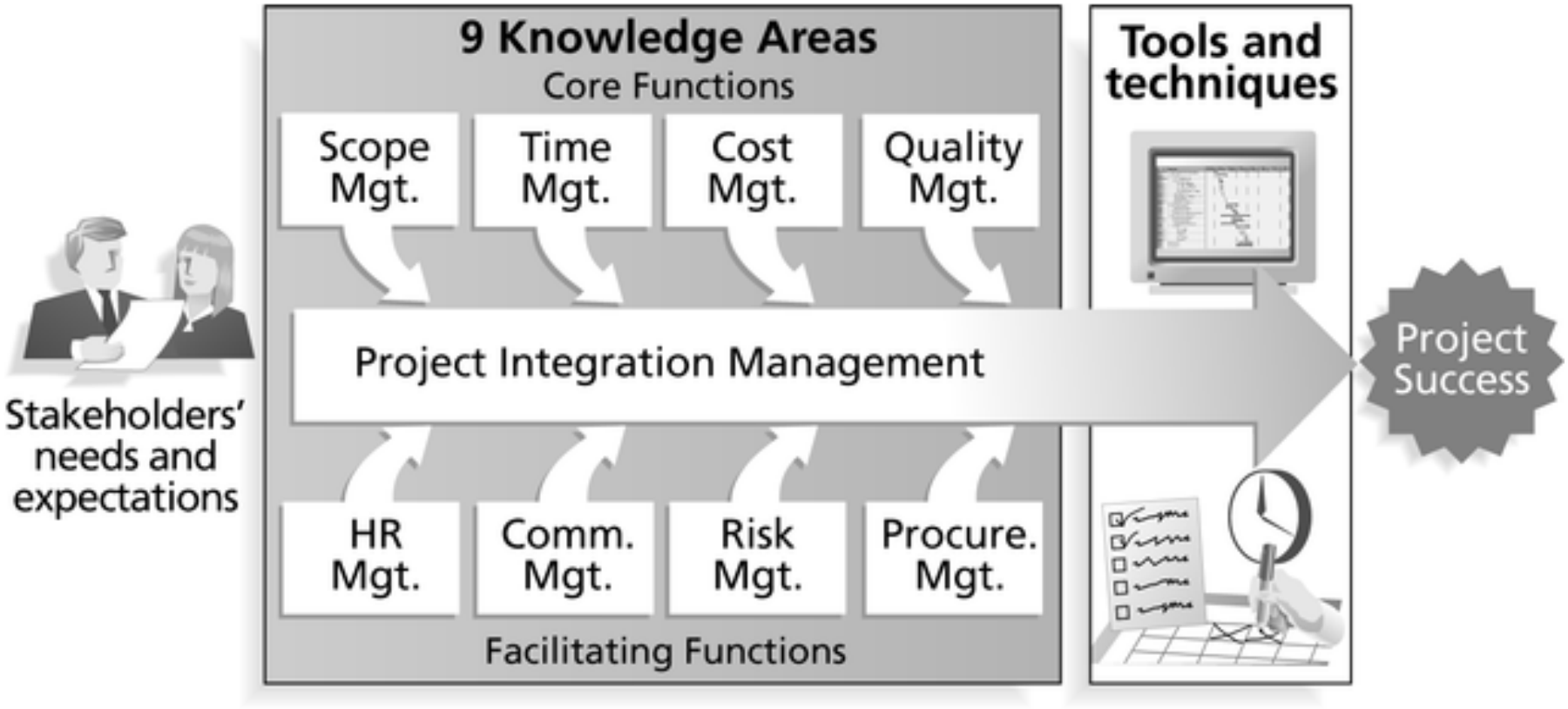
ΚΥΚΛΟΣ ΖΩΗΣ ΛΟΓΙΣΜΙΚΟΥ

Δρ. Πάνος Φιτσιλής

PMBOK

- Οργανώνει τη διαχείριση έργων σε
  - Α) Διεργασίες (Processes)
  - Β) Περιοχές Γνώσεις (Knowledge Areas)
- Διεργασίες 2 ειδών
  - 1. Διεργασίες PM : περιγράφουν και οργανώνουν το έργο
  - 2. Διεργασίες σχετικές με το αντικείμενο του έργου (Product-oriented processes)

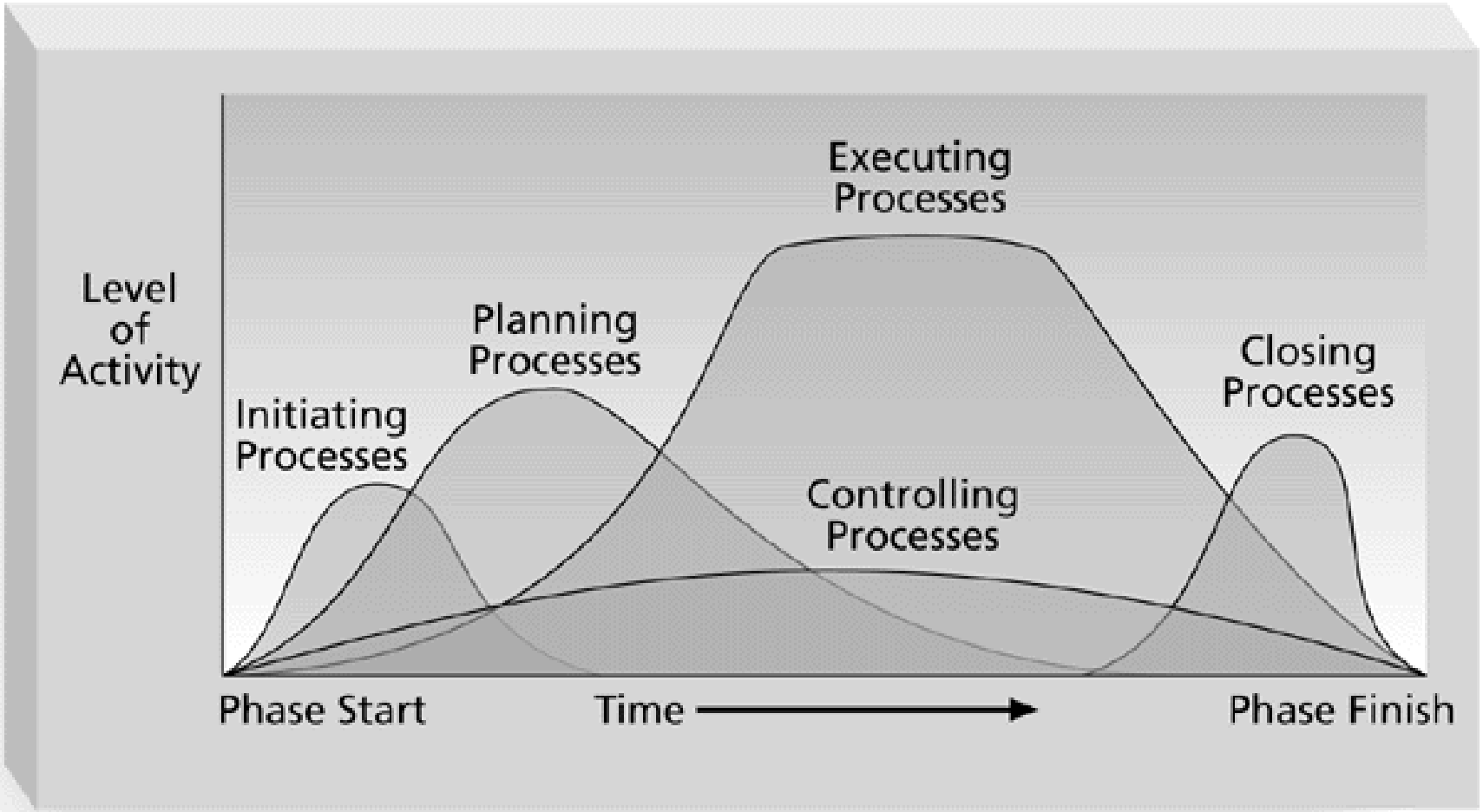
ΤΟ ΠΛΑΊΣΙΟ PMI



ΟΙ ΟΜΆΔΕΣ ΔΙΕΡΓΑΣΙΏΝ PMI (PROCESS GROUPS)

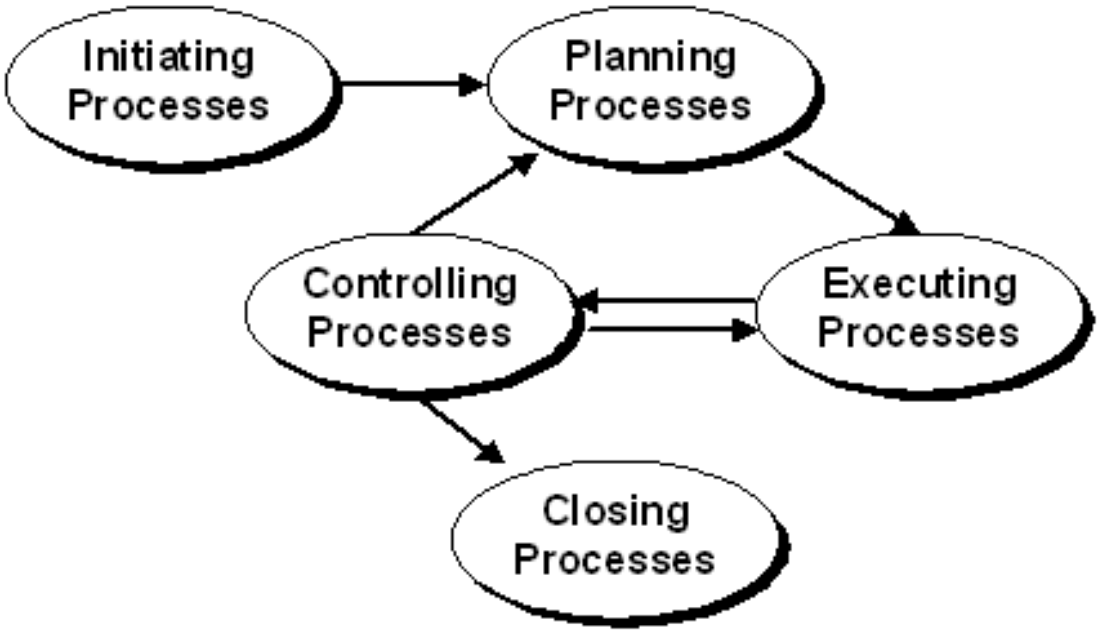
- 1. Initiating
- 2. Planning
- 3. Executing
- 4. Controlling
- 5. Closing
- Η κάθε διεργασία περιγράφεται από :
  - Inputs
  - Tools & Techniques
  - Outputs

ΟΙ ΟΜΆΔΕΣ ΔΙΕΡΓΑΣΙΩΝ PMI

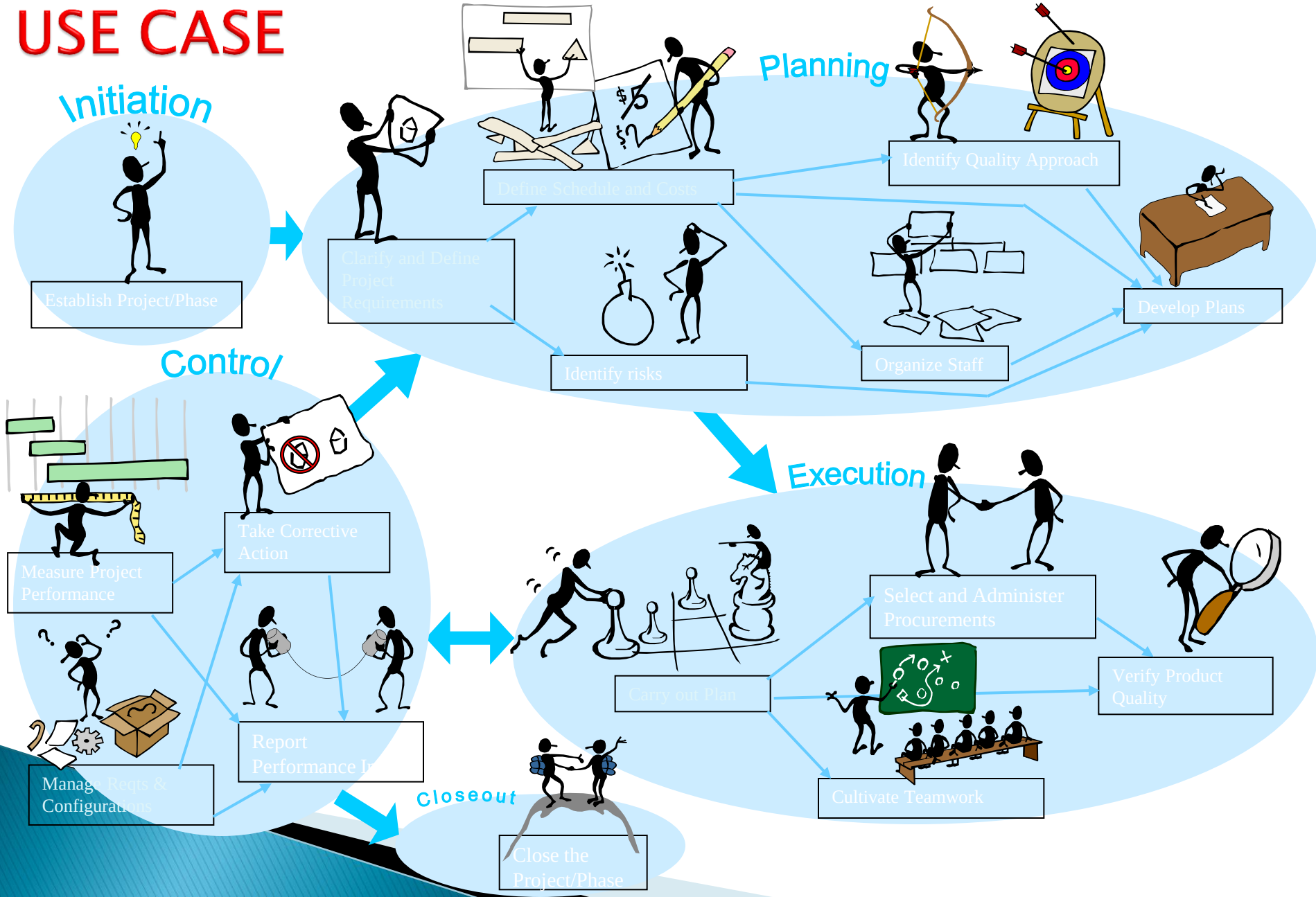


Institute

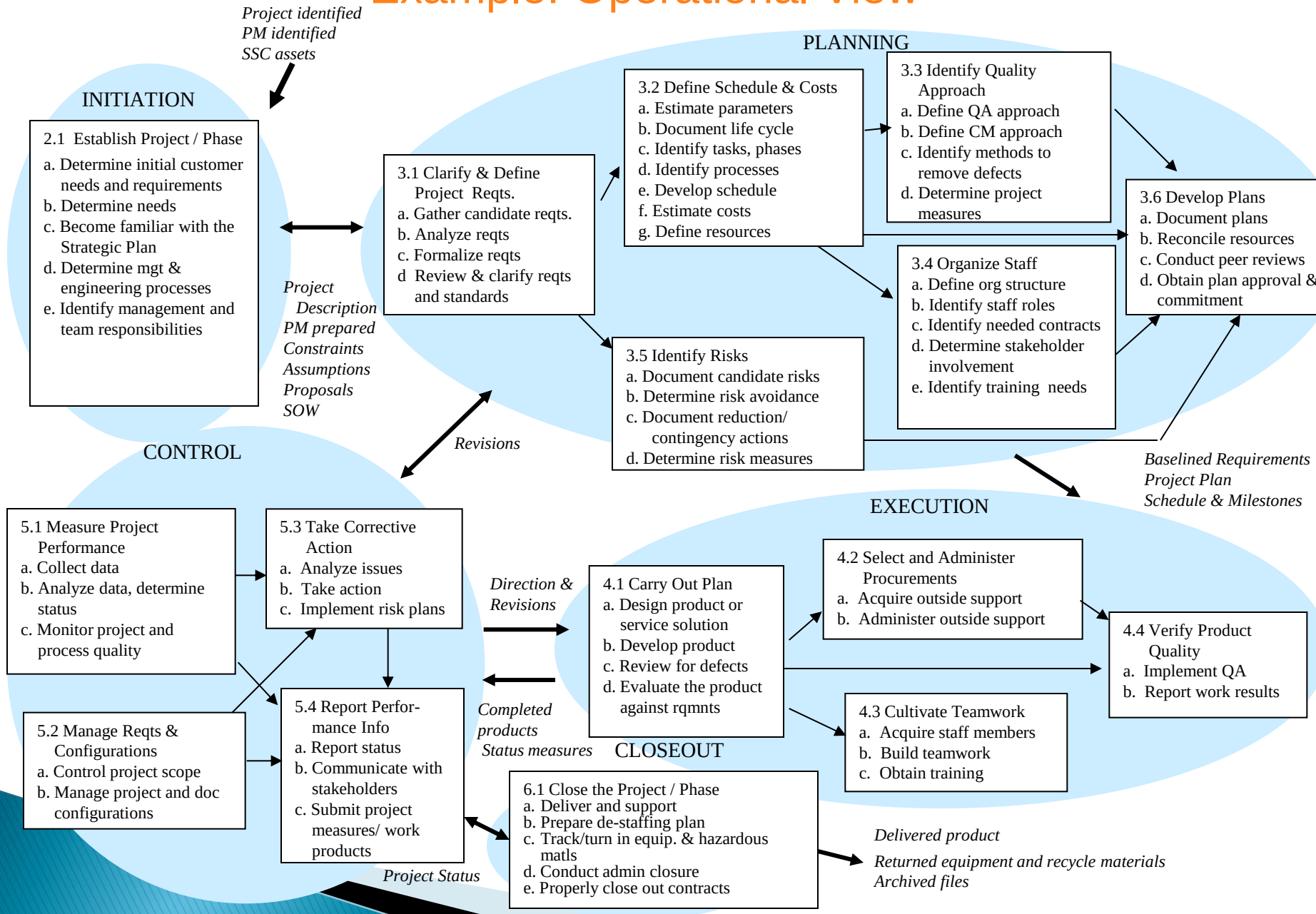
ΟΙ ΟΜΑΔΕΣ ΔΙΕΡΓΑΣΙΩΝ PMI



EXAMPLE: PROJECT MANAGEMENT USE CASE

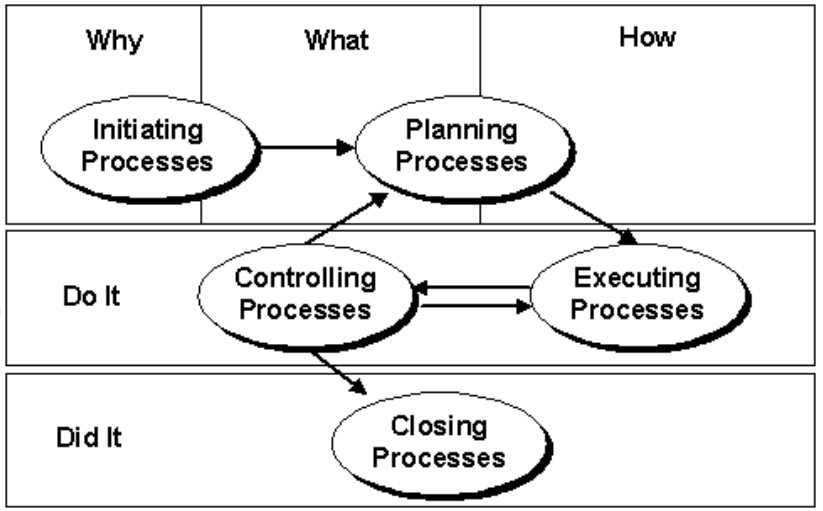


Example: Operational View



ΟΙ ΟΜΑΔΕΣ ΔΙΕΡΓΑΣΙΩΝ PMI

- Γιατί – Why
  - Deliverable: ROI
- Τι – What
  - SOW, Requirements
- Πως – How
  - Design Specification, SDP,
- Κάντο – Do
  - Execution
- Έγινε Done
  - PPR



Futrell, Shafer, Shafer, "Quality Software Project Management"

PMI: INITIATING PROCESS

- Inputs
  - Product Description
  - Strategic plan
  - Project Selection Criteria
  - Historical Information
- Outputs
  - Project charter
  - Project Manager assigned
  - Constraints
  - Assumptions

Αποφασίζουμε για το έργο που θα κάνουμε

PMI: PLANNING PROCESS

- Scope Planning
- Scope Definition
- Activity Definition
- Activity Sequencing
- Activity Duration Estimating
- Resource Planning
- Cost Estimating
- Cost Budgeting
- Risk Planning
- Schedule Development
- Quality Planning
- Communications Planning
- Organization Planning
- Staff Acquisition
- Procurement Planning
- Project Plan Development

PMI: EXECUTING PROCESS

- Project Plan Execution
- Scope Verification
- Quality Assurance
- Team Development
- Information Distribution
- Solicitation
- Source Selection
- Contract Administration

PMI: CONTROLLING PROCESS

- Overall Change Control
- Scope Change Control
- Schedule Control
- Cost Control
- Quality Control
- Performance Reporting
- Risk Response Control

PMI: CLOSING PROCESS

- Administrative Closure
- Contract Close-out

PMI ΠΕΡΙΟΧΕΣ ΓΝΩΣΗΣ/ΟΜΑΔΕΣ ΔΙΕΡΓΑΣΙΩΝ

Process Groups / Knowledge Area	Initiating	Planning	Executing	Controlling	Closing
4. Project Integration Management		4.1 Project Plan Development	4.2 Project Plan Execution	4.3 Integrated Change Control	
5. Project Scope Management	5.1 Initiation	5.2 Scope Planning 5.3 Scope Definition		5.4 Scope Verification 5.5 Scope Change Control	
6. Project Time Management		6.1 Activity Definition 6.2 Activity Sequencing 6.3 Activity Duration Estimating 6.4 Schedule Development		6.5 Schedule Control	
7. Project Cost Management		7.1 Resource Planning 7.2 Cost Estimating 7.3 Cost Budgeting		7.4 Cost Control	
8. Project Quality Management		8.1 Quality Planning	8.2 Quality Assurance	8.3 Quality Control	
9. Project Human Resource Management		9.1 Organizational Planning 9.2 Staff Acquisition	9.3 Team Development		
10. Project Communications Management		10.1 Communications Planning	10.2 Information Distribution	10.3 Performance Reporting	10.4 Administrative Closure
11. Risk Project Management		11.1 Risk Management Planning 11.2 Risk Identification 11.3 Qualitative Risk Analysis 11.4 Quantitative Risk Analysis 11.5 Risk Response Planning		11.6 Risk Monitoring and Control	
12. Project Procurement Management		12.1 Procurement Planning 12.2 Solicitation Planning	12.3 Solicitation 12.4 Source Selection 12.5 Contract Administration		12.6 Contract Closeout

Ο ΚΥΚΛΟΣ ΖΩΗΣ ΣΤΑ ΕΡΓΑ

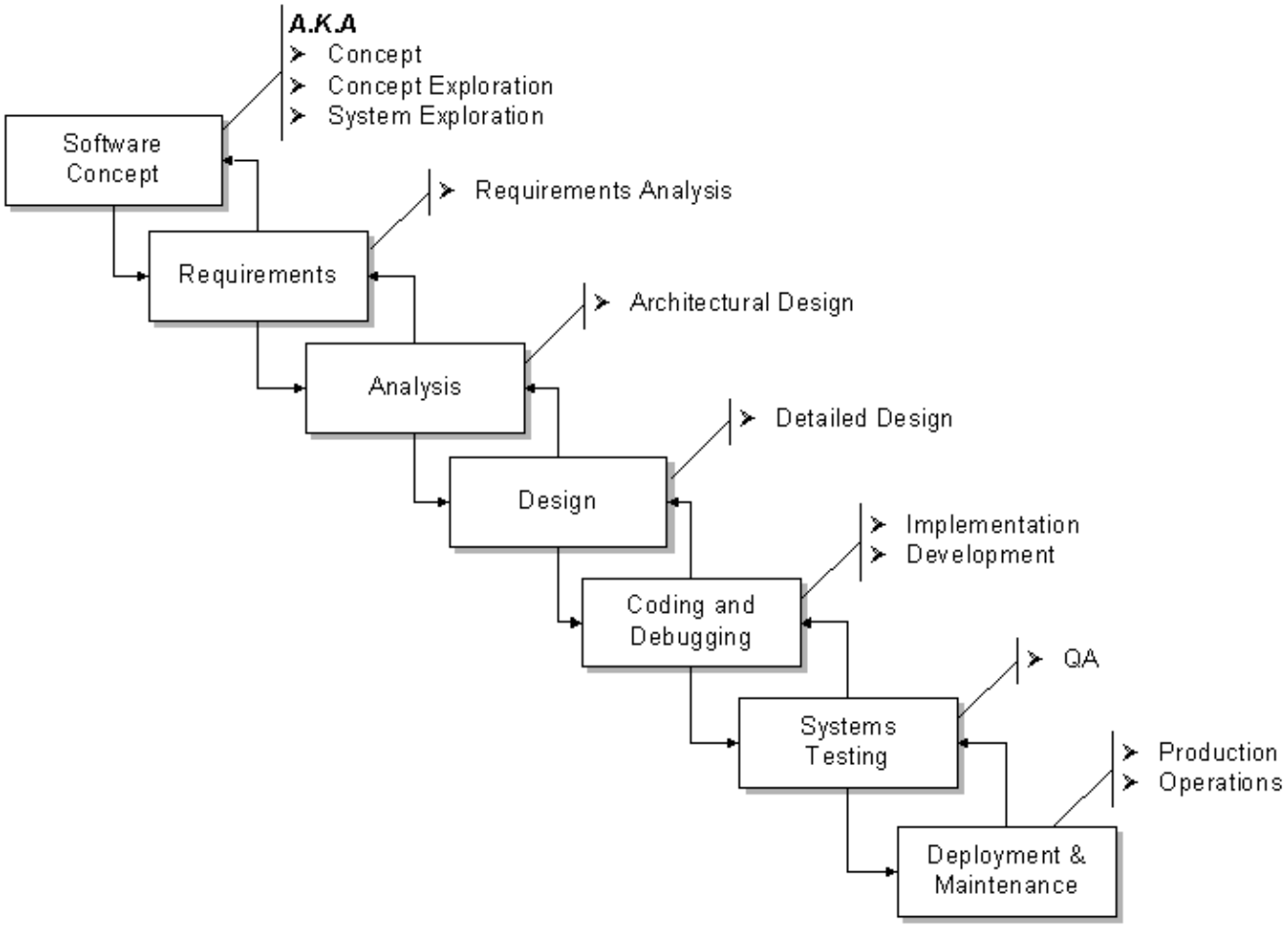
PROJECT LIFE CYCLE

- Οι φάσεις του κάθε έργου προσδιορίζονται από τα παραδοτέα (deliverables)
- Ένα παραδοτέο είναι ένα χειροπιαστό, επαληθεύσιμο ποσό εργασίας:
  - Μελέτη σκοπιμότητας
  - Σχεδιασμός έργου
- Το τέλος κάθε φάσης σηματοδοτεί
  - Έλεγχο παραδοτέων
  - Έλεγχο απόδοσης
    - Συνέχεια σε επόμενη φάση
    - Διόρθωση λαθών

ΠΑΡΑΔΕΙΓΜΑΤΑ

- waterfall model,
- Spiral model,
- V-Model,
- incremental/iterative,
- Unified process,
- Agile,
- etc.

ΒΑΣΙΚΑ ΒΗΜΑΤΑ ΚΥΚΛΟΥ ΖΩΗΣ



ΧΡΟΝΟΣ ΑΝΑ ΦΑΣΗ

- Ο κανόνας 40-20-40
  - Προδιαγραφές – Υλοποίηση – Τέστ

	Planninig	Code & Unit Test	Integration & Test
Εμπορικά Συστήματα	25%	40%	35%
Web Συστήματα	55%	15%	30%
Real-time συστήματα	35%	25%	40%
Αμυντικά Συστήματα	40%	20%	40%

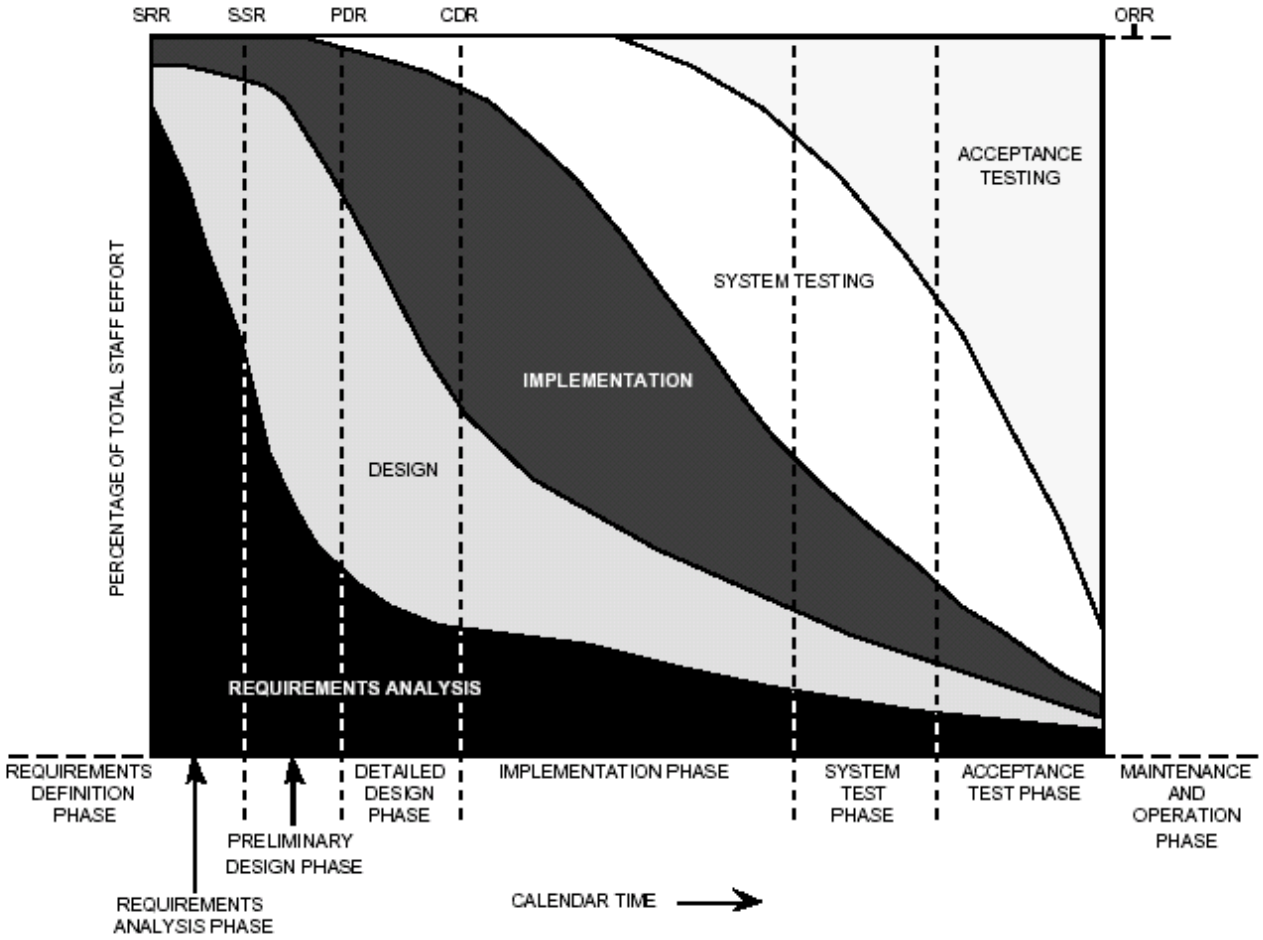
Bennatan, E.M, “On Time Within Budget”

ΧΡΟΝΟΣ ΑΝΑ ΦΑΣΗ

Activity	Small Project (2.5K LOC)	Large Project (500K LOC)
Analysis	10%	30%
Design	20%	20%
Code	25%	10%
Unit Test	20%	5%
Integration	15%	20%
System test	10%	15%

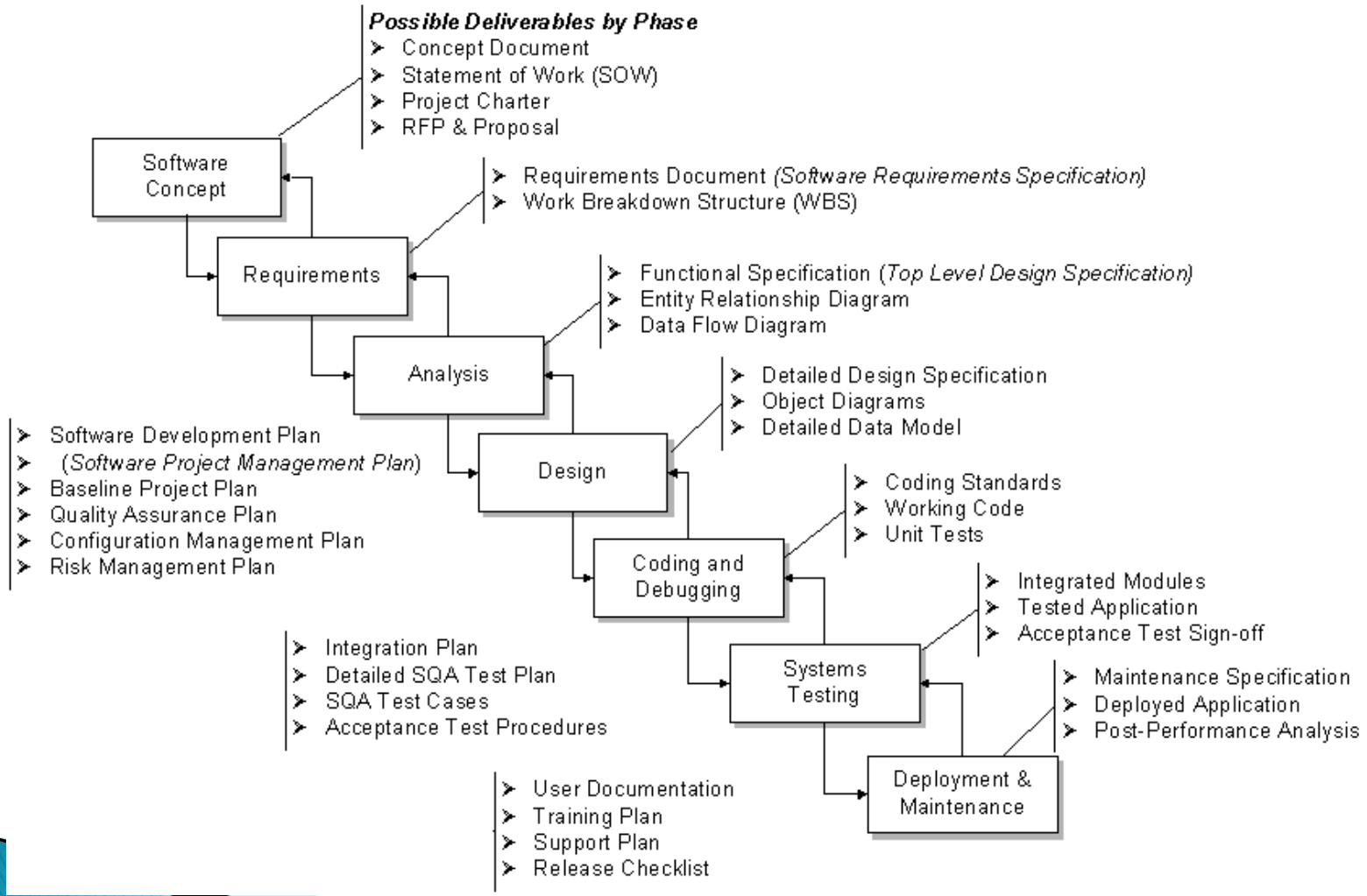
McConnell, Steve, “Rapid Development”

% ΣΥΝΟΛΙΚΗΣ ΠΡΟΣΠΑΘΕΙΑΣ

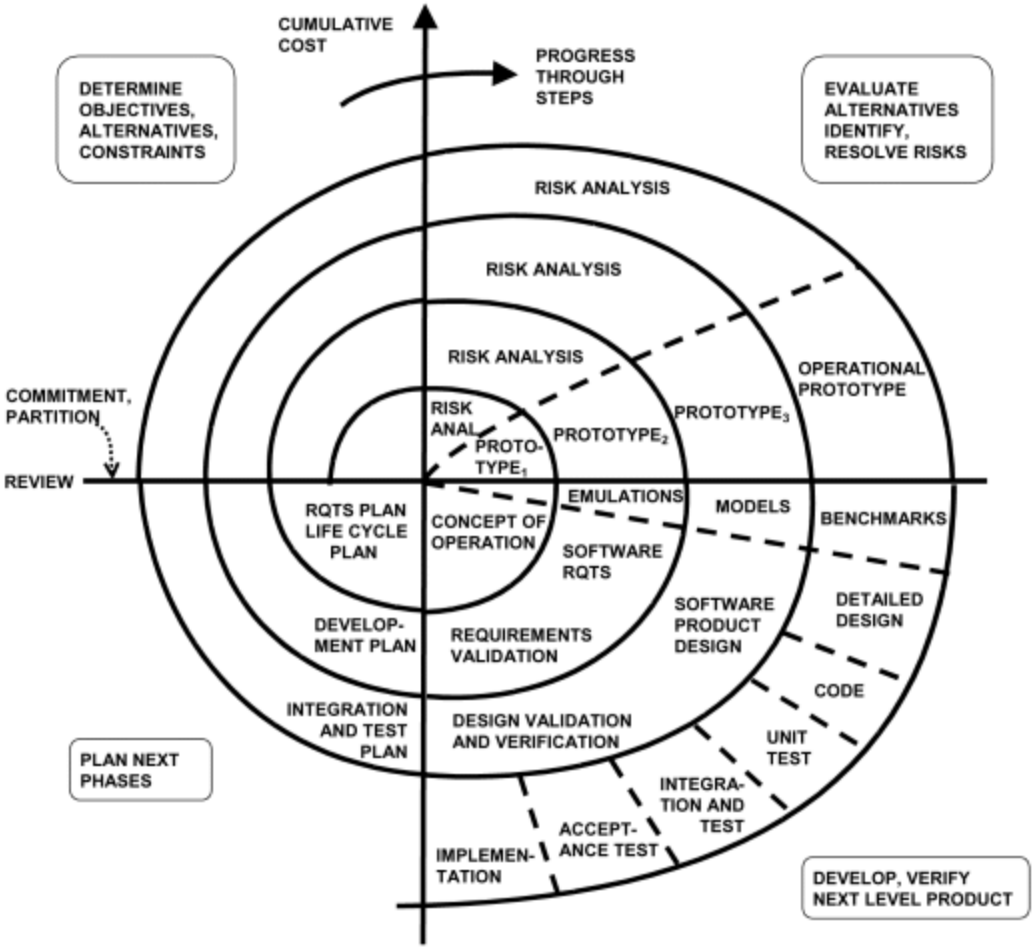


NASA's "Manager's Handbook for Software Development"

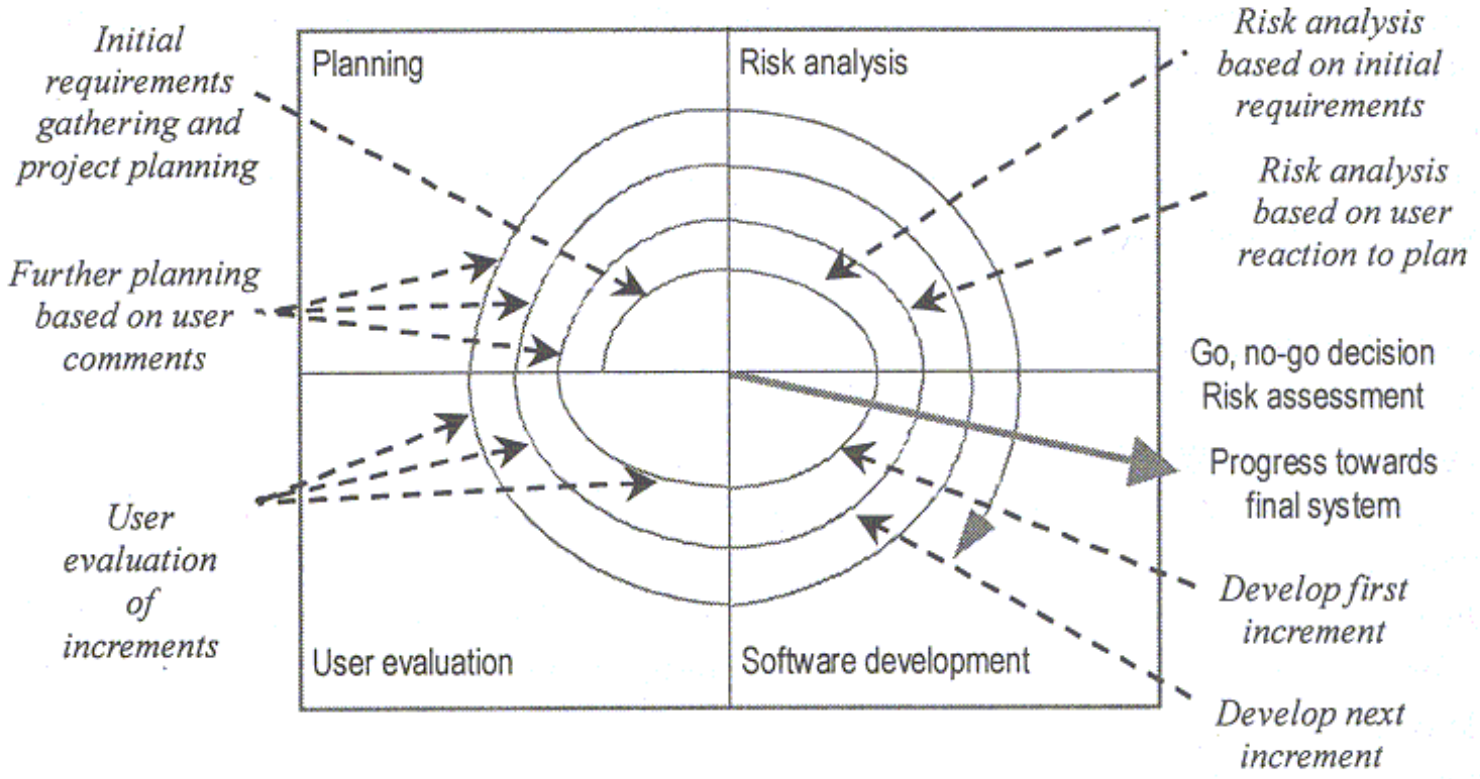
ΠΑΡΑΔΟΤΕΑ ΑΝΑ ΦΑΣΗ



SPIRAL


CUMULATIVE COST
PROGRESS THROUGH STEPS
DETERMINE OBJECTIVES, ALTERNATIVES, CONSTRAINTS
EVALUATE ALTERNATIVES IDENTIFY, RESOLVE RISKS
RISK ANALYSIS
RISK ANALYSIS
RISK ANALYSIS
OPERATIONAL PROTOTYPE
COMMITMENT, PARTITION
REVIEW
RISK ANAL
PROTO-TYPE1
PROTOTYPE2
PROTOTYPE3
RQTS PLAN LIFE CYCLE PLAN
CONCEPT OF OPERATION
EMULATIONS
MODELS
BENCHMARKS
SOFTWARE RQTS
DEVELOP-MENT PLAN
REQUIREMENTS VALIDATION
SOFTWARE PRODUCT DESIGN
DETAILED DESIGN
CODE
INTEGRATION AND TEST PLAN
DESIGN VALIDATION AND VERIFICATION
UNIT TEST
INTEGRA-TION AND TEST
ACCEPT-ANCE TEST
IMPLEMEN-TATION
PLAN NEXT PHASES
DEVELOP, VERIFY NEXT LEVEL PRODUCT

SPIRAL


Initial requirements gathering and project planning
Further planning based on user comments
User evaluation of increments
Planning
Risk analysis
User evaluation
Software development
Risk analysis based on initial requirements
Risk analysis based on user reaction to plan
Go, no-go decision Risk assessment
Progress towards final system
Develop first increment
Develop next increment

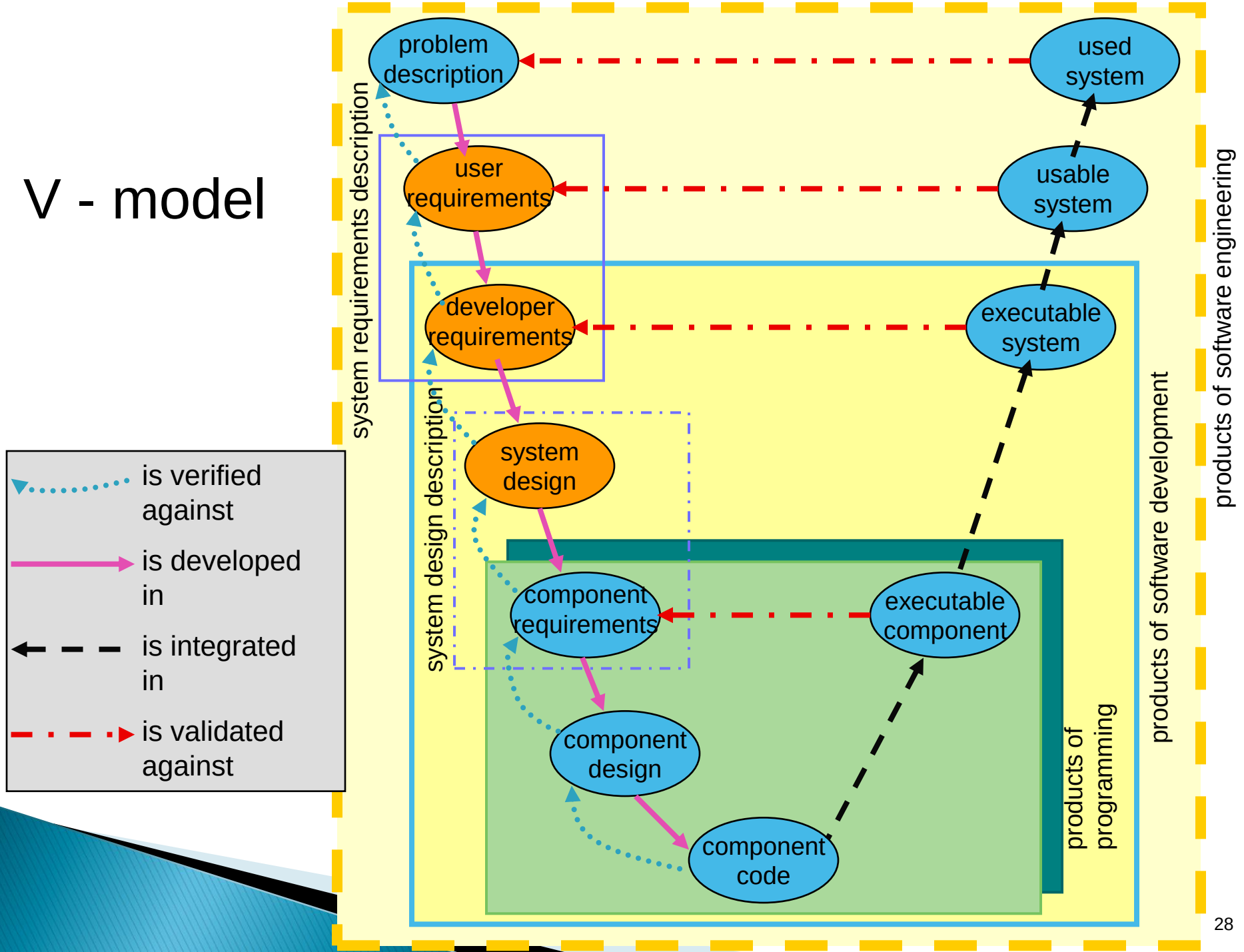
SPIRAL

- Δίνει έμφαση στη διαχείριση κινδύνου
- Είναι μια σειρά από μικρά έργα
- Ο αριθμός των επαναλήψεων είναι μεταβλητός
- Χρησιμοποιεί πρωτότυπα
- Οι πρώτοι κύκλοι είναι οι πιο οικονομικοί.

SPIRAL

- Πλεονεκτήματα
  - Μπορεί να συνδυαστεί με άλλα μοντέλα
  - Χειρίζεται το κίνδυνο από την αρχή.
- Μειονεκτήματα
  - Πολύπλοκος
  - Απαιτεί αρκετή διαχείριση

V - model


problem description
used system
user requirements
usable system
developer requirements
executable system
system design
component requirements
executable component
component design
component code
system requirements description
system design description
products of programming
products of software development
products of software engineering
is verified against
is developed in
is integrated in
is validated against

VERIFICATION VS VALIDATION

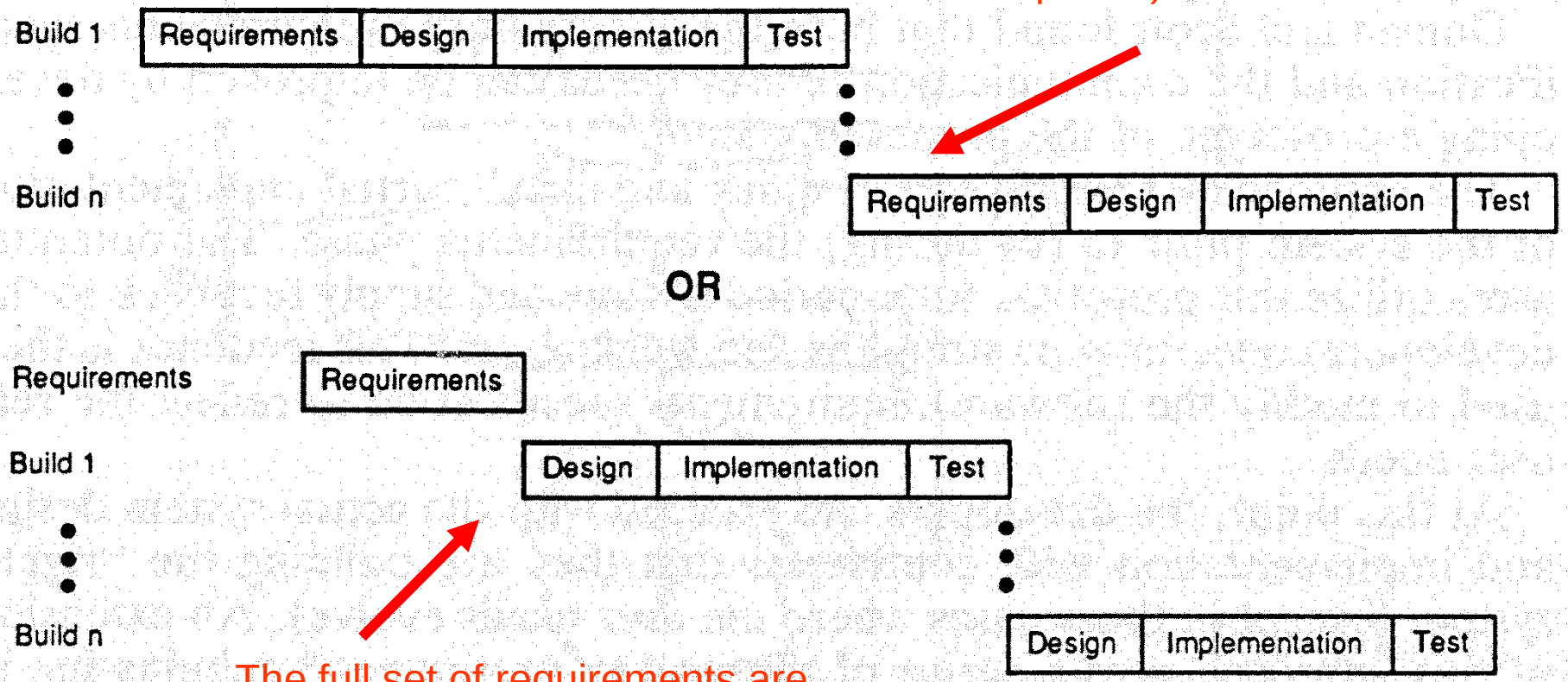
- **Verification** is the process of comparining a work product with its parent specification or a standard for the purpose of detecting errors.
  - Design is verified by comparing it with the requirements and code is verified by comparing it with the design.
  - Verification answers the question, "Did we build the product correctly?“
- **Validation** is the process of comparing a product to its high level requirements.
  - Using a validation method to determine if a system does what the requirements said it should is validation. The same methods used for verification can also be used for validation. Validation answers the question, "Did we build the correct product?"

V MODEL

- Δίνει έμφαση στον έλεγχο
  - Δίνει έμφαση σε Verification & Validation
- Είναι παραλλαγή του waterfall
- Πλεονεκτήματα
  - Ενθαρρύνει V&V σε όλες τις φάσεις
- Μειονεκτήματα
  - Δεν έχει επαναλήψεις
  - Δυσκολία στη διαχείριση αλλαγών

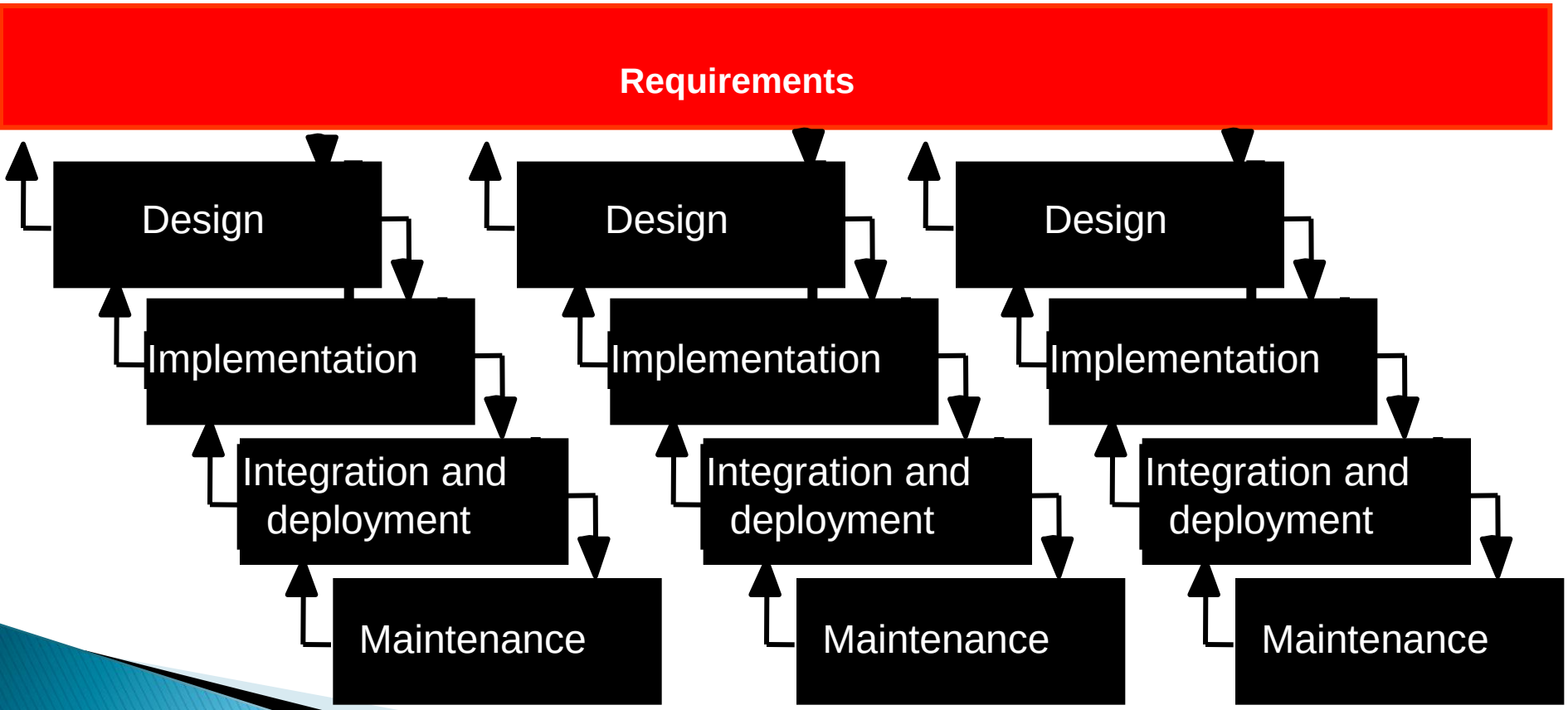
INCREMENTAL & EVOLUTIONARY (ITERATIVE) DELIVERY

The requirements are not known at project inception (evolutionary development)



The full set of requirements are known up front (incremental development)

INCREMENTAL DEVELOPMENT COMBINED WITH WATERFALL



ΣΥΓΚΡΙΣΗ ΤΩΝ ΜΟΝΤΕΛΩΝ ΚΥΚΛΟΥ ΖΩΗΣ

Μοντέλο	Μέγεθος εφαρμογών	Μεταβολές στις απαιτήσεις	Προσαρμοστικότητα στον κατασκευαστή	Διάδοση
Καταρράκτη	Μικρό έως μεσαίο	Ανεπιθύμητες	Καμία	Μεγάλη με τάση μείωσης
Πρωτοτυποποίησης	Μικρό ως μεσαίο	Δεκτές	Μικρή	Μικρή με τάση αύξησης
Λειτουργικής επαύξησης	Μεσαίο ως μεγάλο	Ανεπιθύμητες	Καμία	Μικρή με τάση μείωσης
Σπειροειδές	Μεσαίο ως μεγάλο	Δεκτές	Αρκετή	Μικρή με τάση μείωσης

Η ΕΝΟΠΟΙΗΜΕΝΗ ΠΡΟΣΕΓΓΙΣΗ (UNIFIED PROCESS, UP)

- Εφαρμόζεται επαναληπτικά
- Το σύστημα χτίζεται σταδιακά
- Είναι αρχιτεκτονικοκεντρική
- Βασίζεται στις περιπτώσεις χρήσης
- Χρησιμοποιεί αρχές της αντικειμενοστρεφούς προσέγγισης – Διαγράμματα της Γλώσσας UML (Unified Modeling Language)

ΤΙ ΕΙΝΑΙ Η UP;

- Η UP είναι μια μεθοδολογία
  - Μια διαδικασία μηχανικής λογισμικού
  - Μια διαδικασία για την ανάπτυξη λογισμικού
- Είναι ένα σύνολο γνώσης για το πως να αναπτύξεις λογισμικό
- Είναι μια διαδικασία που χρησιμοποιεί τη UML γλώσσα για να περιγράψει τα παραδοτέα
- Η UP προσδιορίζει
  - Ποιος κάνει τι;
  - Πότε το κάνει;
  - Πώς το κάνει;

  έχοντας πάντα ως στόχο την ανάπτυξη λογισμικού

Η UP ΕΙΝΑΙ ΜΙΑ ΕΠΑΝΑΛΗΠΤΙΚΗ ΔΙΑΔΙΚΑΣΙΑ ΚΑΙ ΕΧΕΙ ΤΕΣΣΕΡΙΣ ΦΑΣΕΙΣ

- Οι φάσεις του κύκλου ζωής είναι τέσσερις:
  - Σύλληψη (inception)
  - Επεξεργασία (elaboration)
  - Κατασκευή (construction)
  - Μετάβαση (transition)

Η ΦΑΣΗ ΤΗΣ ΣΥΛΛΗΨΗΣ (INCEPTION)

- Στη φάση αυτή ορίζουμε
  - Το όραμα που έχουμε για το σύστημα
  - Τη στρατηγική που θέλουμε να υλοποιήσουμε με την ανάπτυξη του συστήματος
  - Τις υψηλού επίπέδου απαιτήσεις και περιπτώσεις χρήσης του συστήματος
  - Τους κινδύνους και τα κέρδη από την ανάπτυξη του συστήματος
  - Παραγωγή αρχικών εκτιμήσεων για το κόστος και διάρκεια των εργασιών ανάπτυξης
- Η φάση της σύλληψης δεν είναι καταγραφή απαιτήσεων. Είναι περισσότερο κάτι σαν μελέτη σκοπιμότητας ανάπτυξης του συστήματος

Η ΦΑΣΗ ΤΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ (ELABORATION)

- Στη φάση αυτή:
  - Αναπτύσσουμε και συγκεκριμενοποιούμε τους στρατηγικούς στόχους του συστήματος
  - Καταγράφουμε τις **απαιτήσεις**
  - Αναλύουμε τις απαιτήσεις λεπτομερώς
  - Προσδιορίζουμε επακριβώς το **αντικείμενο των εργασιών** του έργου (scope)
  - Αναπτύσσουμε τη **βασική αρχιτεκτονική** του συστήματος
  - **Αντιμετωπίζουμε τους κινδύνους** που εντοπίσαμε στη φάση της σύλληψης
  - **Βελτιώνουμε**, συγκεκριμενοποιούμε και επικαιροποιούμε τις **προβλέψεις** σχετικά με το κόστος, χρόνο, πόρους που απαιτεί το έργο

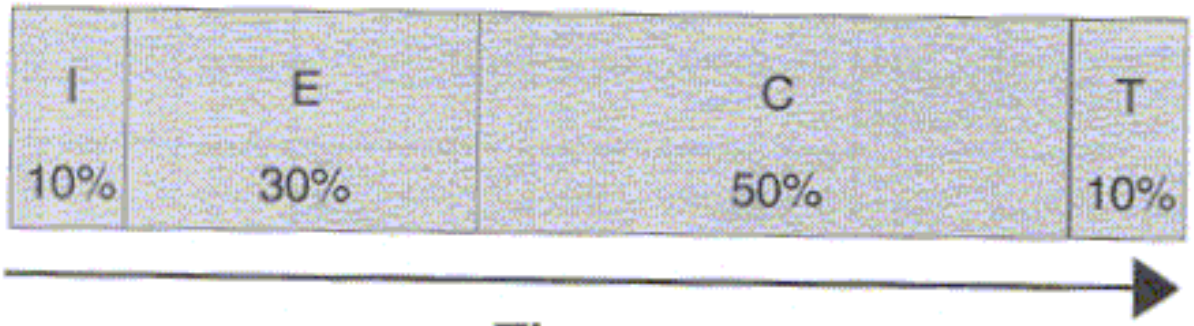
Η ΦΑΣΗ ΤΗΣ ΚΑΤΑΣΚΕΥΗΣ (CONTRUCTION)

- Είναι η φάση κατά την οποία υλοποιούμε και ελέγχουμε τμήματα του συστήματος για τα οποία:
  - Έχουν διευκρινιστεί οι απαιτήσεις
  - Και έχουν αντιμετωπισθεί οι κίνδυνοι

Η ΦΑΣΗ ΤΗΣ ΜΕΤΑΒΑΣΗΣ (TRANSITION)

- Είναι η φάση κατά την οποία το τελικό προϊόν δίνεται στον χρήστη
- Είναι η φάση η οποία έχει μεγάλο βαθμό μεταβλητότητας η οποία εξαρτάται από το συγκεκριμένο έργο και μπορεί να περιλαμβάνει:
  - Εκπαίδευση χρηστών
  - Μεταφορά δεδομένων
  - Δοκιμαστική λειτουργία νέου συστήματος
  - Παράλληλη λειτουργία παλαιού και νέου συστήματος
  - Beta testing
  - ….

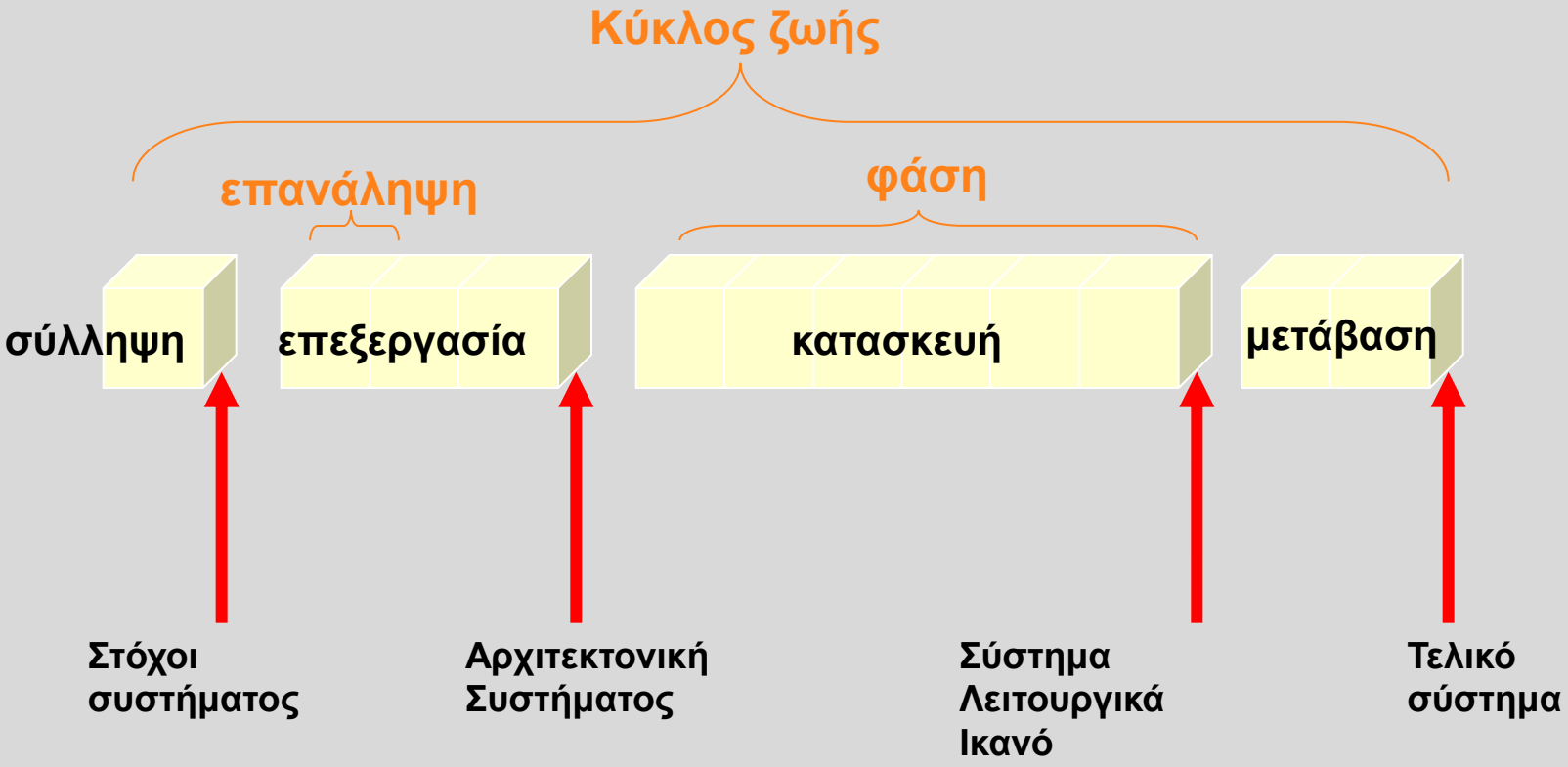
Η ΔΙΑΡΚΕΙΑ ΤΩΝ ΦΑΣΕΩΝ



Typical time line for initial development cycles

- Ο Krutchen δίνει παράδειγμα για ένα έργο διάρκειας 2 ετών
  - Φάση σύλληψης – 2 ½ μήνες
  - Φάση Επεξεργασίας – 7 μήνες
  - Φάση Κατασκευής – 12 μήνες
  - Φάση Μετάβασης – 2 ½ μήνες

ΟΙ ΦΑΣΕΙΣ ΤΟΥ ΚΥΚΛΟΥ ΖΩΗΣ ΣΤΗΝ UP – ΒΑΣΙΚΑ ΟΡΟΣΗΜΑ



Η ΣΥΝΟΛΙΚΗ ΕΙΚΟΝΑ

Σε κάθε φάση εκτελείς όλα τα workflow

Φάσεις

Σύλληψη **Επεξεργασία** **Κατασκευή** **Μετάβαση**

Βασικές Ροές Εργασιών

Σχεδιασμός Επιχειρηματικών Διαδικασιών

Απαιτήσεις

Ανάλυση και Σχεδιασμός

Υλοποίηση

Έλεγχος

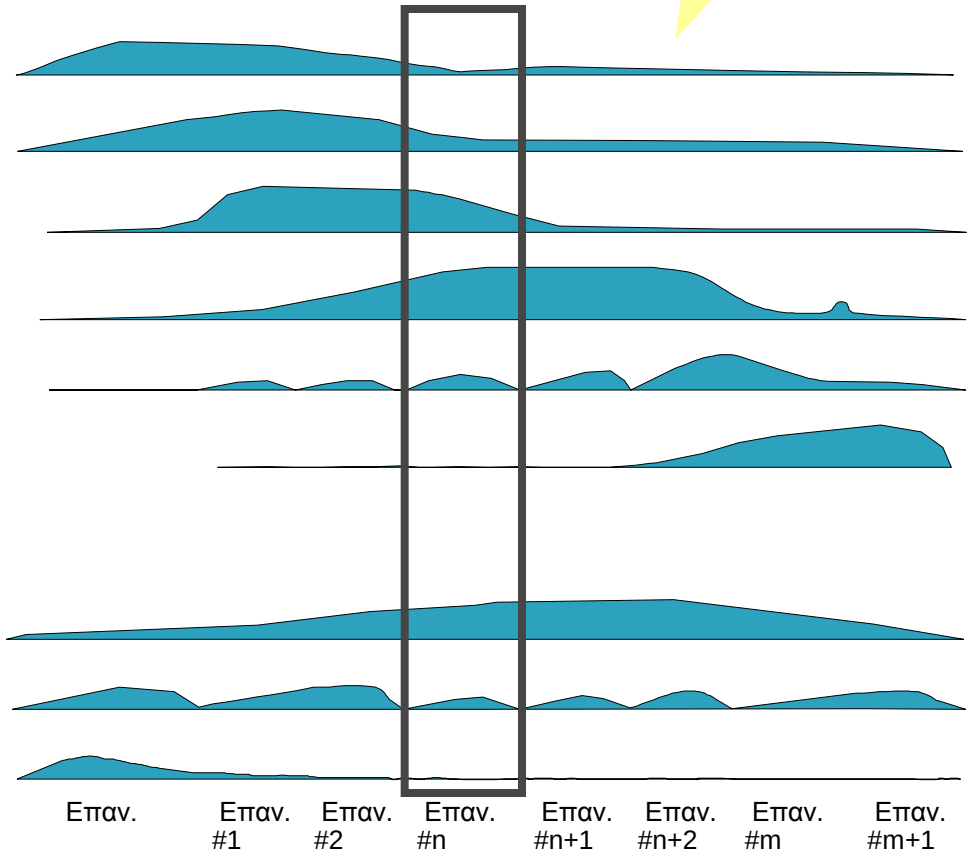
Διάταξη (deployment)

Υποστηρικτικές Ροές Εργασιών

Διαχείριση Σχηματισμών

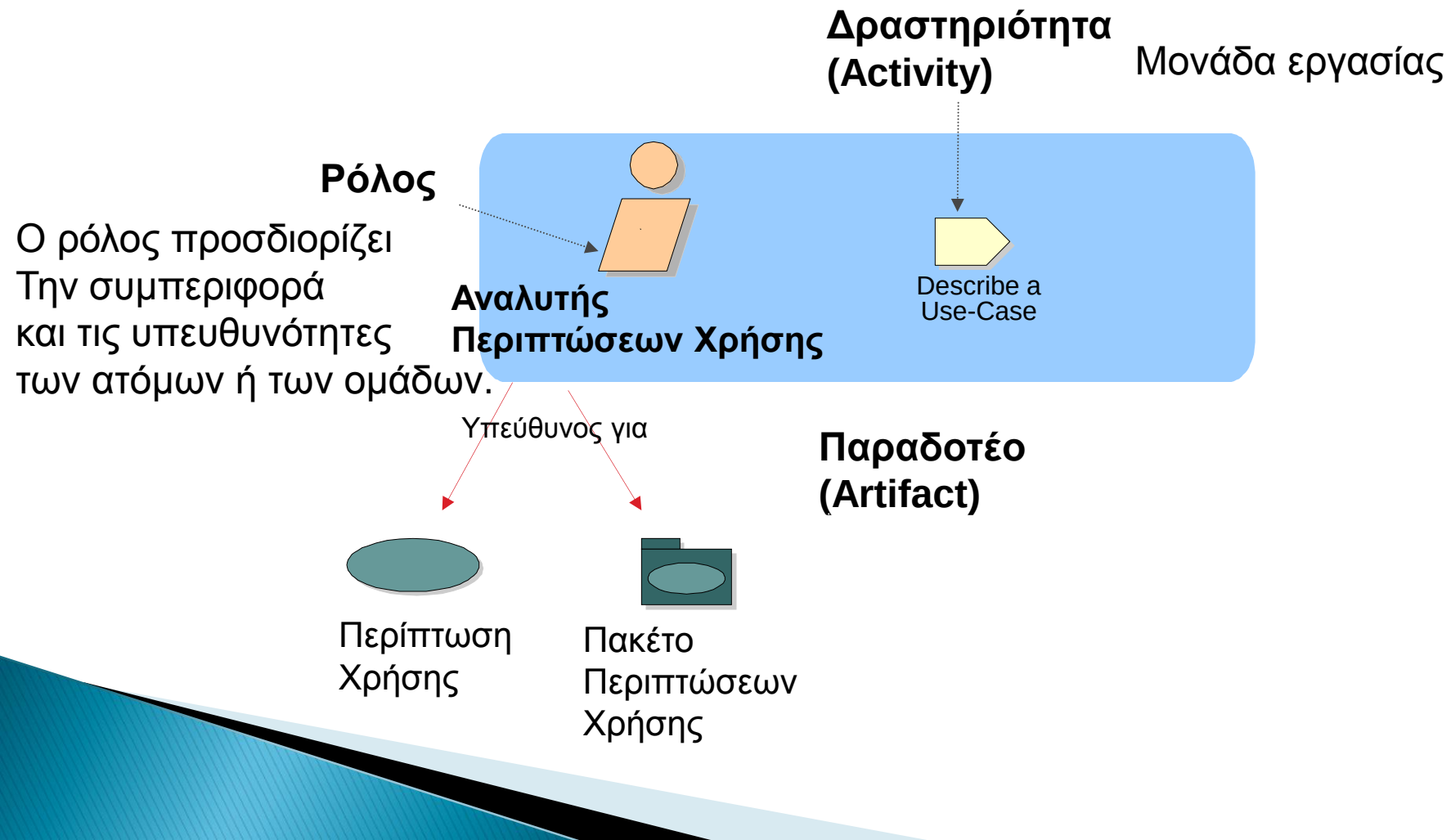
Διαχείριση έργου

Περιβάλλον



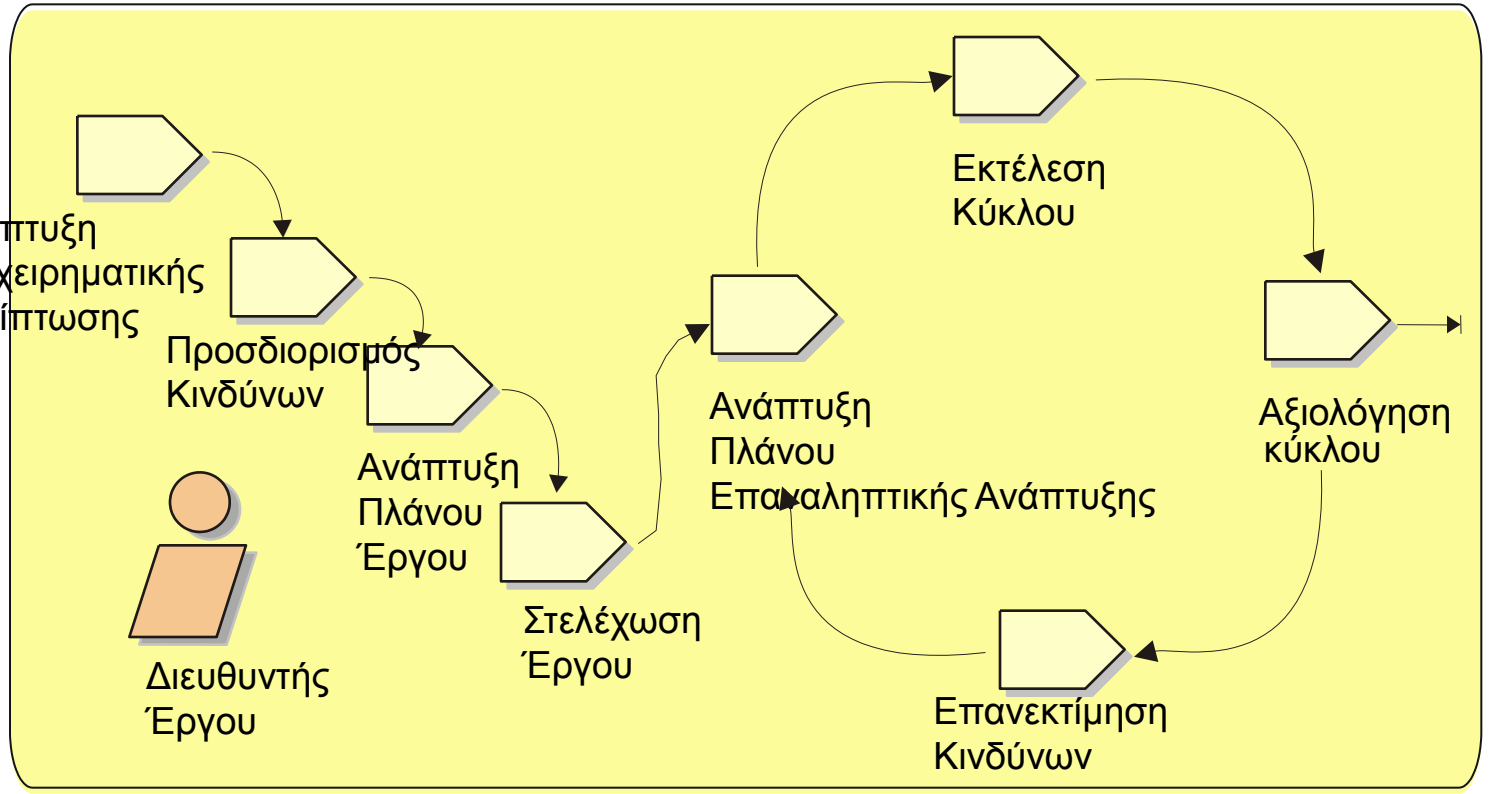
Επαναλήψεις

Ο ΣΥΜΒΟΛΙΣΜΟΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΕΙΤΑΙ ΓΙΑ ΤΗΝ ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΔΙΑΔΙΚΑΣΙΑΣ



Ο ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΠΟΡΩΝ

ΠΑΡΑΔΕΙΓΜΑ ΜΙΑΣ ΡΟΗΣ ΕΡΓΑΣΙΩΝ … ΤΟ WORKFLOW ΔΙΑΧΕΙΡΙΣΗΣ ΕΡΓΩΝ

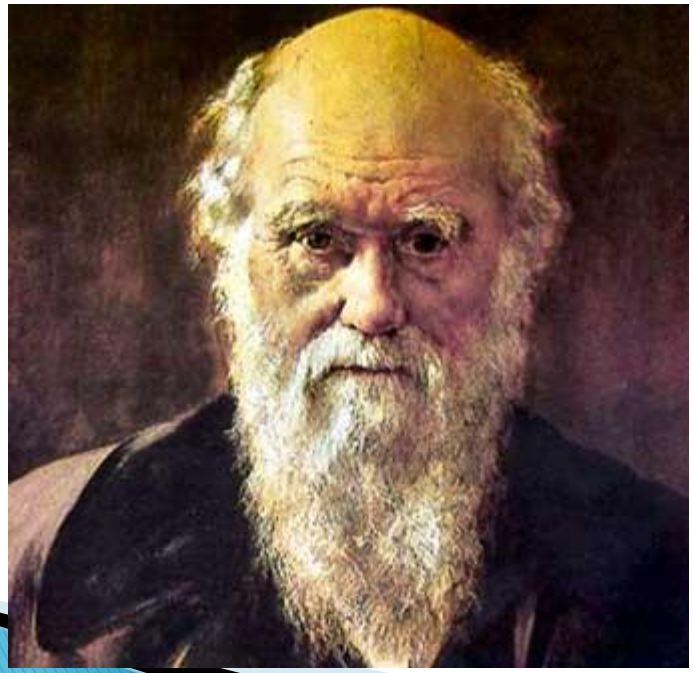


ΠΙΟ ΣΥΓΧΡΟΝΕΣ ΜΕΘΟΔΟΛΟΓΙΕΣ:

ΕΥΕΛΙΚΤΕΣ ΜΕΘΟΔΟΙ (AGILITY) ΚΑΙ ΑΚΡΑΙΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

WHY CHOOSE "AGILE"?

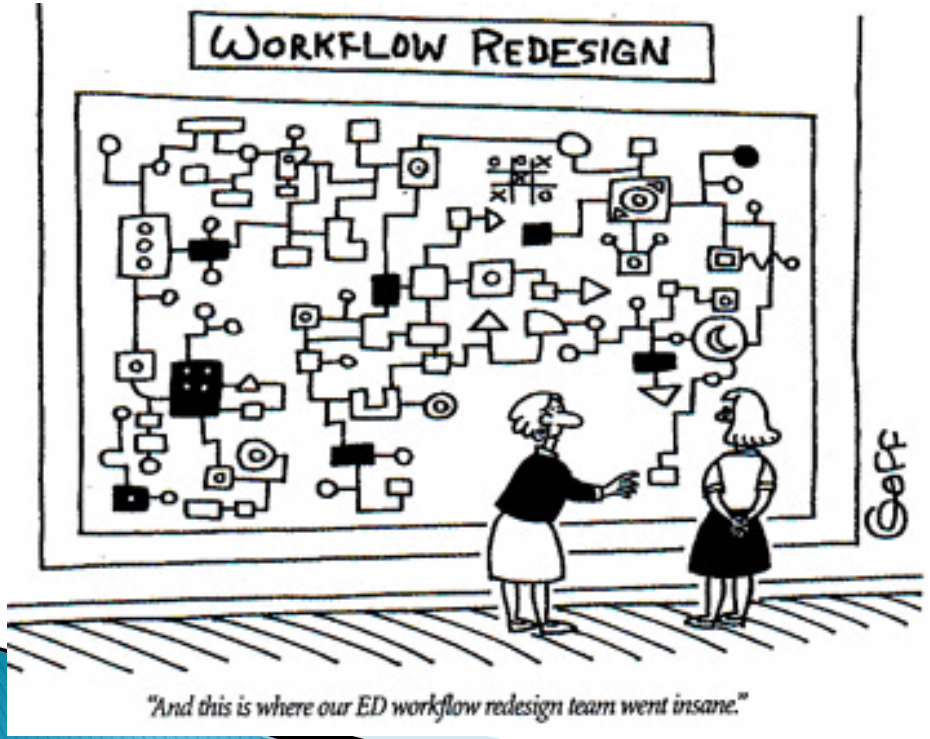
- "It is not the strongest of the species that survive, nor the most intelligent, but the ones most responsive to change."



- Charles Darwin, The Origin of Species

WHY CHOOSE "AGILE"?

"When the process is too complicated for the defined approach, the empirical approach is the appropriate choice."



Process Dynamics, Modeling, and Control,
Ogunnaikeand Ray, Oxford University Press, 1992

DEFINED PROCESS VS. EMPIRICAL

Defined Process Management

Great for known activity

DEFINED PROCESS VS. EMPIRICAL

Not great for unknown activity

$7 million budget
$120 million final

ΕΥΕΛΙΚΤΕΣ ΜΕΘΟΔΟΙ (AGILE METHODS)

- Ευελιξία στον προγραμματισμό (Agility) είναι η ικανότητα της προσαρμογής και επαναπροσδιορισμού ενός αναπτυσσόμενου και συνεχώς εξελισσόμενου συστήματος στην περίπτωση που εμφανίζονται αλλαγές στις αρχικές θεωρήσεις και παραδοχές.
- Οι Ευέλικτες μέθοδοι είναι:
  - Επαναληπτικές (Iterative)
  - Επαυξητικές (Incremental)
  - Αυτό-διοργανούμενες (Self-Organizing)
  - Προκύπτουσες (Emergent)

THE AGILE MANIFESTO

Individuals and interactions	over	Process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

ΑΚΡΑΙΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (XP – PROGRAMMING) – KENT BECK

- Ένας ελαφρύς, αποτελεσματικός, χαμηλού-κινδύνου, ευέλικτος, προβλέψιμος, επιστημονικός και ευχάριστος τρόπος για την ανάπτυξη λογισμικού
- Βασίζεται σε τέσσερις αξίες στην **απλότητα, επικοινωνία, ανατροφοδότηση και κουράγιο**. Η αποτελεσματικότητα της οφείλεται στη στενή συνεργασία της ομάδας κάτω από απλές πρακτικές με συχνή ανατροφοδότηση που τους επιτρέπει να αξιολογούν την πρόοδο τους και να προσαρμόζουν τις πρακτικές στις τρέχουσες ανάγκες.
- Το πρώτο XP-πρόγραμμα ήταν το πρόγραμμα μισθοδοσίας στην Chrysler Comprehensive Compensation (C3), (Beck – Highsmith, 1998).

ΟΙ ΤΕΣΣΕΡΙΣ ΑΡΧΕΣ - *(VALUES)*

- **Επικοινωνία – *Communication***
  - **Απλότητα - *Simplicity***
- **Ανατροφοδότηση - *Feedback***
  - **Κουράγιο - *Courage***

Επικοινωνία

- Η κοινή κατανόηση των προβλημάτων του λογισμικού απαιτεί την *διαπροσωπική επικοινωνία.* Οτιδήποτε εμποδίζει την αμεσότητα αυτή πρέπει να αποβληθεί.

ΕΠΙΚΟΙΝΩΝΙΑ - *(ΣΥΝΕΧΕΙΑ..)*

Οι ομάδες προγραμματισμού XP:

- Χρησιμοποιούν μια κοινή Αρχιτεκτονική εικόνα του συστήματος
- Εργάζονται σε ανοικτό χώρο εργασίας
- Διαρκώς ολοκληρώνουν τον κώδικα
- Επικοινωνούν με ένα πελάτη που βρίσκεται διαρκώς μαζί τους
- Προγραμματίζουν σε ζεύγη
- Κατέχουν όλοι τον κώδικα
- Διαρκώς σχεδιάζουν τεστ ελέγχου

ΑΠΛΟΤΗΤΑ - *(SIMPLICITY)*

Οι ομάδες προγραμματισμού XP:

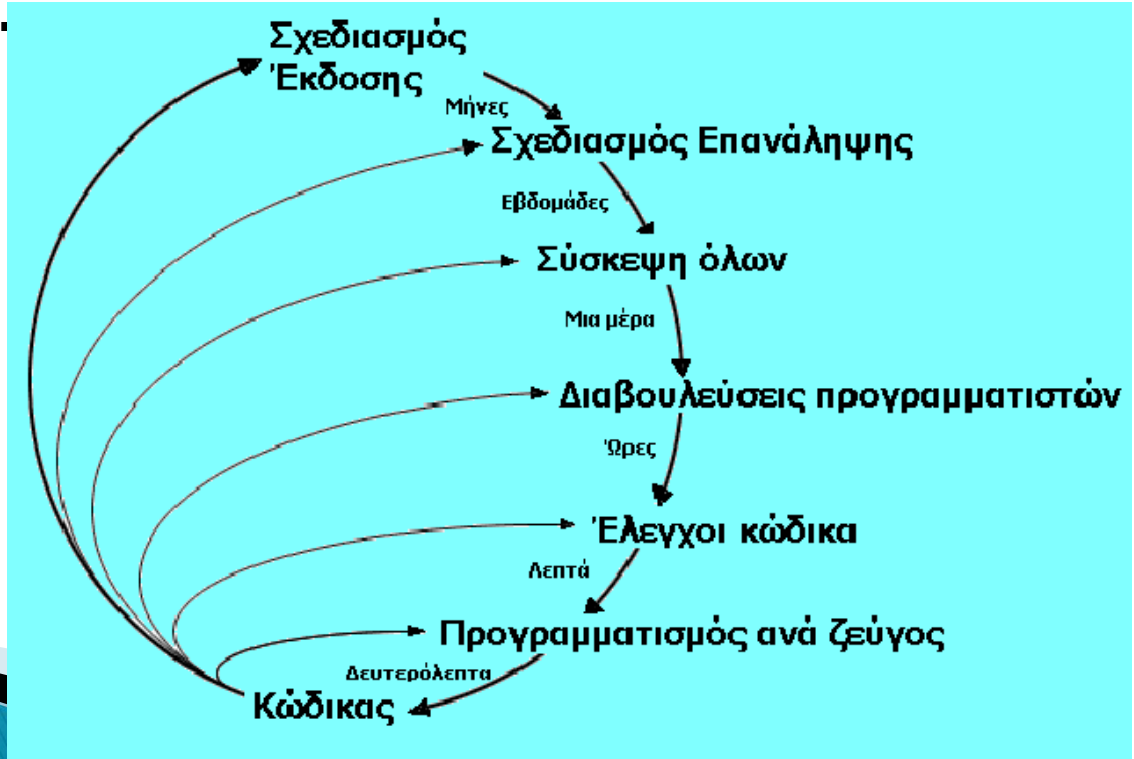
- **Εκτελούν το ποιο απλό σχέδιο που πιθανόν θα δουλέψει**
- **Συνεχώς απλοποιούν και βελτιώνουν την ανάπτυξη του κώδικα με αναδόμηση**

ΑΝΑΤΡΟΦΟΔΟΤΗΣΗ - *(FEEDBACK)*

- **Συγγραφή και χρήση Test cases πριν την παραγωγή κώδικα**

- **Ανάπτυξη σε μικρές εκδόσεις**
  και σε μικρότερες επαναλήψεις
  και σε μικρότερες εργασίες
  και σε ακόμη μικρότερα tests.

..



ΚΟΥΡΑΓΙΟ - *(COURAGE)*

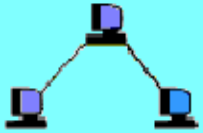
Τα μέλη της ομάδας XP δεν φοβούνται να:

- **Σταματούν όταν κουράζονται**
- **Να αφήνουν τις οικονομικές αποφάσεις στους πελάτες**
- **Προτείνουν στους πελάτες να αλλάξουν την εμβέλεια μιας έκδοσης**
- **Να ζητούν βοήθεια όταν χρειάζεται**
- **Y A G N I** ***(You're not Gonna Need It!)***
- **Αλλάξουν την σχεδίαση και τον κώδικα**
- **Πετάξουν κώδικα που δεν ικανοποιεί**
- **Αλλάξουν την διαδικασία ανάπτυξης όταν δεν λειτουργεί**

ΟΙ 12 ΠΡΑΚΤΙΚΕΣ ΤΟΥ EXTREME PROGRAMMING (XP PRACTISES)

- Το παιχνίδι του σχεδιασμού (The Planning Game)
- Μικρές Εκδόσεις (Small Releases)
- Αρχιτεκτονική Εικόνα (Metaphor)
- Απλή Σχεδίαση (Simple Design)
- Έλεγχοι πριν την κωδικοποίηση (Testing)
- Ανακατασκευή Κώδικα (Refactoring)
- Προγραμματισμός ανά ζεύγη (Pair Programming)
- Συλλογική ιδιοκτησία κώδικα (Collective Ownership)
- Διαρκείς ενοποιήσεις του κώδικα (Continuous Integration)
- Υποφερτός ρυθμός εργασίας (Sustainable pace)
- Διαρκή παρουσία πελάτη (On-site Customer)
- Σταθερές Κωδικοποίησης (Coding Standards)

ΟΙ ΠΡΑΚΤΙΚΕΣ ΣΕ ΕΙΚΟΝΕΣ...

Το παιχνίδι του σχεδιασμού  =συμβολή του πελάτη	Προγραμματισμός ανά ζεύγη  =λιγότερα λάθη στον κώδικα	Έλεγχοι πριν την κωδικοποίηση  =ποιοτικό λογισμικό	Ανακατασκευή κώδικα  =καθαρός κώδικας
Σταθερές κωδικοποίησης  =καλύτερη επικοινωνία	Απλή Σχεδίαση  =ευελιξία	Μικρές εκδόσεις  =προσαρμοστικότητα	Διαρκείς ενοποιήσεις του κώδικα  =διαρκής ανάδραση
Συλλογική ιδιοκτησία κώδικα  =διαμοιρασμός της γνώσης	Διαρκή παρουσία Πελάτη  =ξεκαθάρισμα των απαιτήσεων	Αρχιτεκτονική εικόνα  =κατανόηση του συστήματος	Υποφερτός ρυθμός εργασίας  =αποδοτικότητα

ΤΟ ΠΑΙΧΝΙΔΙ ΤΟΥ ΣΧΕΔΙΑΣΜΟΥ *(PLANNING GAME)*

- Στο παιχνίδι του σχεδιασμού είναι μία δραστηριότητα με την οποία η ομάδα ανάπτυξης του συστήματος και οι πελάτες αποφασίζουν τι θα γίνει σε **κάθε έκδοση** – *release (3-6 μήνες)* και **κάθε επανάληψη** – *iteration (1-3 εβδομάδες).*

- Το παιχνίδι του σχεδιασμού *(planning game)* τρέχει για κάθε επανάληψη για να καθορίσει ποια λειτουργία θα δρομολογηθεί στην επόμενη **ενσωμάτωση**. Οι προγραμματιστές λαμβάνουν τις **τεχνικές αποφάσεις** - εκτιμήσεις και οι πελάτες τις **επιχειρησιακές αποφάσεις**.

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΑΝΑ ΖΕΥΓΗ *(PAIR PROGRAMMING)*

- **Η ανάπτυξη του προγράμματος βασίζεται πάντα σε δύο άτομα που μοιράζονται τον ίδιο υπολογιστή.**
- **Συνήθως ο ένας γράφει ενώ ο άλλος βλέπει, διορθώνει και σκέφτεται ένα βήμα μπροστά.**
- **Τα ζευγάρια εναλλάσσονται συνεχώς με αποτέλεσμα να μεταφέρεται η εμπειρία και η γνώση σε όλα τα μέλη της ομάδας.**
- **Η πιο βασική πρακτική μαζί με την πρακτική των ελέγχων πριν την κωδικοποίηση**

ΕΛΕΓΧΟΙ ΠΡΙΝ ΤΗΝ ΚΩΔΙΚΟΠΟΙΗΣΗ *(TEST-FIRST-DESIGN)*

- Οι προγραμματιστές γράφουν περιπτώσεις ελέγχου πριν αρχίσουν τη συγγραφή κώδικα.
- Η ομάδα δημιουργεί αυτοματοποιημένα τεστ μονάδας – **unit tests** και τεστ αποδοχής - **acceptance tests** τα οποία εφαρμόζονται συχνά.
- Το πρόγραμμα ελέγχεται κάθε φόρα που προστίθεται επιπλέον κώδικας.
- Για κάθε κομμάτι κώδικα δημιουργείται αντίστοιχο τεστ.
- Τα αυτοματοποιημένα τεστ τρέχουν σε όλο το πρόγραμμα και διασφαλίζουν ότι όλα λειτουργούν σωστά.

ΑΝΑΚΑΤΑΣΚΕΥΗ ΚΩΔΙΚΑ *(REFACTORING)*

- **Η τεχνική βελτίωσης του υπάρχοντος κώδικα δίχως να μεταβληθεί η λειτουργικότητά του.**
- **Ο κώδικας απλοποιείται και γίνεται πιο ευέλικτος και κατανοητός.**
- **Μελλοντικές αλλαγές ή προσθήκες είναι εύκολα υλοποιήσιμες.**

ΣΤΑΘΕΡΕΣ ΚΩΔΙΚΟΠΟΙΗΣΗΣ *(CODING STANDARDS)*

- **Η συγγραφή κώδικα είναι μία *ομαδική εργασία.* Κατά καιρούς διαφορετικά άτομα θα εργαστούν σε διαφορετικά τμήματα κώδικα. Οι διαφορές στο ύφος καθιστούν συχνά τον κώδικα δύσκολο αντικείμενο εργασίας.**

- **Ο κώδικας συχνά αναδομείται και η αρχιτεκτονική των συστημάτων αλλάζει. Προκειμένου να υπάρχει αποτελεσματικότητα πρέπει ο κώδικας όλης της ομάδας να μοιάζει σαν να γράφτηκε από ένα μόνο άτομο και για να επιτευχθεί αυτό απαιτούνται πρότυπα κώδικα και σταθερές κωδικοποίησης.**

ΑΠΛΗ ΣΧΕΔΙΑΣΗ
(SIMPLE DESIGN)

Ο σωστός σχεδιασμός μίας εφαρμογής πρέπει:

- ✓**να "τρέχει" σε όλα τα test**
- ✓**να έχει απλή λογική**
- ✓**να δηλώνει κάθε πρόθεση σημαντική για τους προγραμματιστές**
- ✓**να έχει τις λιγότερες δυνατές κλάσεις και μεθόδους.**

ΜΙΚΡΕΣ ΕΚΔΟΣΕΙΣ
(SMALL RELEASES)

- Οι εκδόσεις πρέπει να είναι όσο το δυνατόν μικρότερες. Κάθε έκδοση περιέχει μόνο τα πιο *σημαντικά χαρακτηριστικά* (αυτά που έχουν συμφωνηθεί)

- Οι XP ομάδες πρέπει να δίνουν εκδόσεις στο τέλος κύκλων έκδοσης. Ο κύκλος έκδοσης πρέπει να είναι όσο το δυνατόν μικρότερος, χωρίς όμως να υπάρχουν χαρακτηριστικά που δεν δουλεύουν απόλυτα.

- Πριν την έκδοση υλοποιείται τεστ-αποσφαλμάτωσης και μετά γίνεται η ενσωμάτωση του κώδικα.

ΔΙΑΡΚΕΙΣ ΕΝΟΠΟΙΗΣΕΙΣ ΤΟΥ ΚΩΔΙΚΑ *(CONTINUOUS INTEGRATION)*

- Οι XP ομάδες εργάζονται με μικρά βήματα και ***ενσωματώνουν τον κώδικά τους αρκετές φορές την ημέρα.*** Αυτό σημαίνει πως προβλήματα ενσωμάτωσης ανακαλύπτονται γρήγορα από τη στιγμή που εμφανιστούν και είναι πιο εύκολο να διευθετηθούν.

- Η συνεχής ενσωμάτωση κώδικα στο πρόγραμμα συνεπάγεται ότι δεν υπάρχουν μεγάλες εξελίξεις που είναι ασυμβίβαστες με το υπόλοιπο πρόγραμμα και ότι όλοι μπορούν να εργάζονται πάνω στην πιο πρόσφατη έκδοση του συστήματος.

ΣΥΛΛΟΓΙΚΗ ΙΔΙΟΚΤΗΣΙΑ ΚΩΔΙΚΑ *(COLLECTIVE OWNERSHIP)*

- Καθένας στην ομάδα έχει την ***δικαιοδοσία να αλλάξει οπιδήποτε στον κώδικα,*** αρκεί να κάνει τις αλλαγές με τον συνάδελφό του, να ακολουθούν τα πρότυπα κώδικα, και να διασφαλίσουν ότι όλα τα τεστ δουλεύουν, όταν τελειώσουν τις αλλαγές.

- Αυτό αφαιρεί τις δυσχέρειες και τις αρχιτεκτονικές διαστρεβλώσεις που μπορούν να εμφανιστούν με τη μεμονωμένη-προσωπική ιδιοκτησία κώδικα.

ΔΙΑΡΚΗ ΠΑΡΟΥΣΙΑ ΠΕΛΑΤΗ *(ON-SITE CUSTOMER)*

- Καμία γραπτή απαίτηση δεν είναι πλήρης και σαφής. Οι προγραμματιστές χρειάζονται *πάντα επικοινωνία με τον πελάτη για διευκρινίσεις*, ανεξάρτητα από το πόση προσπάθεια καταβλήθηκε στην αρχική προδιαγραφή απαιτήσεων.

- Μία XP ομάδα παρακάμπτει όλη αυτή την προσπάθεια αποτελεσματικής προδιαγραφής και ανάλυσης απαιτήσεων έχοντας κάποιον *πελάτη διαθέσιμο συνέχεια στο χώρο εργασίας.*

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΙΚΟΝΑ *(METAPHOR)*

- Η ***αρχιτεκτονική εικόνα*** **που δίνει συνοχή και συνέπεια στον τρόπο με τον οποίο η ομάδα αναπτύσσει το σύστημα**

- **Μέσα στις ιδιότητες της αρχιτεκτονικής εικόνας περιλαμβάνεται και η *ονομασία των διαφόρων κλάσεων και μεθόδων*. Είναι πολύ σημαντική η ονοματολογία στην κατανόηση της αρχιτεκτονικής του συστήματος και στη δυνατότητα επαναχρησιμοποίησης κώδικα.**

ΥΠΟΦΕΡΤΟΣ ΡΥΘΜΟΣ ΕΡΓΑΣΙΑΣ
(SUSTAINABLE PACE)

- **Η ανάπτυξη λογισμικού είναι μία δημιουργική εργασία και κανείς δεν μπορεί να είναι παραγωγικός και δημιουργικός αν είναι εξαντλημένος.**

- **Περιορίζοντας *τις ώρες εργασίες σε 40-ώρες ανά εβδομάδα* διατηρεί την ομάδα ξεκούραστη, μειώνει τον κύκλο εργασιών του προσωπικού, και βελτιώνει την ποιότητα του ολοκληρωμένου προϊόντος.**

ΑΚΡΑΙΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ…

Είναι μια πειθαρχημένη προσέγγιση όπου πρέπει να:

- Γράφετε test πριν τον κώδικα
- Προγραμματίζετε σε ζεύγη
- Ολοκληρώνετε τακτικά τον κώδικα
- Ξεκουράζονται οι προγραμματιστές
- Επικοινωνούν οι προγραμματιστές με τους πελάτες συνεχώς στο χώρο εργασίας
- Ακολουθούνται οι προτεραιότητες των πελατών
- Αφήνεται το λογισμικό καθαρό και απλό στο τέλος της ημέρας
- Προσαρμόζεστε στις διαδικασίες και πρακτικές του περιβάλλοντος σας

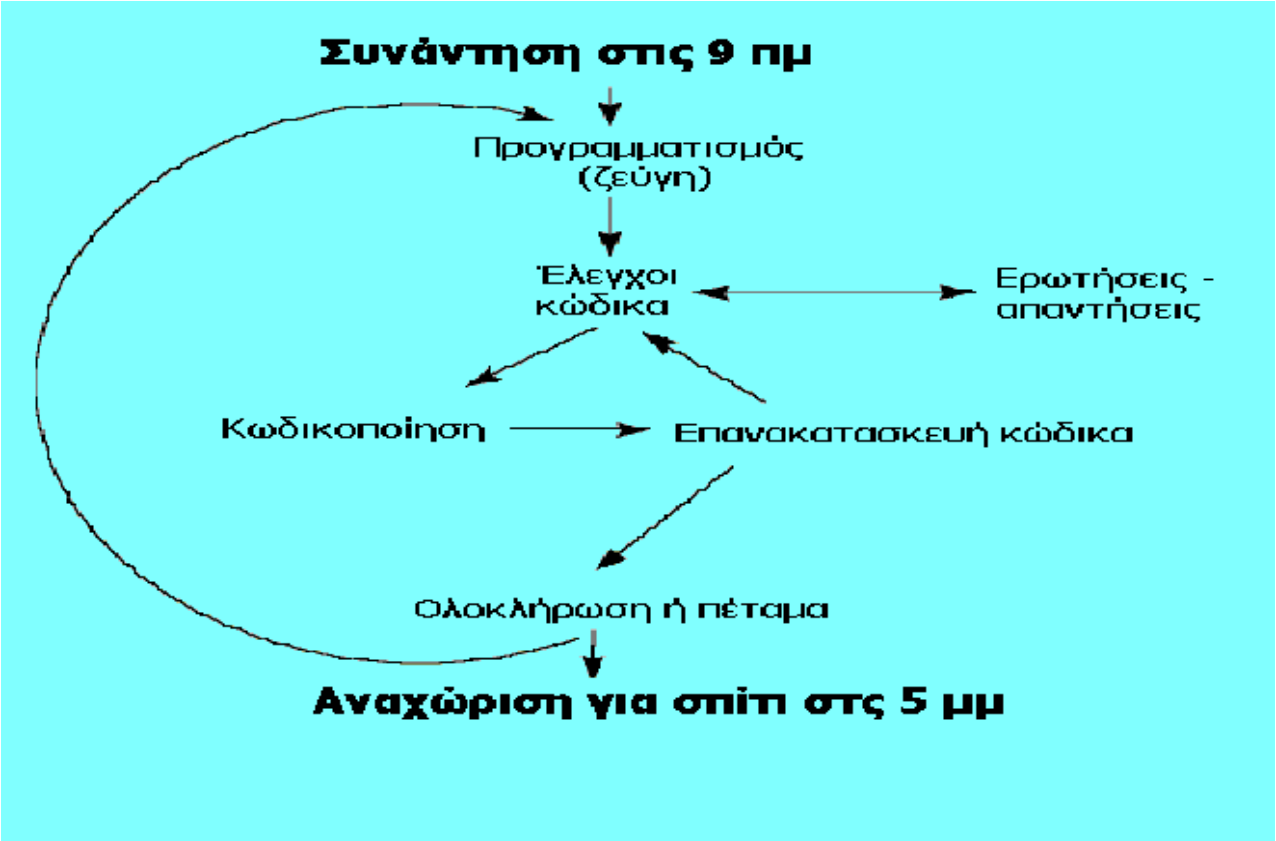
ΓΙΑΤΙ ΣΤΑ ΑΚΡΑ;

- Αν η επιθεώρηση κώδικα είναι ωφέλιμη, τότε κάνετε το συνεχώς *(pair programming)*
- Αν το ο έλεγχος κώδικα είναι ωφέλιμος, τότε κάνετε το συνεχώς *(unit tests - acceptance tests)*
- Αν ο ανασχεδιασμός είναι καλός, τότε κάνετε το συνεχώς *(refactoring)*
- Αν η απλότητα είναι καλή, τότε κάνε το απλούστερο που μπορεί να δουλέψει *(simple design)*

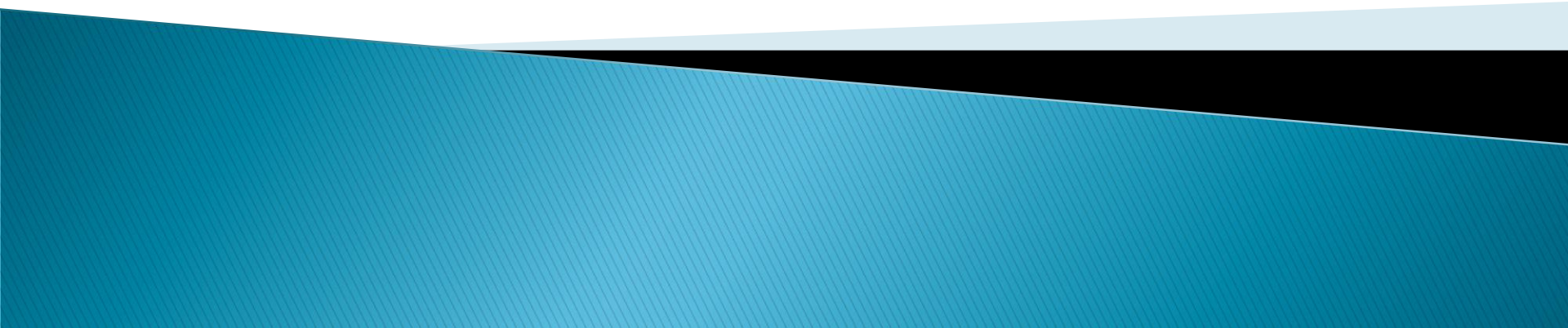
ΓΙΑΤΙ ΣΤΑ ΑΚΡΑ; *(ΣΥΝΕΧΕΙΑ…)*

- Αν η αρχιτεκτονική του συστήματος είναι σημαντική, τότε όλοι θα την ορίζουν και επανακαθορίζουν *(metaphor)*
- Αν οι έλεγχοι ολοκλήρωσης είναι σημαντικοί, τότε να εκτελείς τέτοιους ελέγχους καθημερινά *(continuous integration)*
- Αν η ανατροφοδότηση είναι καλή, τότε να την λαμβάνεις συνεχώς *(pair programming, planning game, on-site customer)*

ΚΥΚΛΟΣ ΑΝΑΠΤΥΞΗΣ

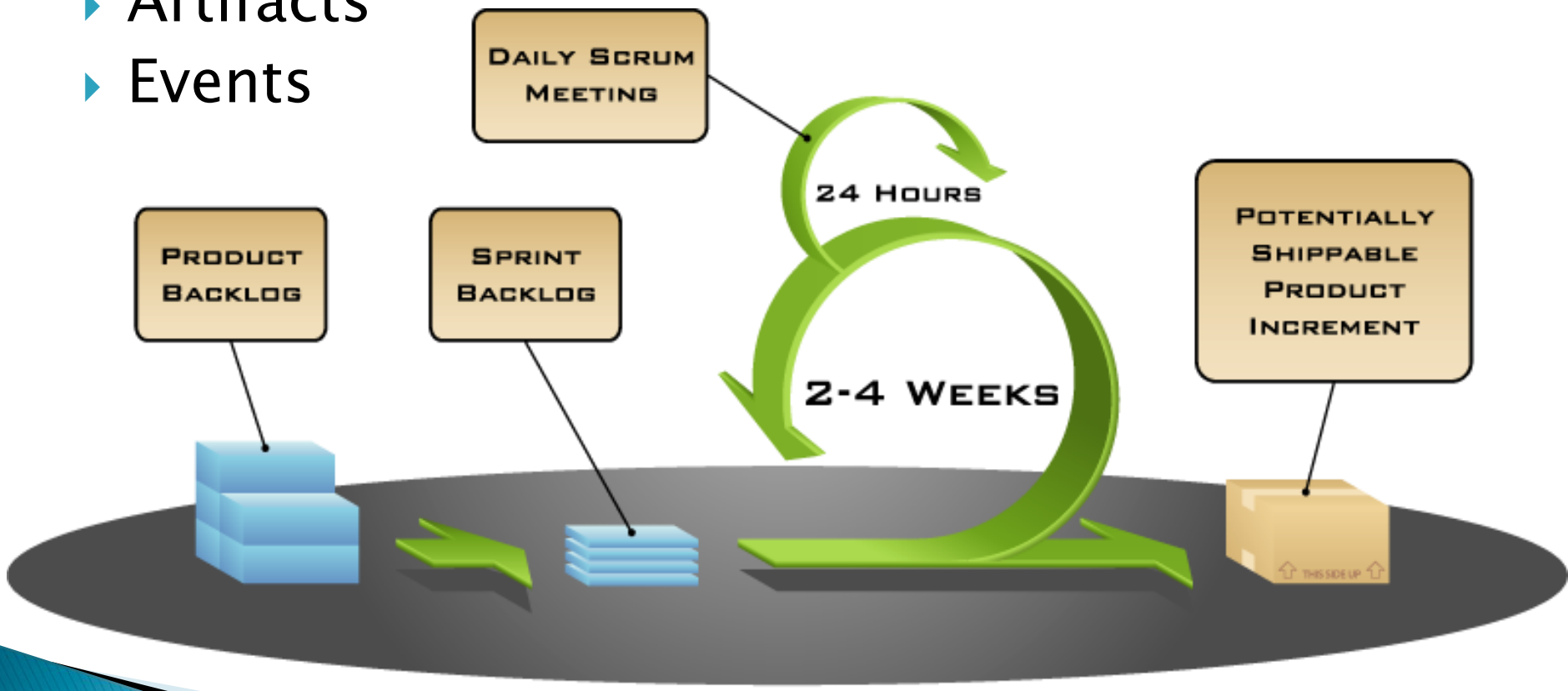


SCRUM



FOR EXAMPLE SCRUM

- Roles
- Artifacts
- Events



PRODUCT BACKLOG

Backlog Description	Initial Estimate	Adjust-ment Factor	Adjusted Estimate	work remaining until completion						
				1	2	3	4	5	6	7
Title Import				256	209	193	140	140	140	140
Project selection or new	3	0.2	3.6	3.6	0	0	0	0	0	0
Template backlog for new projects	2	0.2	2.4	2.4	0	0	0	0	0	0
Create product backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Create sprint backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Display tree view of product backlog, releases, sprints	2	0.2	2.4	2.4	0	0	0	0	0	0
Sprint-1	13	0.2	15.6	16	0	0	0	0	0	0
Create a new window containing product backlog template	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Create a new window containing sprint backlog template	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Burndown window of product backlog	5	0.2	6	6	6	0	0	0	0	0
Burndown window of sprint backlog	1	0.2	1.2	1.2	1.2	0	0	0	0	0
Display tree view of product backlog, releases, prints	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Display burndown for selected sprint or release	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Sprint-2	16	0.2	19.2	19	19	1.2	0	0	0	0
Automatic recalculating of values and totals	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
As changes are made to backlog in secondary window, update burndown graph on main page	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Hide/automatic redisplay of burndown window	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
Insert Sprint capability ... adds summing Sprint row	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Insert Release capability ... adds summary row for backlog in Sprint	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Owner/assigned capability and columns optional	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Print burndown graphs	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Sprint-3	14	0.2	16.8	17	17	17	0	0	0	0
Duplicate incomplete backlog without affecting totals	5	0.2	6	6	6	6	6	6	6	6
Note capability	6	0.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2
What-if release capability on burndown graph	15	0.2	18	18	18	18	18	18	18	18

ΒΑΣΙΚΟΙ ΡΟΛΟΙ ΣΤΗ SCRUM

- Product Owner
- ScrumMaster
- Team

PRODUCT OWNER

- αντιπροσώπευση των συμφερόντων
- έχει λόγο για την εξέλιξη του project
- Χρηματοδότηση
- δημιουργία των αρχικών γενικών απαιτήσεων και των στόχων απόδοσης της επένδυσης
- σχέδια για την διάδοση του προϊόντος στην αγορά.

TEAM

- αρμόδια για την ανάπτυξη της λειτουργικότητας
- self-managing
- self-organizing
- cross-functional
- τα μέλη της ομάδας είναι συλλογικά υπεύθυνα για την επιτυχία κάθε επανάληψης και του project συνολικά.

SCRUMMASTER

- υπεύθυνος για τη διαδικασία Scrum
- διδάσκει τη Scrum σε όλους όσοι συμμετέχουν στο project
- προστατεύει την ομάδα από εμπόδια
- βοηθά θέτοντας ερωτήματα και παρέχοντας συμβουλές
- λειτουργεί με βάση την κουλτούρα του οργανισμού

SPRINT

- Short
- Time boxed

PLANNING MEETING

- Stories
- Breakdown
- Time boxed

DAILY SCRUM/STANDUP

- 15 minutes (at most)
- 3 questions

REVIEW

- Inspection

ΓΙΑΤΙ Η SCRUM ΕΙΝΑΙ ΕΥΕΛΙΚΤΗ;

- Time boxing
  - καθορίζεται το χρονικό διάστημα της κάθε επανάληψης
  - Κατά τη διάρκεια αυτού του χρόνου θα πρέπει κάθε ομάδα να αναλάβει ένα κομμάτι της τελικής λειτουργικότητας του προϊόντος και να προσπαθήσει να κατανοήσει πως θα το φέρει εις πέρας.Η ομάδα επικεντρώνεται μόνο σε αυτούς τους στόχους και όχι στα γενικότερα παραδοτέα του προϊόντος.
- Incremental Delivery
  - Εφόσον κάθε επαυξητικό προϊόν ελέγχεται και κωδικοποιείται ,ο αριθμός των αδυναμιών ποτέ δεν είναι μεγάλος.
- Selforganization of the team

ΔΙΑΘΕΣΙΜΕΣ ΕΥΕΛΙΚΤΕΣ ΜΕΘΟΔΟΙ

- Agile Modeling
- Adaptive Software Development (ASD)
- Crystal methods
- Dynamic System Development Methodology (DSDM)
- eXtreme Programming (XP)
- Feature Driven Development (FDD)
- Lean Development
- Scrum

AGILE PROCESS MATURITY MODEL

1 Core Agile Development

Focus is on construction

Goal is to develop a high-quality system in an evolutionary, collaborative, and self-organizing manner

Value-driven lifecycle with regular production of working software

2 Disciplined Agile Delivery

Extends agile development to address full system lifecycle

Risk and value-driven lifecycle

Self organization within an appropriate governance framework

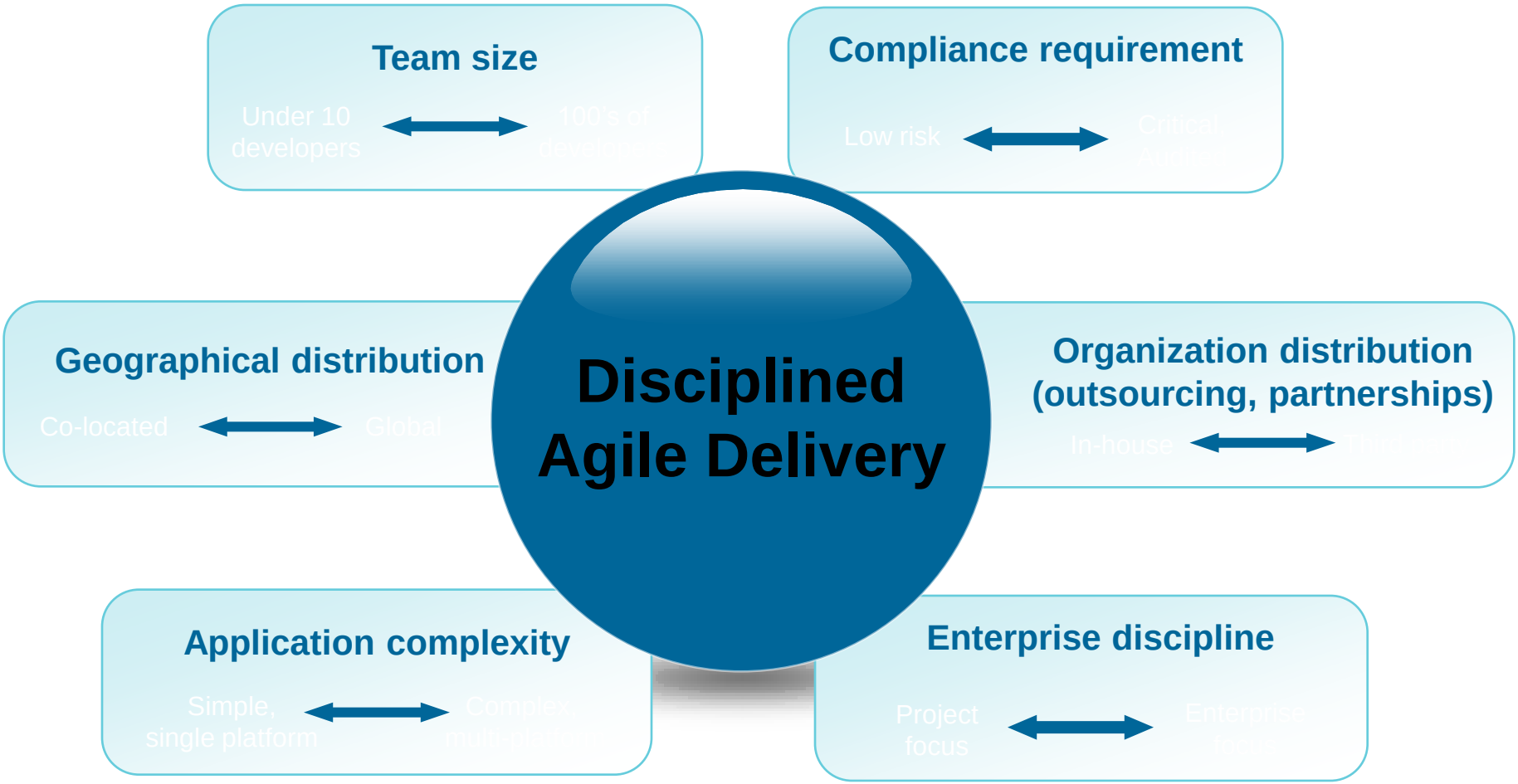
3 Agility at Scale

Addresses one or more scaling factors:

- Team size
- Geographical distribution
- Organizational distribution
- Regulatory compliance
- Environmental complexity
- Enterprise discipline

Christopher de Kok
IBM Rational IT Specialist

WHAT IS AGILITY AT SCALE?



Christopher de Kok
IBM Rational IT Specialist

ΕΠΙΛΟΓΗ ΤΟΥ ΚΥΚΛΟΥ ΖΩΗς

- Εξαρτάται από το έργο
- Επαναληπτικός “iterative” ή αυξητικός “incremental”
- Πόσο καλά έχουμε κατανοήσει τις απαιτήσεις
- Ποιοι είναι οι κίνδυνοι
- Τα χρονικά όρια είναι στενά
- Υπάρχει εμπειρία στην ομάδα και /ή στον πελάτη