Curs 2 - Metodologii de management a proiectelor software

- Concepte utilizate în realizarea proiectelor software
- Definirea și conținutul metodologiilor
- Clasificarea metodologiilor de realizare a proiectelor software
- Metodologii structurate:
  - SSADM
  - MERISE
- Metodologii orientate obiect
  - OMT
  - Proces iterativ de dezvoltare a proiectelor software utilizând UML
  - Metodologia unificată de realizare a proiectelor software (RUP)
- Metodologii bazate pe dezvoltarea rapidă
- Metodologii bazate pe dezvoltarea agilă

Concepte utilizate în realizarea proiectelor software

Concept	Definire
Proces/etapa	Este un ansamblu de activităţi intercorelate, care utilizează resurse în vederea atingerii unei funcţii obiectiv, bine stabilit. Procesele pot fi primare, suport și organizatorice. În anumite metodologii se regăsesc sub denumirea de **cale** sau **flux de lucru**.
Activitate	Cuprinde tipurile de acţiuni întreprinse pentru utilizarea eficientă a resurselor. Este o parte a unui proces. În unele metodologii, pentru conceptul de acţiune se foloseşte conceptul de **fază**, **pas** sau **segment**.
Fază	Reprezintă intervalul de timp cuprins între două puncte-cheie ale unui proces, pe durata căruia este atins un set bine definit de obiective (în Rational Unified Process).
Paşi	Reprezintă o succesiune de activităţi desfăşurate în cadrul etapei de lucru (în metodologia SSADM).

Concepte utilizate în realizarea proiectelor software

Concept	Definire
Sarcinile	Sunt componente ale activităţilor şi constituie un ansamblu de acţiuni de realizarea cărora sunt responsabile persoane sau grupuri de persoane. O sarcină este caracterizată de precondiţii, elemente livrabile şi postcondiţii. Modul de ordonare în timp a sarcinilor, activităţilor, etapelor sau proceselor formează **ciclul de viaţă al sistemului informatic**.
Ciclul de viaţă al sistemului informatic	Este definit prin de modul de ordonare în timp a sarcinilor, activităţilor, etapelor sau proceselor. este un şablon pentru ordonarea activităţilor de realizare a sistemului informatic, cuprinzând intervalul de timp care începe cu decizia de elaborarea a unui sistem informatic şi se încheie cu decizia de abandonare a acestuia şi înlocuirea lui cu un nou sistem informatic.
Ciclul de dezvoltare al sistemului informatic	este cuprins în ciclul de viaţă al sistemului informatic. El cuprinde intervalul de timp de la luarea deciziei de realizare a unui sistem informatic până în momentul intrării sistemului în exploatare.

Definirea şi conținutul metodologiilor

O metodologie de realizare a unui sistem informatic trebuie să cuprindă:

- **etapele/procesele** de realizare a unui sistem informatic structurate în subetape, activități, sarcini și conținutul lor;
- **fluxul** realizării acestor etape/procese, subetape și activități;
- **modalitatea de derulare** a ciclului de viață a sistemului informatic;
- modul de abordare al sistemelor;
- **strategiile de lucru**/metodele de realizare;
- **regulile de formalizare** a componentelor sistemului informatic;
- tehnicile, procedurile, instrumentele, normele și standardele utilizate;
- modalitățile de conducere a proiectului (planificare, programare, urmărire) și modul de utilizare a resurselor financiare, umane și materiale etc.

Clasificarea metodologiilor de realizare a proiectelor software

A. Clasificare după gradul de generalitate:

- ***Metodologiile generale*** au un grad înalt de **generalitate**, pot fi folosite pentru realizarea proiectelor softwaredin domenii diferite. Exemple: SSADM (Structured System Analysis and Design Methodology), MERISE (Méthode d'Etude et de Realization, Informatique pour les Systèmes d'Entreprise), OMT (Object Modeling Technique), RUP (Rational Unified Process).

- ***Metodologiile cadru*** cuprind elemente aplicabile **exclusiv numai unor produse software**. Exemple de metodologii: Selection and Implementation of Integrated Packaged Software (SIIPS) elaborată de **KPMG**. Ea are acceleratori de implementare pentru ORACLE și SAP.

- ***Metodologii specializate*** sunt cele dezvoltate și utilizate pentru implementare a **unui singur produs software**. Exemple: AIM (pentru Oracle E Business Suite), POIS (pentru Sun Systems), Signature (pentru Scala), ASAP (pentru SAP).

Clasificarea metodologiilor de realizare a proiectelor software

B. Clasificare din punct de vedere al modului de abordare al sistemelor:

- ***Metodologiile cu abordare structurată*** au ca principiu de lucru împărţirea sistemului în subsisteme pe baza funcţiilor sistemului (***abordarea funcţională***) sau în funcţie de date (***abordarea bazată pe date***). Propun modelarea datelor separat de modelarea procedurilor. Modelarea procedurilor se face plecând de la ideea că funcţiile sunt active, având un comportament, iar datele sunt afectate de aceste funcţii.

- ***Metodologiile cu abordare orientată obiect*** permit construirea proiectelor softwarefolosind *conceptele tehnologiei orientate obiect*. Tehnologia orientată obiect a apărut odată cu apariţia limbajelor de programare orientate obiect. Primele limbaje orientate obiect au fost SIMULA (1960), SMALLTALK (1970), CLOS, EIFFEL, ACTOR, C++, Object Pascal (1980).

Metodologiile cu abordare structurată

Avantajele utilizării metodologiilor structurate:

- Utilizarea reprezentării **grafice**, la îndemâna atât a analiştilor, cât şi a beneficiarilor;
- Planificarea eficace a proiectului prin **divizarea în subsisteme**;
- Un mediu bine structurat este **flexibil** din punct de vedere al comportamentului;
- Are **obiective clare**, o **arie de cuprindere cunoscută**;
- Posibilitatea **reducerii timpului şi a costului** de dezvoltare a sistemului prin luarea în considerare de la început a **detaliilor**, prin interacţiunea continuă cu beneficiarul;
- Modificarea unei anumite activităţi a sistemului nu conduce la reluarea integrală a studiului.

Metodologiile cu abordare structurată

Exemple de metodologii cu abordare structurată:

- Structured Analysis and Design Information Systems (STRADIS). Este prima metodologie descrisă, propusă de Cris Gane şi Trish Sarson.
- Yourdon Systems Method (YSM).
- Information Engineering (IE).
- Structured System Analysis and Design Methodology (SSADM).
- Méthode d'Etude et de Realization, Informatique pour les Systèmes d'Entreprise **MERISE** .
- Jackson System Development (JSD) .
- Information System Work and Analysis of Changes (ISAC).
- Effective Techical and Human Implementation of Computer-based Systems (ETHICS) .
- Soft System Methodology (SSM) .
- Multiview .
- Process Innovation.
- Rapid Application Development (RAD) .
- Metodologia Institutului Centrului de Informatică din România

Metodologiile cu abordare orientată obiect

Avantajele utilizării metodologiilor orientate obiect:

- **Datele şi prelucrările** nu mai sunt reprezentate distinct, ca în cazul abordării structurate, ci **încapsulat** în clase de obiecte.
- **Analiza** realizată pentru un sistem poate fi modificată în scurt timp pentru a fi **reutilizată** pentru analiza sistemelor din aceeaşi sferă de activitate.
- **Modelele** utilizate sunt **flexibile** şi uşor de întreţinut.
- Posibilitatea de a aborda domenii şi tipuri de probleme din ce în ce mai provocatoare.
- **Consistenţă crescută** între activităţile de analiză, proiectare şi programare.
- **Robusteţea sistemelor**.
- **Reutilizarea** rezultatelor analizei, proiectării şi implementării.
- Reprezentarea explicită a elementelor comune tuturor componentelor sistemului.
- **Consistenţă** crescută între toate modelele dezvoltate în timpul analizei orientate obiect, proiectării şi programării.

Metodologiile cu abordare orientată obiect

Exemple de metodologii cu abordare orientată obiect:

- Object Oriented Software Engineering (**OOSE**) concepută de Ivar Jacobson.
- Object Modeling Technique (**OMT**) elaborată de James Rumbaugh, Michael Blaha şi alţii. Metodologia a fost iniţial utilizată de General Electric and Development Center;
- Object Oriented Design (**OOD**) elaborată de Grady Booch, este o metodologie similară metodologiei OMT, dezvoltă aceeaşi idee – analiza şi proiectarea iterativă, insistând asupra părţii de proiectare ;
- Object Oriented Analysis (**OOA**) elaborată de Peter Coad şi Edward Yourdon;
- Object Oriented Structured Design (**OOSD**) elaborată de Wasserman;
- Object Oriented System Analysis (**OOSA**) este o metodologie de proiectare a sistemelor în timp real propusă de Sally Shlaer şi Steven Mellor. Autorii au continuat să îmbunătăţească această metodologie, publicând chiar şi o lucrare despre cum se pot utiliza notaţiile UML în cadrul metodologiei Shlaer/Mellor.;
- Responsibility Driven Design (**RDD**), aparţinând lui Wirfs – Brock, Wilkesson şi Wienner;
- Object Oriented Role Analysis, Synthesis and Structuring aparţinând lui Reens Kaugh;

Mare parte din deosebirile dintre OOD, OAD, OOSA, OMT şi OOSE au fost înlăturate în anul 1997 prin elaborarea unui standard cu privire la simboluri, notaţii, tipuri de diagrame, tipuri de modele etc., numit UML (Unified Modeling Language).

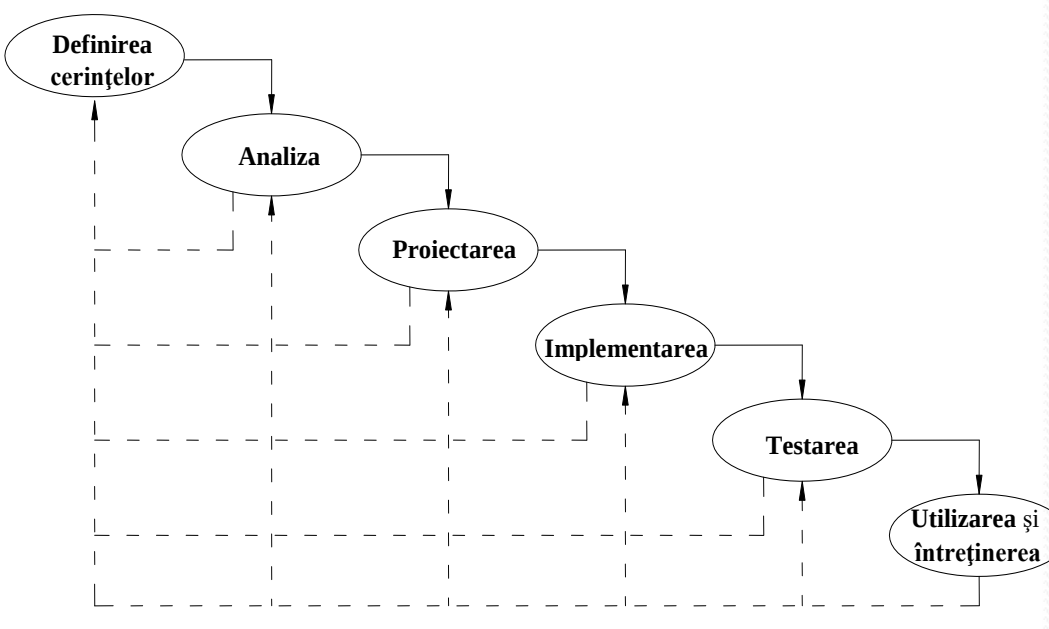
Clasificarea metodologiilor de realizare a sistemelor informatice

C. Clasificare după modelul ciclului de viață:

I. *Modelul de parcurgere în cascadă*
II. *Modelul de parcurgere în spirală*
III. *Modelul de parcurgere cu extensii*
IV. *Modelul de parcurgere evolutiv*
V. *Modele de parcurgere compozite (cicluri în V și în X)*

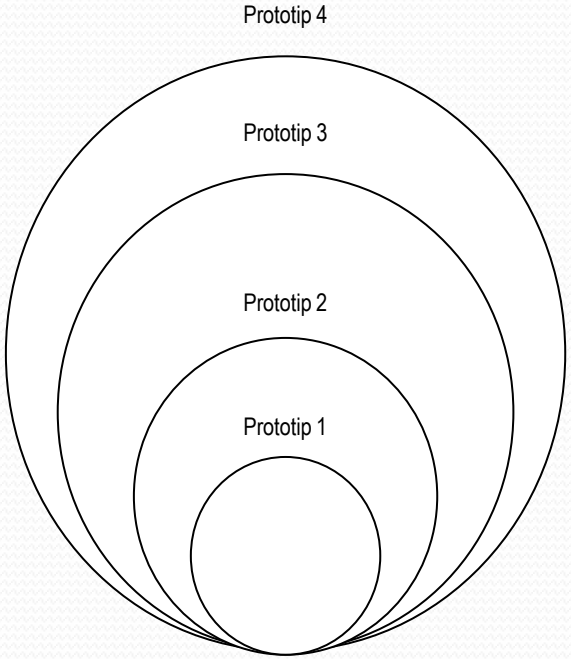
I. Modelul de parcurgere în cascadă

- ✓ Parcurgerea **secvențială** a etapelor, cu eventuale reveniri la etapa precedentă.
- ✓ Utilizat pentru sisteme informatice de **mică complexitate**.
- ✓ Modelul în cascadă sau liniar este **teoretic**, deoarece în realitate, parcurgerea etapelor este un proces iterativ, desfășurându-se adesea în paralel mai multe activităţi.



II. Modelul de parcurgere în spirală (modelul cu prototip)

- ✓ Presupune elaborarea completă, **rapidă** și la **costuri scăzute** a unei versiuni inițiale, simplificate, cu caracter de **prototip**, pe baza căreia se stabilesc noi specificații de definire a sistemului informatic și se desfășoară activitatea de realizare a unei noi versiuni de sistem informatic.
- ✓ Elaborarea noii versiuni presupune parcurgerea **integrală** sau **parțială** a etapelor, modificându-se numai anumite părți din prototip.

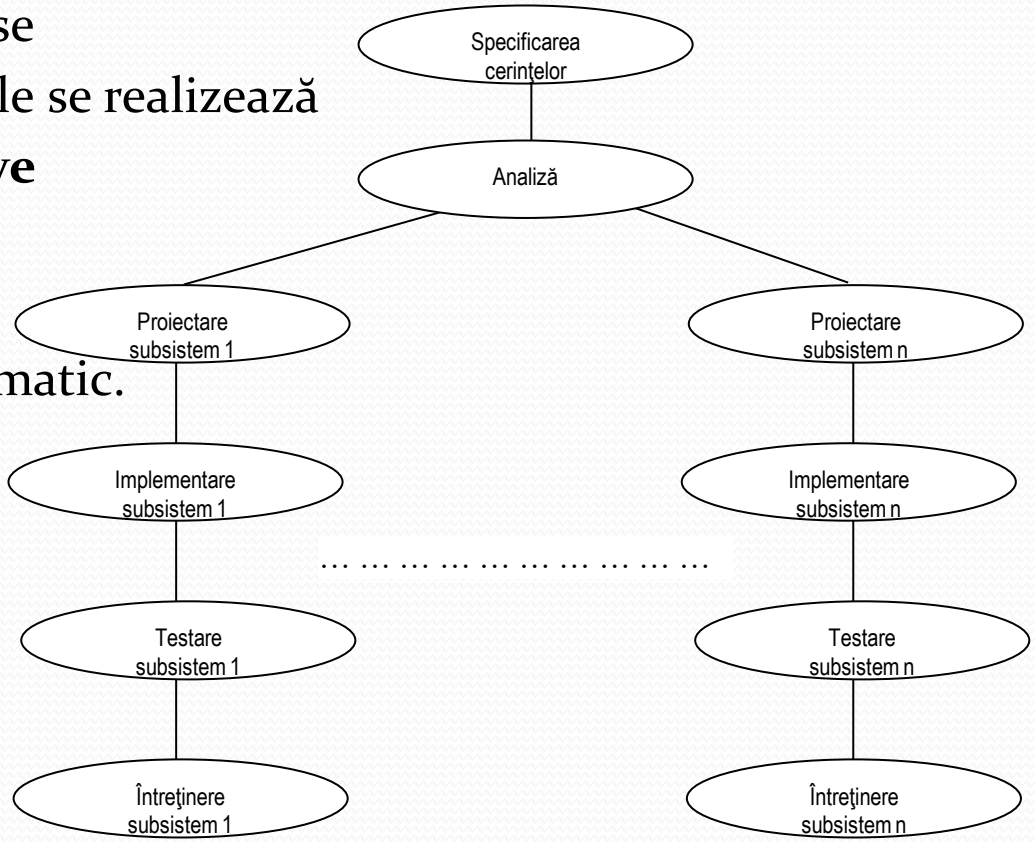


III. Modelul de parcurgere cu extensii (incremental)

- ✓Se utilizează atunci când sistemele informatice se pot realiza şi pune în funcţiune parţial **pe subsisteme, aplicaţii, module**.
- ✓Realizarea lor se poate face deci în **maniera extensibilă**,

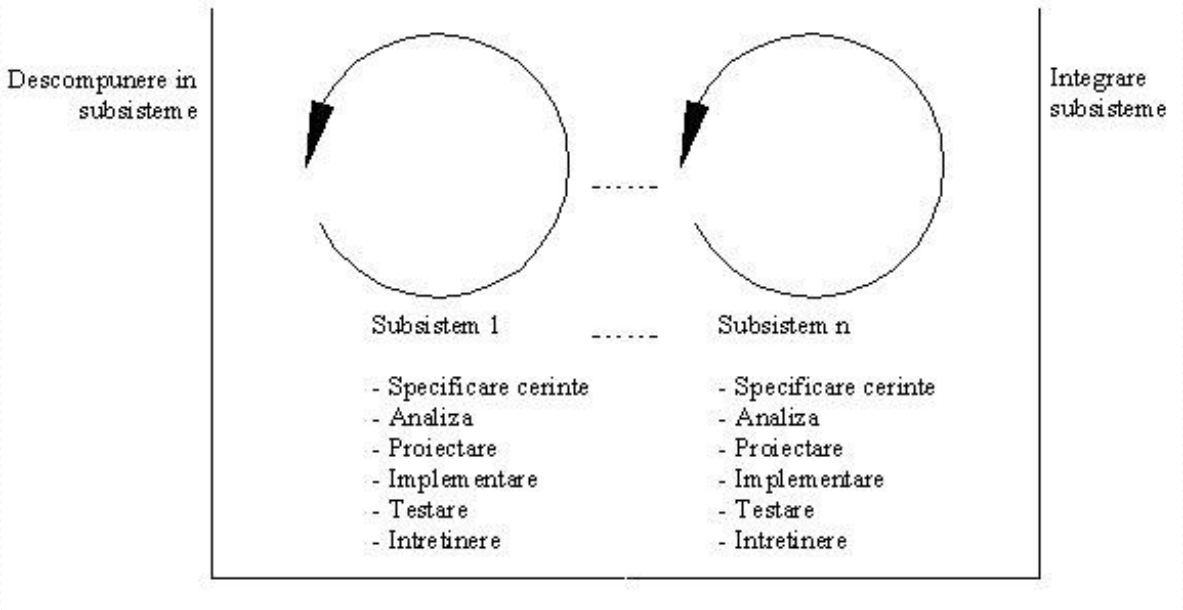
astfel încât la început se analizează şi se definesc cerinţele, iar apoi subsistemele se realizează şi se integrează prin **extensii succesive sau simultane.**

- ✓De obicei, extensiile se ramifică din etapa de proiectare a sistemului informatic.



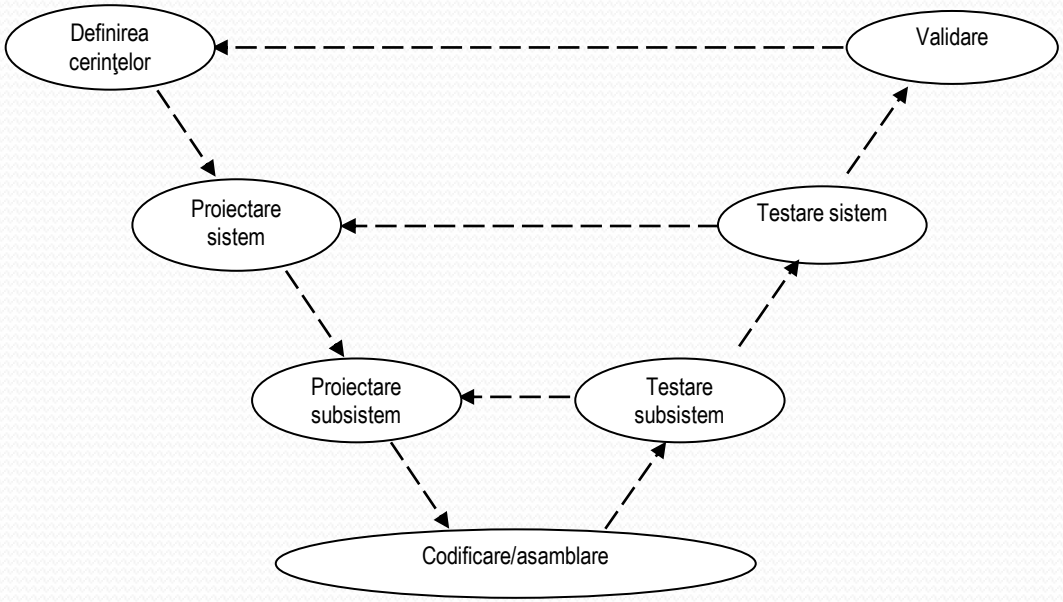
IV. Modelul de parcurgere evolutiv

- ✓ Se utilizează în cazul **sistemelor complexe**, care **se descompun în subsisteme**. Ele sunt realizate şi livrate în mod **iterativ** şi contribuie la sporirea treptată a performanţelor sistemului.
- ✓ Oricare dintre subsisteme trece prin **toate fazele** de dezvoltare a sistemelor: definirea cerinţelor, analiză, proiectare, implementare, testare, intreţinere, pentru ca în final acestea să fie **integrate**.



V. Modele de parcurgere compozite (ciclul în V)

- ✓ Este o **variantă a modelului cascadă**, în care se aplică **teste explicite** pentru creșterea controlului asupra modului în care se desfășoară etapele.
- ✓ Latura din **stânga** a literei "V" este parcursă **descendent** și conține etapele de dezvoltare propriu-zise, iar cea de-a doua latură, din **dreapta**, se parcurge **ascendent**, pe ea realizându-se verificările și validările elementelor create anterior.

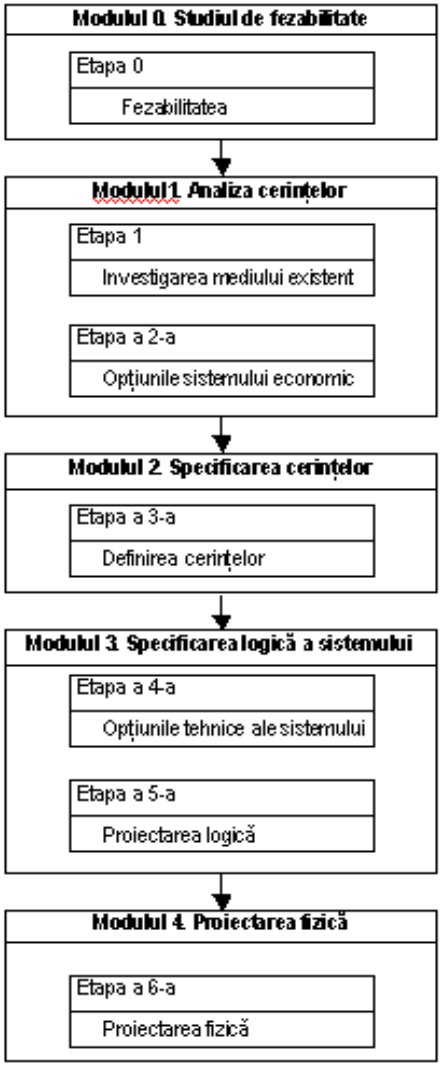


Etapele de realizare a proiectelor softwareconform metodologiei SSADM

SSADM include un set de tehnici, instrumente și formulare standard pentru descrierea **sistemului existent** sau a **sistemului proiectat** (noul sistem). Caracteristici generale:

- Este o metodologie orientată pe **structura datelor**.
- Pune în evidență două tipuri de modele: ***modelul logic*** și ***modelul fizic*** al sistemului, deci separă proiectarea logică de proiectarea fizică.
- Se bazează pe specificarea clară a cerințelor și a unor reguli detaliate pentru construirea (proiectarea) celor două modele. Face apel la reprezentarea **fluxurilor de date** și **prelucrărilor** cu ajutorul diagramelor.
- Conține **cinci module**: studiul de fezabilitate, analiza cerințelor, specificarea cerințelor, specificarea logică a sistemului și proiectarea fizică. Fiecare modul este divizat în **etape de lucru**. Fiecare etapă este împărțită într-un număr de **pași** care definesc intrările, ieșirile și sarcinile ce trebuie realizate.

Etapele de realizare a proiectelor softwareconform metodologiei SSADM



Metodologia MERISE

Din punctul de vedere al metodologiei MERISE, sistemele informatice au trei cicluri: ciclul de viață, ciclul de decizie, ciclul de abstractizare.

1. **Ciclul de viață** al sistemului informatic se bazează pe dualitatea obiect natural (sistemul informational) – obiect artificial (sistemul informatic). E împărțit în trei perioade:
   1. **concepția** sistemului concretizată în **specificațiile funcționale și tehnice**,
   2. **realizarea** sistemului concretizată în **specificațiile de detaliu** și
   3. **implementarea** sistemului.
2. **Ciclul de decizie** al sistemului informatic cuprinde ansamblul de **opțiuni** care trebuie să existe în timpul ciclului de viață și definește **interfața** între sistemul informatic (obiectul artificial) și sistemul informațional (obiect natural). Aici sunt definite punctele de decizie privind scopurile sistemului informatic și punctele de decizie privind introducerea sistemului informatic ca obiect natural.
3. **Ciclul de abstractizare** este util pentru a surprinde elementele semnificative în descrierea sistemului, ignorând detaliile. El privește în exclusivitate sistemul informatic ca obiect artificial, funcționarea lui fiind verificată prin **simularea unor părți** din el. Presupune utilizarea a trei niveluri de abstractizare:
   - *Nivelul conceptual*
   - *Nivelul organizațional pentru prelucrări și nivelul logic pentru date*
   - *Nivelul operațional pentru prelucrări și nivelul fizic pentru date*

Etapele de realizare a sistemelor informatice conform metodologiei MERISE

- **Elaborarea schemei directoare** presupune stabilirea concordanţei dintre **obiectivele strategice** ale organizaţiei şi **cerinţele informaţionale** ale conducerii. În acest sens se va face o formalizare globală a situaţiei existente şi se va folosi *metoda scenariilor* pentru împărţirea sistemului informaţional în *domenii*.
- **Studiul prealabil** se realizează pe un **subansamblu reprezentativ al domeniului** ce urmează a fi informatizat. Pe acest subansamblu se derulează ciclul de abstractizare. Uneori această etapă se include în elaborarea schemei directoare.
- **Studiul detaliat** se face plecând de la rezultatele studiului prealabil şi presupune obţinerea **specificaţiilor funcţionale generale detaliate ale noului sist**em. Aceste specificaţii sunt orientate către proiectanţii care realizează efectiv sistemul.
- **Studiul tehnic** presupune proiectarea logică şi tehnică a **fişierelor/bazei de date**, descrierea arhitecturii **prelucrării** datelor, proiectarea arhitecturii **produsului – program**.
- **Elaborarea programelor** constă în scrierea, testarea şi punerea la punct a programelor în funcţie de specificaţiile din etapa precedentă.
- **Introducerea sistemului** presupune pregătirea lansării în execuţie şi lansarea propriu-zisă a acestuia.
- **Menţinerea în funcţiune** a sistemului presupune asigurarea funcţionării sistemului la parametrii proiectaţi şi, eventual, mici dezvoltări pe parcurs

Etapele de realizare a proiectelor softwareconform OMT

- Modelarea sistemului este realizată prin prisma a trei modele diferite:
  - **Modelul obiectelor** descrie din punct de vedere **static obiectele**, relațiile dintre obiecte, atributele și operațiile fiecărei clase de obiecte. Este de fapt un **model al datelor**, privit prin prisma abordării orientate obiect, arătând **CE se analizează**.
  - **Modelul dinamic** pune în evidență **stările** datelor, precum și fluxul **evenimentelor** care conduc trecerea dintr-o stare în alta.
  - **Modelul funcțional** descrie modul de obținere a ieșirilor informaționale din intrări sau alte informații intermediare.

- Etapele de realizare a sistemului informatic conform acestei metodologii sunt:
  1. analiza,
  2. proiectarea sistemului,
  3. proiectarea obiectelor
  4. implementarea.

Etapele de realizare a proiectelor softwareconform OMT

Etapa	Activități specifice
Analiza sistemului	○ *Definirea problemei* ○ *Inițierea realizării* ***modelului obiectelor*** ○ *Inițierea realizării* ***modelului dinamic*** • identificarea intrărilor și ieșirilor (privite ca parametri ai evenimentelor din sistem) și reprezentarea diagramei de flux a datelor (**DFD**) • descrierea proceselor elementare • identificarea constrângerilor • identificarea modalităților de **optimizare**
Proiectarea sistemului	○ *Descompunerea în* ***subsisteme*** ○ *Identificarea subsistemelor concurente* ○ *Stabilirea* ***necesarului de resurse*** *și a* ***modului de implementare*** *hardware și software pentru fiecare subsistem* ○ *Alegerea* ***modului de organizare*** *a datelor și a* ***tipurilor de acces*** *la date* ○ *Stabilirea* ***controlului*** *intern și extern pe fluxul evenimentelor sau pe fluxul prelucrărilor.* ○ *Stabilirea condițiilor limită, care se referă la inițializări, terminarea normală sau anormală, priorități*

Etapele de realizare a proiectelor softwareconform OMT

Etapa	Activități specifice
Proiectarea obiectelor	Rafinează modelele obţinute în faza de analiză, prin adăugarea detaliilor de implementare. ○ *Identificarea operatiilor* ○ *Proiectarea algoritmilor;* ○ *Rafinarea, restructurarea modelului datelor;* ○ *Implementarea asocierilor;* ○ *Gruparea datelor și asocierilor în module.*
Implementarea	Se realizează transpunerea într-un limbaj de programare şi transferul sistemului.

Exemplu de proces iterativ de dezvoltare a proiectelor softwareutilizând UML (Unified Modeling Language)

Definirea problemei
Se identifică caracteristicile principale ale unității economice studiate și modul de funcționare a activității de implementat.
Structurarea soluției
Se determină și se detaliază cerințele beneficiarului. Include subetapele: **Stabilirea actorilor; Stabilirea cazurilor de utilizare; Stabilirea relațiilor dintre cazurile de utilizare; Construirea diagramelor cazurilor de utilizare**
Analiza sistemului
Sunt analizate specificațiile și cazurile de utilizare, identificându-se cele mai importante concepte cu care lucrează sistemul, împreună cu relațiile dintre acestea. Se construiesc: **diagrama de clase, diagrame de obiecte, de stare, de activitate, de secvență, de comunicare.**
Proiectarea sistemului
Conține două subetape: proiectarea arhitecturii si proiectarea de detaliu. Implică proiectarea arhitecturii sistemului, a bazei de date, a interfeței, eventualilor algoritmi modelați în cadrul sistemului. Se construitesc **diagramele de componete și de desfășurare.**
Implementarea sistemului
Implică programarea efectivă a claselor identificate, prin scrierea codului sursă și realizarea videoformatelor și a situațiilor de ieșire.

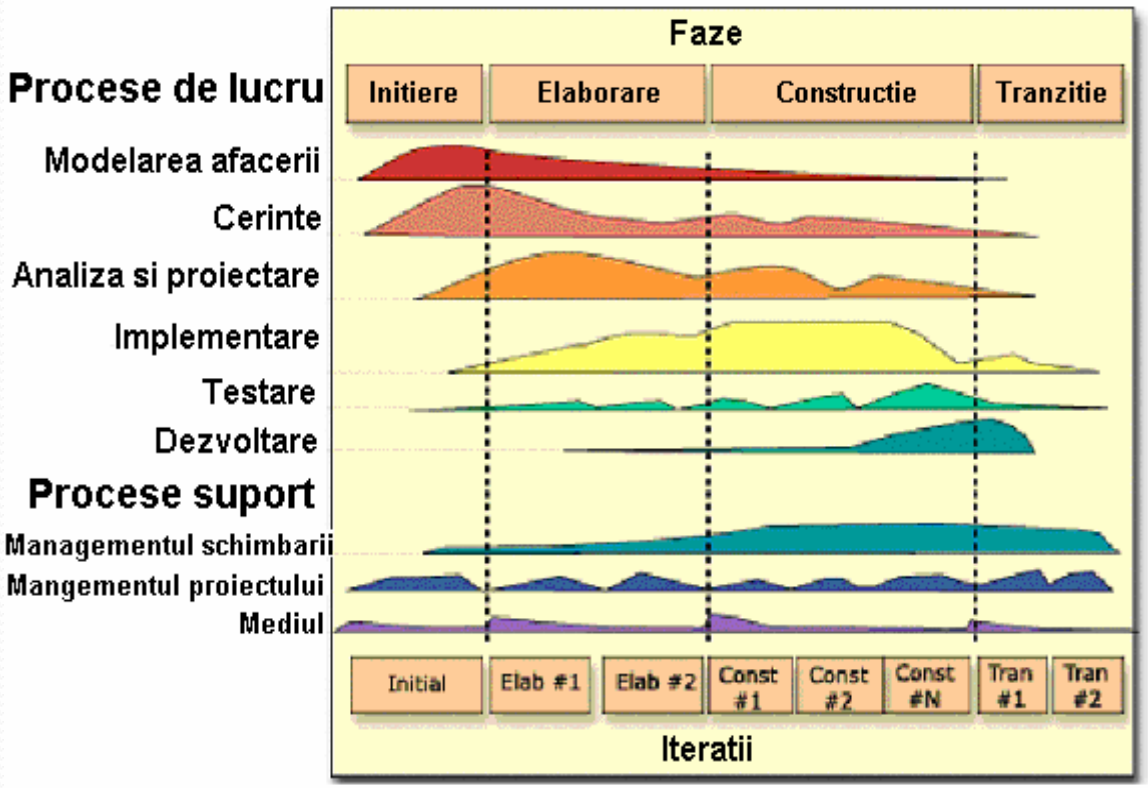
Metodologia unificata de realizare a proiectelor software(RUP)

- **Rational Unified Process** (**RUP**) este un proces general pentru dezvoltarea orientată obiect de produse informatice.
- Este un ghid care arată cum se poate utiliza practic UML (Unified Modeling Language) pentru a dezvolta un sistem informatic.
- RUP a fost realizat de **compania Rational** și dezvoltat acum de către **IMB**.
- Nucleul îl reprezintă metodologia propusă de Jacobson, Booch și Rumbaugh (**Unified Process**) care este mai mult decât un simplu proces, este un cadru general care a permis dezvoltarea de metodologii specializate (pe diverse tipuri de sisteme informatice în funcție de aria de aplicare, diverse tipuri de organizații, niveluri de competență sau dimensiuni diferite ale proiectelor)

Fazele RUP

Axa orizontală: reprezintă **timpul** şi evidenţiază aspectele dinamice ale procesului. Pe axa orizontală procesul se exprimă în termeni de: cicluri, faze, iteraţii şi jaloane.

Axa verticală: reprezintă **aspectele statice ale procesului** şi se exprimă în termeni de: activităţi, produse, executanţi şi fluxuri.



1. Faza de explorare inițială

Caracteristici:

- Poate avea o întindere însemnată în cazul proiectelor noi.
- Are rolul de a crea o **viziune de ansamblu** asupra proiectului, de a determina **necesitatea** proiectului şi de a identifica **riscurile**.
- În cazul proiectelor care au ca scop îmbunătăţirea unui sistem existent, faza de explorare iniţială este mai scurtă şi are rolul de a determina utilitatea proiectului.
- Documentul cheie produs în cadrul acestei etape este ***Viziunea***. Acest document trebuie să cuprindă o descriere de nivel înalt a sistemului şi a funcţionalităţii critice (este un document scurt, de obicei două, trei paragrafe).

Rezultatele fazei:

- **Documentul viziune**: o descriere generală a principalelor cerinţe ale proiectului, problemele-cheie şi principalele constrângeri.
- Un model iniţial al **cazurilor de utilizare** (10%-20% complet).
- Un **glosar** iniţial al proiectului (opţional exprimat ca model al domeniului).
- **Descriere inițială a procesului de afaceri**, care include contextul, criteriile de succes şi o previziune financiară.
- Evaluare iniţială a **riscului**.
- Un **plan al proiectului**, care să precizeze fazele şi iteraţiile.
- Un **model al afacerii**, dacă este necesar.
- Unul sau mai multe **prototipuri**.

2. Faza de elaborare

Caracteristici:

- Are ca scop conturarea **arhitecturii de bază** care va constitui documentul de pornire pentru proiectarea și implementarea din faza de construcție.
- Arhitectura este rezultatul analizei cerințelor principale (care au un efect major asupra arhitecturii) și a riscurilor asociate.
- Stabilitatea arhitecturii este evaluată prin intermediul unuia sau mai multor **prototipuri**.
- În timpul fazei de elaborare este construită o **arhitectură „executabilă"**, fapt care reduce riscurile legate de **cerințele nefuncționale** (performanță, robustețe, scalabilitate).

Rezultatele fazei:

- **Un model al cazurilor de utilizare** (cel puțin 80% complet) (sunt identificate toate cazurile de utilizare și toți actorii, iar majoritatea cazurilor de utilizare sunt dezvoltate).
- **Cerințe suplimentare** care să surprindă cerințele nefuncționale și orice altă cerință care nu are legătură cu un caz de utilizare specific.
- O **descriere a arhitecturii sistemului**.
- Un **prototip executabil** al arhitecturii.
- **Listă revizuită a riscurilor** și o descriere revizuită a procesului de afacere.
- Un **plan de dezvoltare al întregului proiect**, care să cuprindă planul proiectului cu iterațiile și criteriile de evaluare pentru fiecare iterație.
- Un proces avansat de **dezvoltare** care să specifice versiunea utilizată.
- Un **manual de utilizare** preliminar (opțional).

3. Faza de construcție

Caracteristici:

- Are ca scop dezvoltarea efectivă a sistemului.
- Spre deosebire de primele două faze în care efortul era dedicat producerii de „idei“, în această fază accentul este pus pe **managementul resurselor** în scopul **implementării** sistemului.
- Fiecare iterație a fazei de construcție conține trei activități de bază:
  - managementul resurselor și controlul procesului,
  - dezvoltarea și testarea componentelor și
  - evaluarea la sfârșitul iterației.
- Principalele documente rezultate sunt: componentele cu documentația aferentă, materialele de instruire, planul de instalare și planul pentru faza de tranziție.

Rezultatele fazei:

- **Produsul software integrat** pe o platformă corespunzătoare;
- Manualele de utilizare;
- Descriere a versiunii actuale.

4. Faza de tranziție

Caracteristici:

- Cuprinde **testarea finală**, pregătirea lansării și efectuarea de modificări minore dictate de reacțiile beneficiarilor.
- Accentul în această fază este pus pe **configurarea și reglarea** sistemului și pe detaliile referitoare la instalare.
- Activităţile principale sunt:
  - finalizarea documentației,
  - testarea produsului la client,
  - modificări minore dictate de client și
  - lansarea sau instalarea produsului final.

Rezultatele fazei:

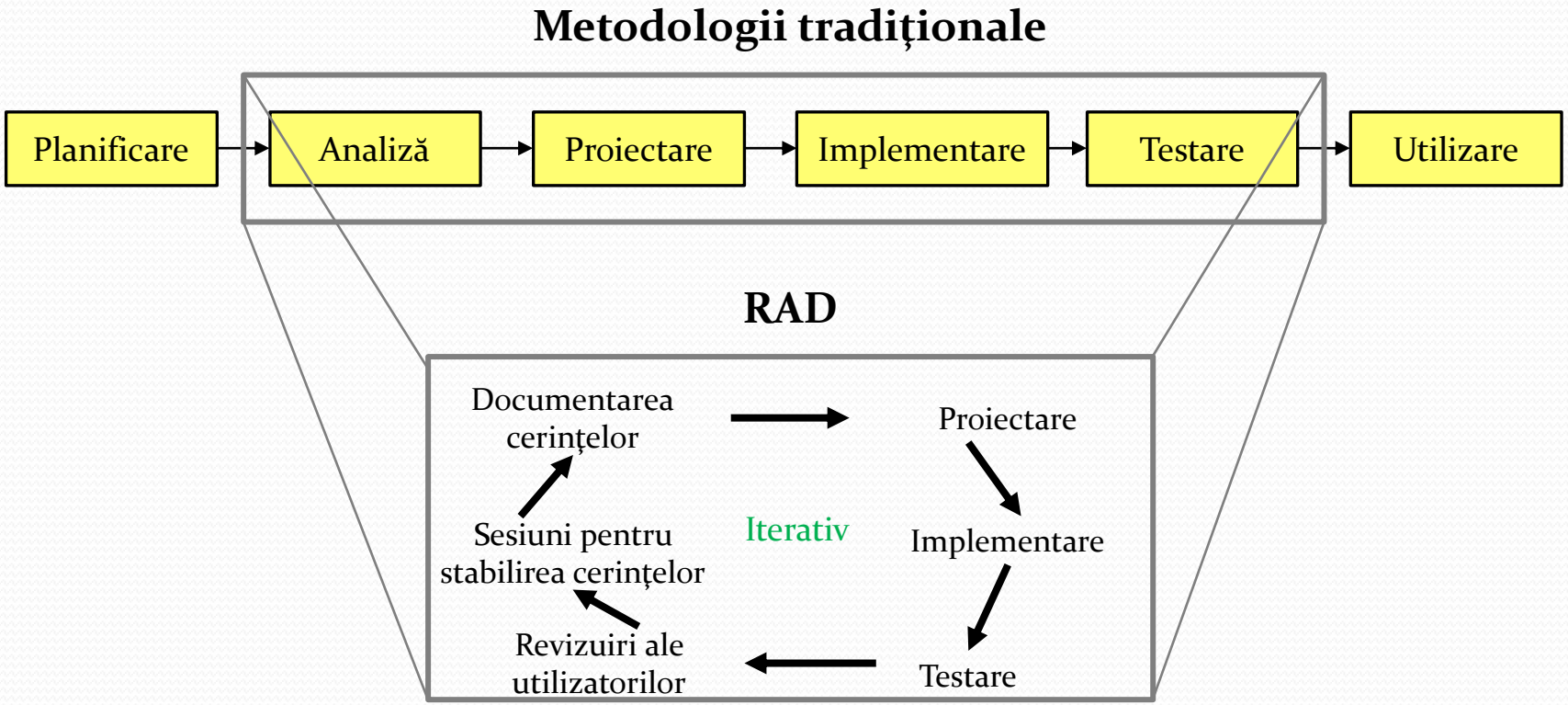
- Planul de instalare;
- Notele finale asupra sistemului ;
- Documentația.

Metodologii bazate pe dezvoltarea rapidă RAD (Rapid Application Development)

- Se referă la o serie de principii care urmăresc ajustarea etapelor de realizare a sistemelor informatice, astfel încât o parte a sistemului să fie dezvoltată și să ajungă la utilizatori rapid.
- Astfel, utilizatorii pot înțelege mai bine sistemul și pot sugera revizuiri care aduc sistemul mai aproape de cerințele acestuia.
- Se recomandă ca analiștii să folosească tehnici speciale și instrumente informatice pentru a accelera etapele de analiză, proiectare și implementare, cum ar fi:
  - instrumente CASE
  - sesiuni comune pentru stabilirea cerințelor Join Requirement Planning (JRP)
  - limbaje de programare de generația a patra
  - limbaje de programare vizuale
  - generatoare de cod
  - reutilizarea componentelor software
- Astăzi, aproape toate mediile de dezvoltare integrate au facilități specifice RAD.

Metodologii bazate pe dezvoltarea rapidă RAD (Rapid Application Development)

- RAD comprimă pașii metodologiilor tradiționale într-un proces **iterativ**.
- Se bazează pe **prototipizare** și pe **revizuiri ale utilizatorilor** înainte de a trece la parcurgerea unei noi iterații.



Metodologii bazate pe dezvoltarea agilă

- Aceste metodologii sunt **orientate pe programare** și au puține reguli și practici.
- Toate metodologiile de dezvoltare agile sunt bazate pe manifestul agil și pe un set de douăsprezece principii:

1. Este prioritară **satisfacția clientului** prin livrarea rapidă și continuă de software calitativ.
2. **Schimbarea cerințelor** este binevenită chiar și într-o fază avansată a dezvoltării. Procesele agile valorifică schimbarea în avantajul competitiv al clientului.
3. **Livrarea** de software funcțional se face **frecvent**, de preferință la intervale de timp cât mai mici, de la câteva săptămâni la câteva luni.
4. Oamenii de afaceri și dezvoltatorii trebuie **să colaboreze zilnic** pe parcursul proiectului.
5. Proiecte se construiesc în jurul **oamenilor motivați**. Oferindu-le mediul propice și suportul necesar este foarte probabil că obiectivele vor fi atinse.

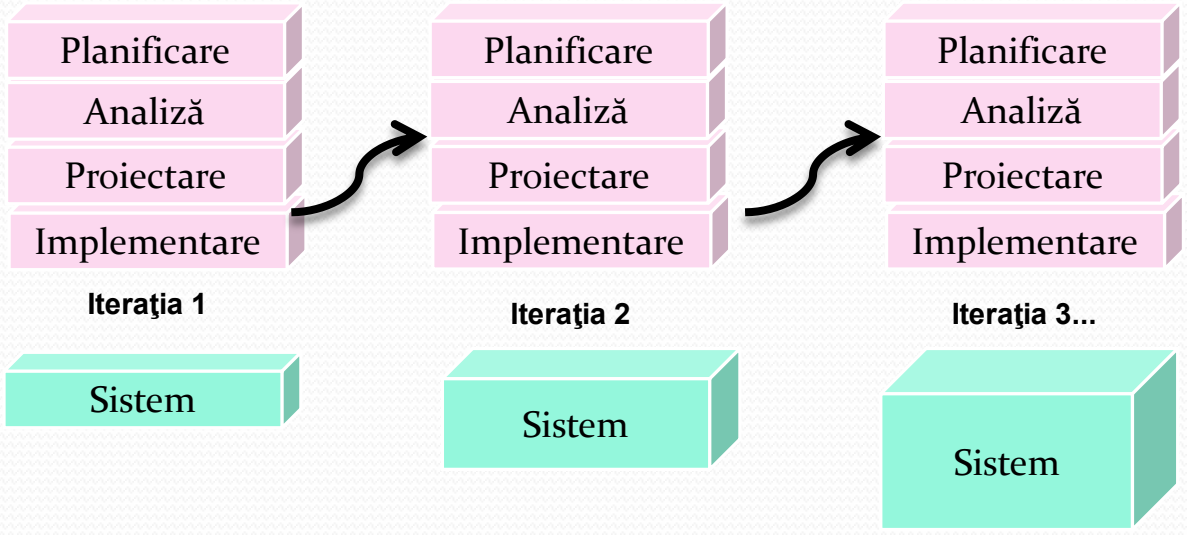
Metodologii bazate pe dezvoltarea agilă

Principii ale manifestului agil- continuare:

6. Cea mai eficientă metodă de a transmite informaţii înspre şi în interiorul echipei de dezvoltare este **comunicarea faţă în faţă**.
7. **Software-ul funcţional** este principala măsură a progresului.
8. Procesele agile promovează **dezvoltarea durabilă**. Beneficiarii, dezvoltatorii şi utilizatorii trebuie să poată menţine un ritm de lucru constant pe termen lung.
9. Atenţia continuă pentru **excelenţă tehnică** şi **design bun** îmbunătăţeşte agilitatea.
10. **Simplitatea** -arta de a maximiza cantitatea de muncă nerealizată- este esenţială.
11. Cele mai bune arhitecturi, cerinţe şi design sunt create de **echipe care se auto-organizează.**
12. La intervale regulate, **echipa reflectă** la cum să devină mai eficientă, apoi îşi adaptează şi **ajustează comportamentul** în consecinţă.

Metodologii bazate pe dezvoltarea agilă

- Pe baza acestor principii, metodologiile agile se concentrează pe optimizarea procesului de dezvoltare a sistemelor prin eliminarea unei părți semnificative din modelare și documentare .
- Este susținută realizarea simplă, iterativă a sistemelor. Practic toate metodologiile agile sunt folosite în combinație cu tehnologiile orientate-obiect.
- Toate metodologiile bazate pe dezvoltarea agilă urmează un ciclu de dezvoltare simplu prin parcurgerea etapelor tradiționale ale procesului de dezvoltare a sistemelor.
- Două dintre cele mai populare exemple de metodologii de dezvoltare agile sunt Extreme Programming (XP) și Scrum.

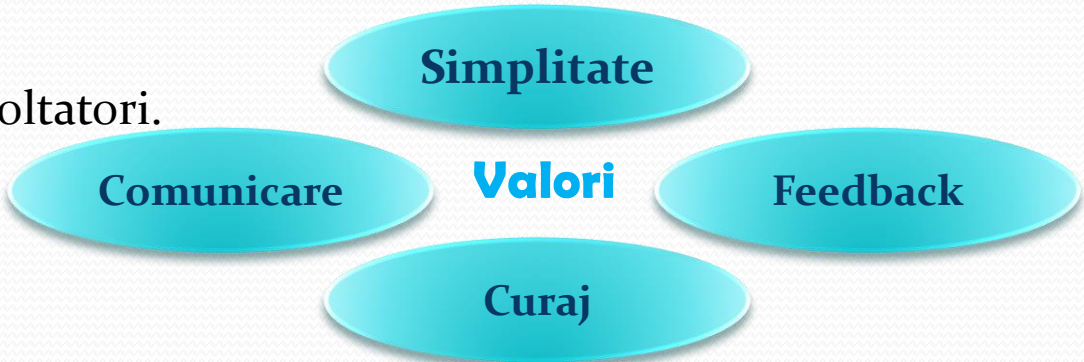


Metodologii bazate pe dezvoltarea agilă

Avantaje	Dezavantaje
Abordare realistă în realizarea sistemelor informatice.	Nu sunt potrivite pentru a gestiona dependențe complexe.
Promovează lucrul în echipă și învățarea.	Risc crescut de sustenabilitate, mentenabilitate și extensibilitate.
Funcționalitățile pot fi implementate rapid și verificate.	Fără o documentație suficientă, nici sistemul și nici procesul de dezvoltare al sistemelor nu pot fi auditate.
Model potrivit pentru mediile care se schimbă în mod continuu.	Depinde foarte mult de interacțiunea cu beneficiarul.
Reguli minime, documentație ușor de realizat.	Transferul de informații către membrii noi ai echipei este îngreunat de lipsa documentației.
Ușor de gestionat.	Lipsa regulilor poate duce la apariția unui mediu de lucru haotic.
Oferă flexibilitate.	Dependența de membrii echipei care cunosc cerințele sistemului.

Extreme Programming - XP

- Pune accentul pe codificare (standarde, principii) - utilizează un set comun de nume, descrieri și practici de codificare.
- Susține ca programatorii să lucreze câte doi ("pair programming"), iar programatorii au o responsabilitate comună pentru fiecare componentă software elaborată.
- Numeroase sesiuni de discuții pe parcursul dezvoltării.
- Feedback rapid al utilizatorilor finali în mod continuu.
- Dezvoltatorii trebuie să aibă o mentalitate orientată către calitate.
- Se bazează foarte mult pe refactoring, care este un mod disciplinat de restructurare a codului pentru a-l păstra simplu.
- Sistemul este dezvoltat într-un mod evolutiv și incremental.
- Fiecare iterație (1-4 săptămâni) are un rezultat funcțional.
- Suport redus pentru modelare.
- Relație strânsă între clienți și dezvoltatori.
- Lipsa documentației de realizare.



Extreme Programming - XP

Când se recomandă:

- Pentru **proiectele mici** cu **echipe** extrem de **motivate, unite, stabile**, și cu experiență, XP ar trebui să funcționeze foarte bine.
- XP este recomandat numai pentru **grupuri mici** de dezvoltatori, nu mai mult de zece persoane.
- Pentru cicluri scurte de dezvoltare și atunci când sunt facilitate dicuțiile frecvente cu utilizatorii finali.

Când NU se recomandă:

- În cazul în care proiectul nu este mic sau **echipele nu sunt unite**, succesul unui efort de dezvoltare XP este îndoielnic.
- Există dubii asupra beneficiilor introducerii unor **contractori externi** în cadrul unei echipe existente, când se lucrează conform XP.
- XP necesită un grad ridicat de **disciplină**; în caz contrar proiectele vor deveni nefocalizate și haotice.
- Nu se recomandă pentru **aplicații mari**. Din cauza lipsei de analiză și documentației de proiectare, există doar documentație cod asociat cu XP, deci mentenanta sistemelor de mari dimensiuni construite cu XP poate fi imposibilă.

SCRUM

- Numele metodologiei este preluat din jocul de rugby, și desemnează o grămadă ordonată folosită pentru a reporni un joc
- Creatorii metodei Scrum cred că indiferent cât bine este realizată planificarea dezvoltării unui sistem, de îndată ce software-ul începe să fie dezvoltat va izbucni haosul și planurile nu vor mai avea utilitate.

Principii de organizare

- Echipele sunt auto-organizate și auto-dirijate.
- Spre deosebire de alte abordări, echipele Scrum nu au un lider de echipă desemnat.
- Echipele se organizeze într-o manieră simbiotică și își stabileasc propriile obiective pentru fiecare **sprint** (iterație).

SCRUM

Principii de funcţionare

- Odată ce un **sprint** a început, echipele Scrum nu mai iau în considerare **nici o cerință suplimentară.**
- Orice cerințe noi care sunt descoperite sunt plasate într-o listă de cerințe care urmează să fie abordate.
- La începutul fiecărui zile de lucru, are loc o reuniune unde toți membrii echipei stau într-un cerc și raportează realizările zilei precedente, stabilesc ce au de gând să facă astăzi și descriu tot ceea ce a blocat progresul în ziua precedentă.
- Pentru a asigura un progres continuu, orice blocaj identificat este abordat în următoarea oră.
- La sfârșitul fiecărui sprint, echipa prezintă software-ul clientului.
- Pe baza rezultatelor iterației încheiate, este început un nou plan pentru următoarea iterație.