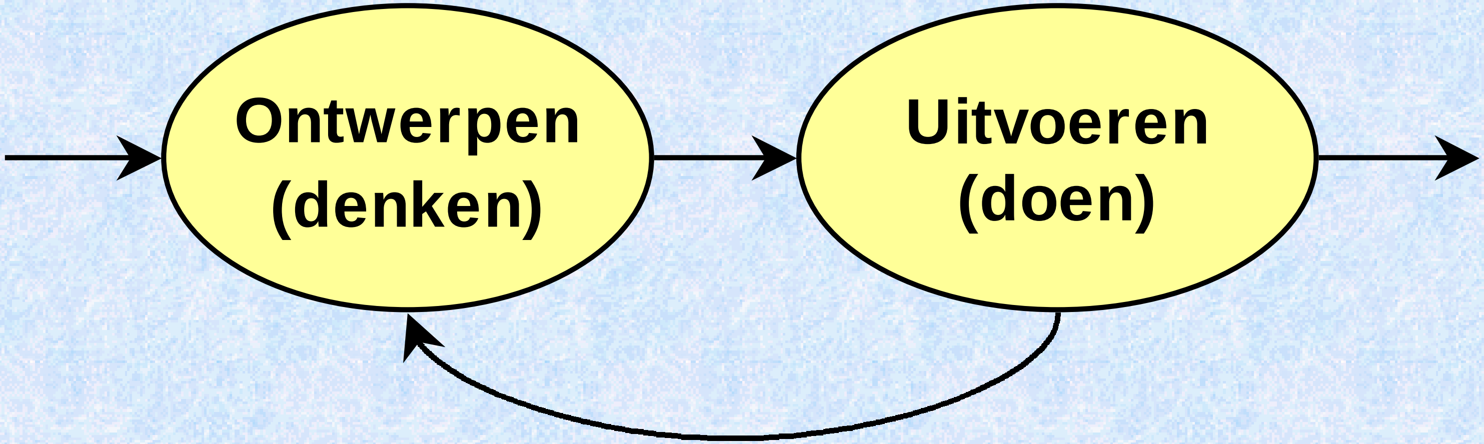




© 2000 N R Malotaux - Consultancy

file: http://www.malotaux.nl/nrm/pdf/subprocess.pdf

Bij vastlopen in de uitvoering:
terug naar ontwerp!



Cleanroom principles

- Incremental development
  - User verifyable increments
- Team organisation
  - 4~8 people
- Formal methods of specification and design
  - Level of formalism varies even within project
- Intense review
  - Mathematical proof of correctness
  - Verifying individual control structures
- No unit test
  - No testing of infinite number of paths
- Statistical testing as reliability measurement
  - Testing is not suitable for bug-hunting

Wanneer compileren?

- Als je hoopt dat het werkt
- Als je weet dat het werkt
- Als je zeker weet dat het werkt

Debuggen???

Defecten

- **Fouten ontstaan niet vanzelf**
- **Een ontwerp heeft geen *bugs* maar *defecten***
- **Ontwerpers maken fouten en veroorzaken daarmee defecten**
- **Wijzigen in het PvE veroorzaakt defecten**

Testen van software

- 50% van defecten wordt in test niet gevonden
- Reparatie van defecten veroorzaakt defecten
- Een compiler vindt 10% syntaxfouten niet
- Van 4 defecten worden 2 gevonden bij compileren, 1 bij testen en 1 bij de klant ...
- Testen om bugs te vinden kost veel te veel tijd

Testen is dus uitermate inadequaat voor het detecteren van fouten!

Niet testen dan?

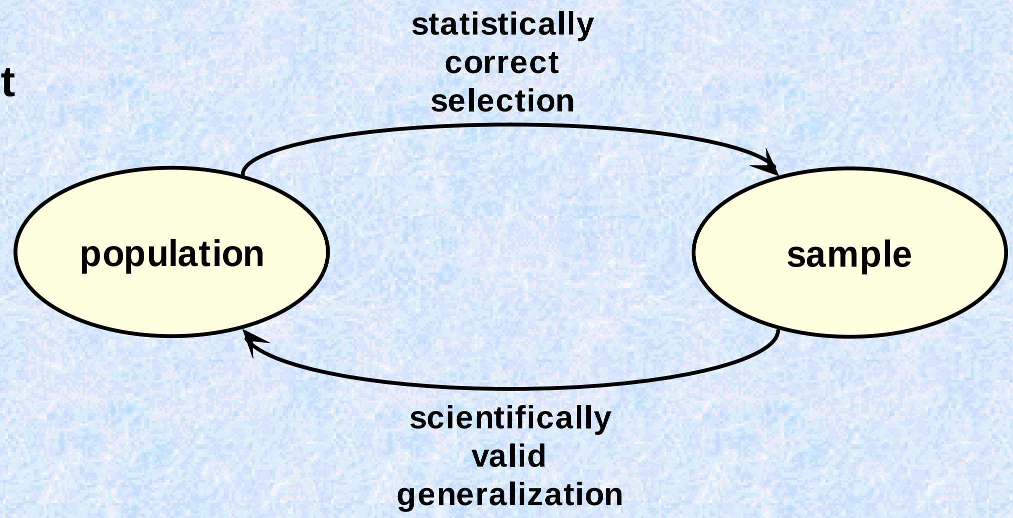
- Wel testen
- Doel moet niet foutdetectie zijn maar:

Constateren dat het werkt

Cleanroom principles

- Incremental development
  - User verifyable increments
- Team organisation
  - 4~8 people
- Formal methods of specification and design
  - Level of formalism varies even within project
- Intense review
  - Mathematical proof of correctness
  - Verifying individual control structures
- No unit test
  - No testing of infinite number of paths
- Statistical testing as reliability measurement
  - Testing is not suitable for bug-hunting

Statistical experiment



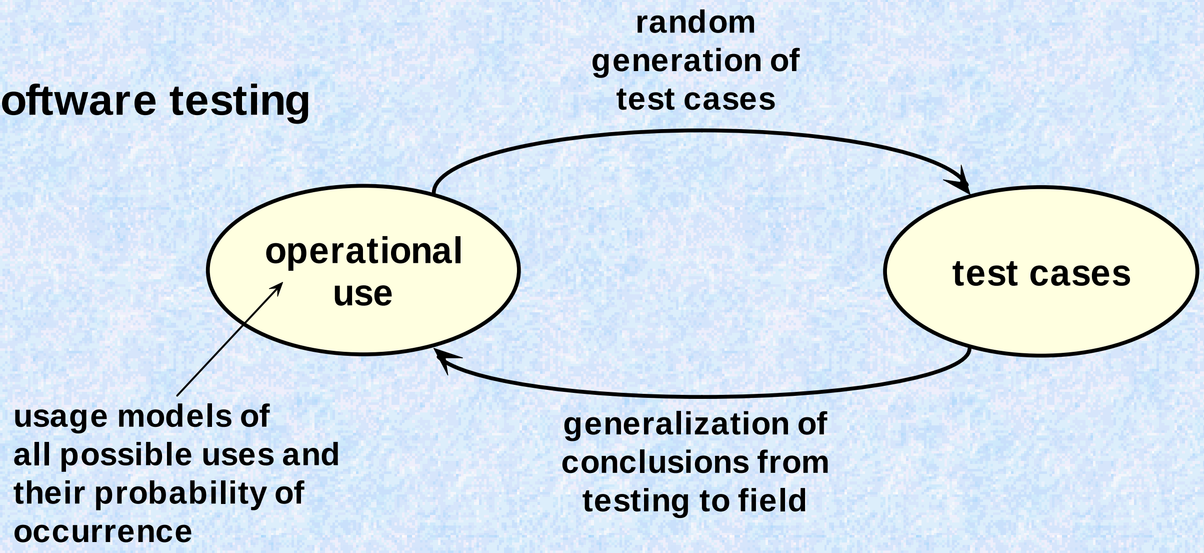
statistically correct selection

population

sample

scientifically valid generalization

Statistical software testing



random generation of test cases

operational use

test cases

usage models of all possible uses and their probability of occurrence

generalization of conclusions from testing to field

Cleanroom processes (SEI)

- Management
- Specification
- Development
- Testing and certification

Cleanroom management processes

- **Project planning process** CMM-2
  - Cleanroom engineering guide
  - Software development plan
- **Project management process** CMM-2
  - Project record
- **Performance improvement process** CMM-5
  - Performance improvement plans
- **Engineering change process** CMM-2
  - Engineering change log

Cleanroom plans

Software development plan

1. Project mission plan
2. Project organisation plan
3. Work product plan
4. Schedule and resource plan
5. Measurement plan
6. Reuse analysis plan
7. Risk analysis plan
8. Standards plan
9. Training plan
10. Configuration management plan

Cleanroom specification processes

- Requirements analysis process CMM-2
  - Software requirements
- Function specification process CMM-3
  - Function specification (black box, state box, clear box)
- Usage specification process CMM-2
  - Usage specification
- Architecture specification process CMM-3
  - Software architecture
- Increment planning process CMM-2
  - Increment construction plan

Cleanroom development processes

- Software reengineering process CMM-3
  - Reengineering plan
  - Reengineered software
- Increment design process CMM-2
  - Increment design
- Correctness verification process CMM-3
  - Increment verification reports
- Architecture specification process CMM-3
  - Software architecture

Cleanroom certification processes

- Usage modelling and test planning process
  - Usage models (abuse models)
  - Increment test plan
  - Statistical test cases
- Statistical testing and certification process
  - Executable system
  - Statistical testing reports
  - Increment certification reports

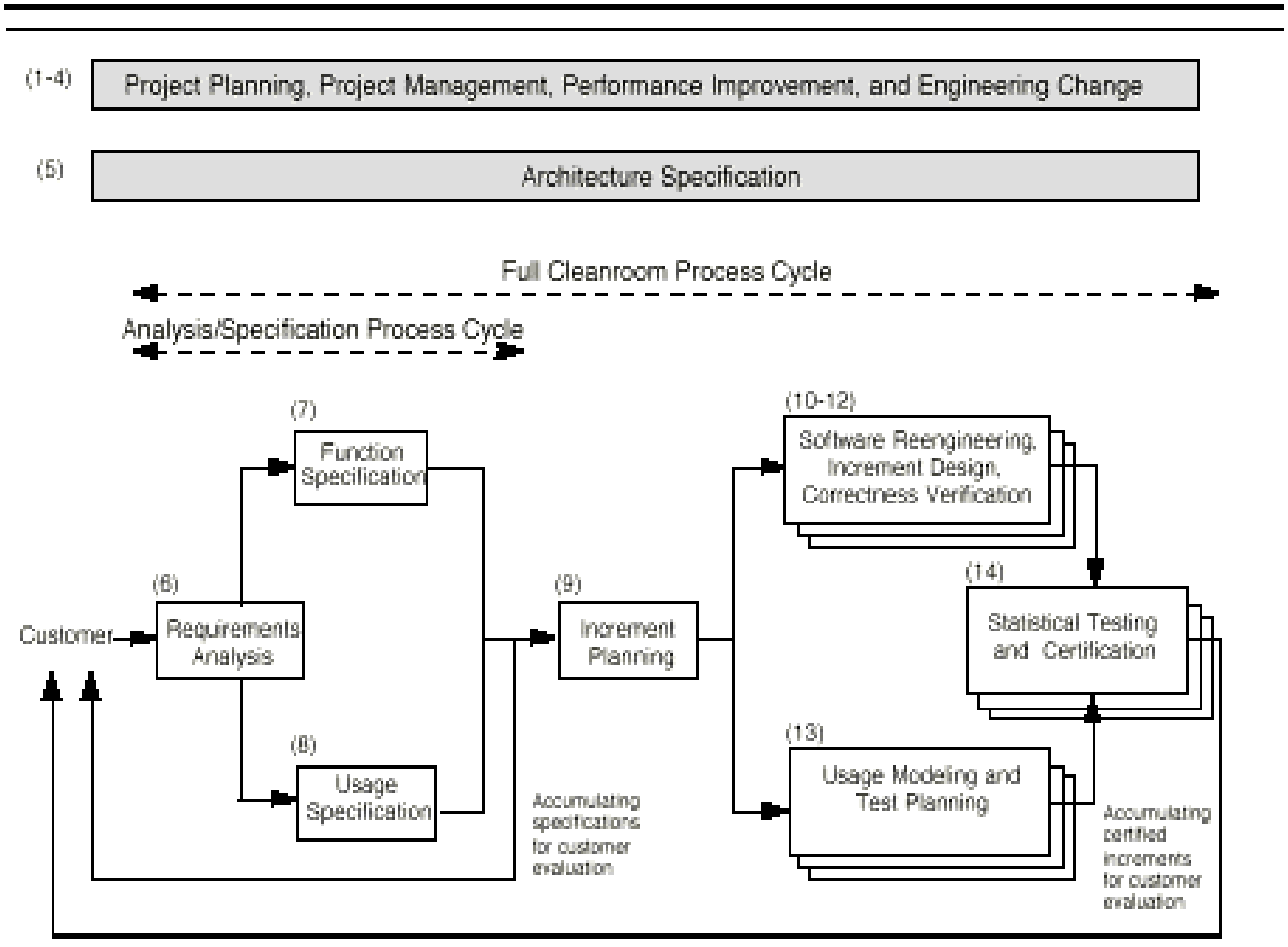

(1-4)
Project Planning, Project Management, Performance Improvement, and Engineering Change
(5)
Architecture Specification
Full Cleanroom Process Cycle
Analysis/Specification Process Cycle
(7)
Function Specification
(6)
Requirements Analysis
Customer
(8)
Usage Specification
(9)
Increment Planning
Accumulating specifications for customer evaluation
(10-12)
Software Reengineering, Increment Design, Correctness Verification
(14)
Statistical Testing and Certification
(13)
Usage Modeling and Test Planning
Accumulating certified increments for customer evaluation

Figure 1. Cleanroom Process Flow

Cleanroom fundamentals

- **Design principle**
  - **Designers *can* and *should* produce systems free of defects before testing**
- **Testing principle**
  - **The purpose of testing is to measure quality**
- **Main development model**
  - **Incremental (Cleanroom)/evolutionary (Gilb)/cyclic (TSP)**
    - **Each increment is a workingsubset of the final product**
    - **Stable requirements for each increment (req. paradox!)**
    - **No eleventh hour integration**

Philosophy behind Cleanroom

- **To avoid dependence on costly defect-removal processes**
- **By writing code increments right the first time and**
- **Verifying their correctness before testing.**

(Linger, 1994)

What should I do?

1. **Start designing in stead of hacking**
2. **Use Inspections (Gilb)**
3. **Evolutionary development model (Gilb)**
4. **Prevent defects**
5. **Testing is not bug finding**

References

- Look at

  http://www.malotaux.nl/nrm

- Download page for slides
- Books page for literature
  - SP8: Dyer: Cleanroom approach to software development
  - SP11: SEI: Cleanroom software engineering reference model
  - SP12: SEI: Mapping of CMM and Cleanroom
  - SP17: HP: Evolutionary Fusion
  - SP18: DoD: Cleanroom engineering tutorial

Cleanroom software engineering

Ir. Niels Malotaux

030-228 88 68
niels@malotaux.nl
www.malotaux.nl/nrm

Cost of slipping defects

			Only compile and test			PSP + own review			Add group review		
	rel cost	defect detect	slipped	not detect	cost	slipped	not detect	cost	slipped	not detect	cost
PSP	1	50%		-			50,0%	0,5		50,0%	0,5
Review	1	70%	-	-	-	50,0%	15,0%	0,4	50,0%	15,0%	0,4
Inspection	4	70%	-	-	-	-	-	-	15,0%	4,5%	0,4
Compile	1	50%	100,0%	50,0%	0,5	15,0%	7,5%	0,1	4,5%	2,3%	0,02
Test	30	50%	50,0%	25,0%	7,5	7,5%	3,8%	1,1	2,3%	1,1%	0,3
Use	100	100%	25,0%	0,0%	25	3,8%	0,0%	3,8	1,1%	0,0%	1,1
	Cost of slipping defects				33			6			3
				normalized	12			2			1
	Cost pre-customer				8			2,1			2
				normalized	5			1,3			1
	Cost at customer				25			4			1
				normalized	22			3			1