

Ingineria Programării

Cursul 2 – 24,25 Februarie
adiftene@info.uaic.ro

Cuprins

- ▶ Din Cursul 1...
- ▶ V-Model
- ▶ Extreme Programming
- ▶ Agile
- ▶ Scrum
- ▶ Lean
- ▶ MDD, AMDD
- ▶ TDD
- ▶ Ingineria cerințelor

Din Cursul 1 – 1

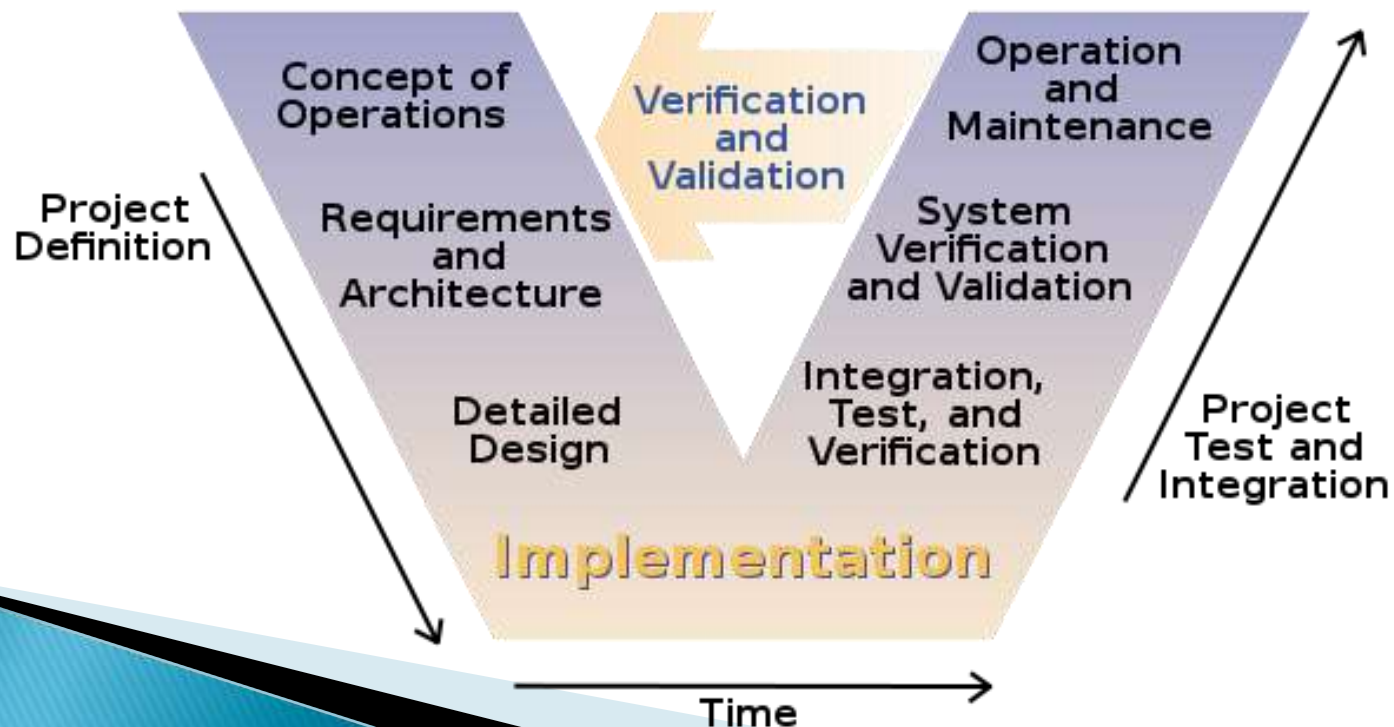
- ▶ Ingineria programării (Software engineering)
 - Se referă la metodologiile folosite în rezolvarea proiectelor mari care sunt rezolvate de echipe de oameni
 - Folosirea principiilor ingineresti în *analizarea, dezvoltarea, punerea în funcțiune, testarea, întreținerea, retragerea* produselor software
 - Tot aici mai pot fi prinse: *gestionarea resurselor, coordonarea echipelor, planificare, buget*
- ▶ **Scop:** obținerea de programe sigure și care funcționează eficient pe mașini de calcul concrete

Din Cursul 1 – Etapele dezvoltării programelor

- ▶ Analiza cerințelor (Requirements analysis)
- ▶ Proiectarea architecturală (Architectural design)
- ▶ Proiectarea detaliată (Detailed design)
- ▶ Scrierea codului (Implementation)
- ▶ Integrarea componentelor (Integration)
- ▶ Validare (Validation)
- ▶ Verificare (Verification)
- ▶ Întreținere (Maintenance)

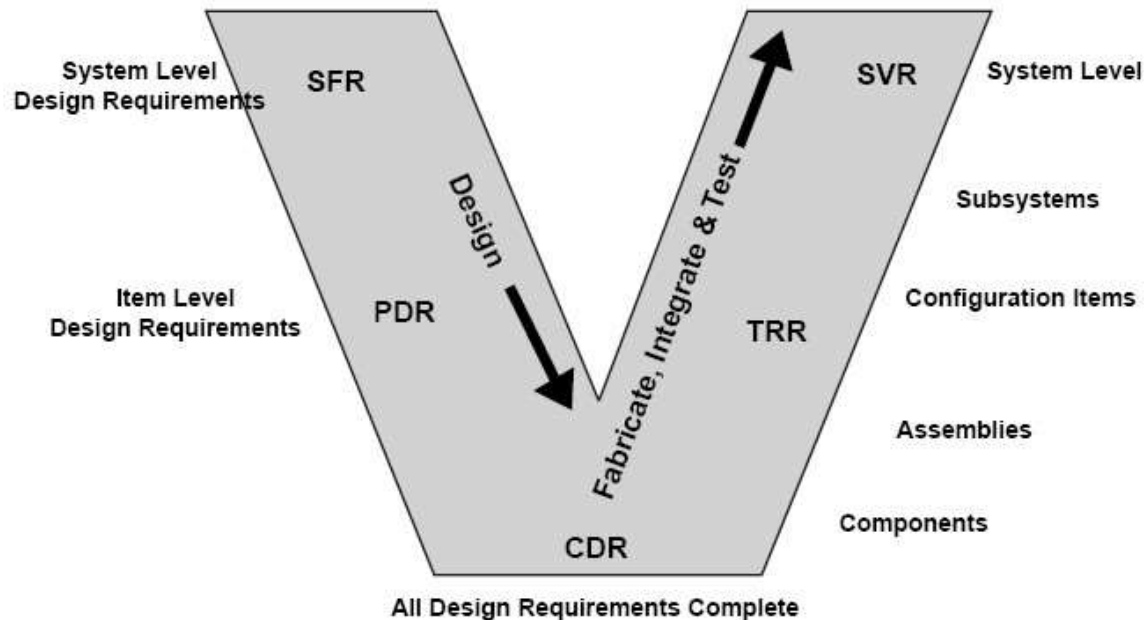
V-Model

- ▶ A fost utilizat în Germania și SUA în anii '80
- ▶ **Partea stângă** – analiza cerințelor și crearea specificațiilor sistemului
- ▶ **Partea dreaptă** – integrarea părților și validarea lor
- ▶ V de la Verificare și Validare



V-Model – Objective

- ▶ Minimizarea riscurilor
- ▶ Îmbunătățirea și garantarea calității
- ▶ Reducerea costurilor
- ▶ Îmbunătățirea comunicării



SFR = System Functional Review
PDR = Preliminary Design Review
CDR = Critical Design Review

TRR = Test Readiness Review
SVR = System Verification Review

V-Model + -

- ▶ +: utilizatorii V-model participă la dezvoltare și la întreținere
- ▶ +: există un grup ce controlează modificările din specificații. Acesta se întâlnește odată pe an și hotărăște ce modificări sunt acceptate
- ▶ +: modelul asigură la fiecare etapă asistență și definește explicit ce avem de făcut
- ▶ -: nu se asigură întreținerea
- ▶ -: e folosit la proiecte de dimensiuni relativ mici

Extreme Programming

► Why "Extreme"?

- "Extreme" means these practices get "turned up" to a much higher "volume" than on traditional projects.

► What really matters?

- Listening, Testing, Coding, Designing



XP – Caracteristici

- ▶ XP este un model modern, ușor (lightweight), de dezvoltare, inspirat din RUP
- ▶ Dezvoltarea programelor nu înseamnă ierarhii, responsabilități și termene limită, ci înseamnă colaborarea oamenilor din care este formată echipa
- ▶ Membrii echipei sunt încurajați să-și afirme personalitatea, să ofere și să primească cunoaștere și să devină programatori străluciți
- ▶ XP consideră că dezvoltarea de programe înseamnă în primul rând scrierea de programe (fișierele PowerPoint nu se pot compila)

Idei majore in XP

- ▶ Proiectul este în mintea tuturor programatorilor din echipa, nu în documentații, modele sau rapoarte
- ▶ La orice moment, un reprezentant al clientului este disponibil pentru clarificarea cerințelor
- ▶ Codul se scrie cât mai simplu. **Se scrie cod de test întâi**
- ▶ Dacă apare necesitatea re-scrierii sau aruncării de cod, aceasta se face fără milă
- ▶ Modificările aduse codului sunt integrate continuu (de câteva ori pe zi)
- ▶ **Se programează în echipă (programare în perechi). Echipele se schimbă la sfârșitul unei iterații (1–2 săptămâni)**
- ▶ **Se lucrează 40 de ore pe săptămână, fără lucru suplimentar**

XP Rules

Planning

- ❖ User stories are written.
- ❖ Release planning creates the schedule.
- ❖ Make frequent small releases.
- ❖ The Project Velocity is measured.
- ❖ The project is divided into iterations.
- ❖ Iteration planning starts each iteration.
- ❖ Move people around.
- ❖ A stand-up meeting starts each day.
- ❖ Fix XP when it breaks.

Designing

- ❖ Simplicity.
- ❖ Choose a system metaphor.
- ❖ Use CRC cards for design sessions.
- ❖ Create spike solutions to reduce risk.
- ❖ No functionality is added early.
- ❖ Refactor whenever and wherever possible.

Coding

- ❖ The customer is always available.
- ❖ Code must be written to agreed standards.
- ❖ Code the unit test first.
- ❖ All production code is pair programmed.
- ❖ Only one pair integrates code at a time.
- ❖ Integrate often.
- ❖ Use collective code ownership.
- ❖ Leave optimization till last.
- ❖ No overtime.

Testing

- ❖ All code must have unit tests.
- ❖ All code must pass all unit tests before it can be released.
- ❖ When a bug is found tests are created.
- ❖ Acceptance tests are run often and the score is published.

Agile

- ▶ Satisfacerea rapidă a clientului prin oferirea continuă de software util (săptămânal dacă e posibil)
- ▶ Progresul se măsoară în funcție de partea funcțională a proiectului
- ▶ **Chiar și modificările târzii în cerințe sunt binevenite**
- ▶ **O cooperare foarte apropiată între client și programatori**
- ▶ Discuțiile face-to-face constituie cea mai bună formă de comunicare
- ▶ **Adaptare continuă la modificările care apar**
- ▶ Dezvoltarea unui spirit de evidențiere și rezolvare a problemelor, nu de ascundere sau 'neobservare' a lor

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck

Mike Beedle

Arie van Bennekum

Alistair Cockburn

Ward Cunningham

Martin Fowler

James Grenning

Jim Highsmith

Andrew Hunt

Ron Jeffries

Jon Kern

Brian Marick

Robert C. Martin

Steve Mellor

Ken Schwaber

Jeff Sutherland

Dave Thomas

Agile +-

▶ -:

- Imposibilitatea realizării documentației necesare
- Se lucrează doar cu dezvoltatori “senior-level”
- Insuficientă structurare a modelării software
- Poate duce la negocieri de contract dificile

▶ +:

- **Companiile care au adoptat metoda de lucru Toyota și-au îmbunătățit cu 83% productivitatea, cu 93% timpul de producție, cu 91% calitatea produselor și au redus la jumătate overtime-ul – după cum arată un studiu oficial U.S., realizat în urmă cu câțiva ani pe companii din industria auto**



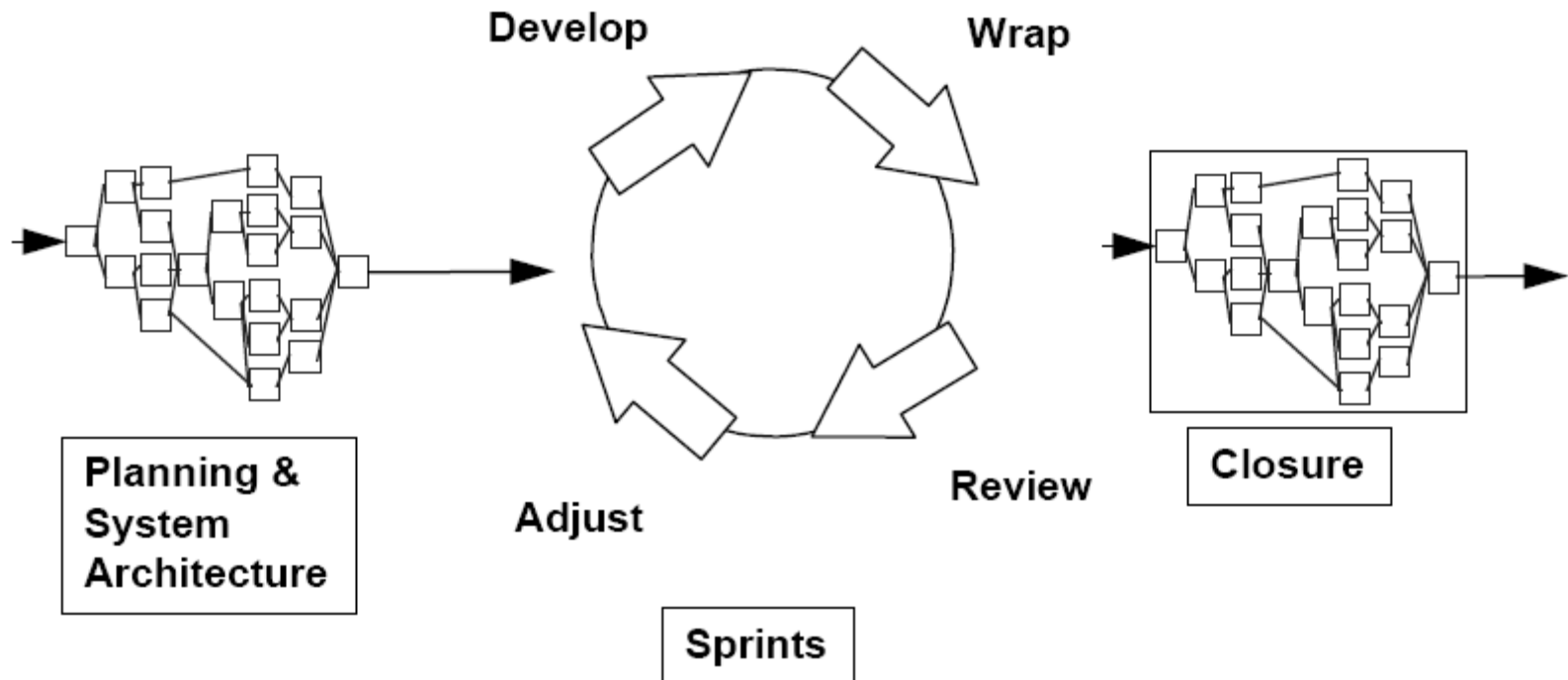
Scrum (“grămadă”)



- ▶ Clientul devine parte a echipei de dezvoltare
- ▶ Frecvente distribuiri intermediare a părții software, cu verificări și validări imediate
- ▶ Discuții zilnice:
 - Ce ai făcut ieri? (realizări)
 - Ce ai de gând să faci până mâine? (de realizat)
 - Care sunt problemele care te-ar putea încurca? (probleme/riscuri)
- ▶ Transparență în planificare și dezvoltare
- ▶ Întâlniri frecvente pentru a monitoriza progresul
- ▶ Nu sunt probleme ținute sub covor
- ▶ Eficiența muncii: “să lucrezi mai multe ore” nu înseamnă neapărat “obținerea mai multor rezultate”

Metodologia Scrum

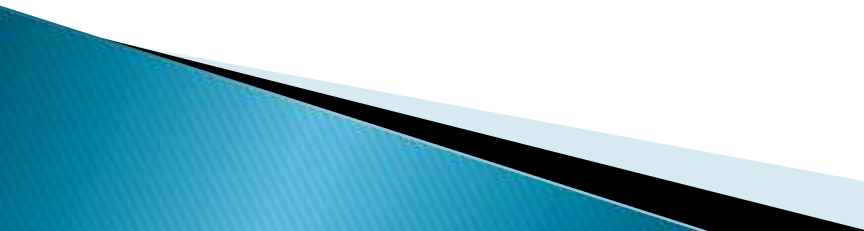
SCRUM Methodology



Lean (“sprijin”)

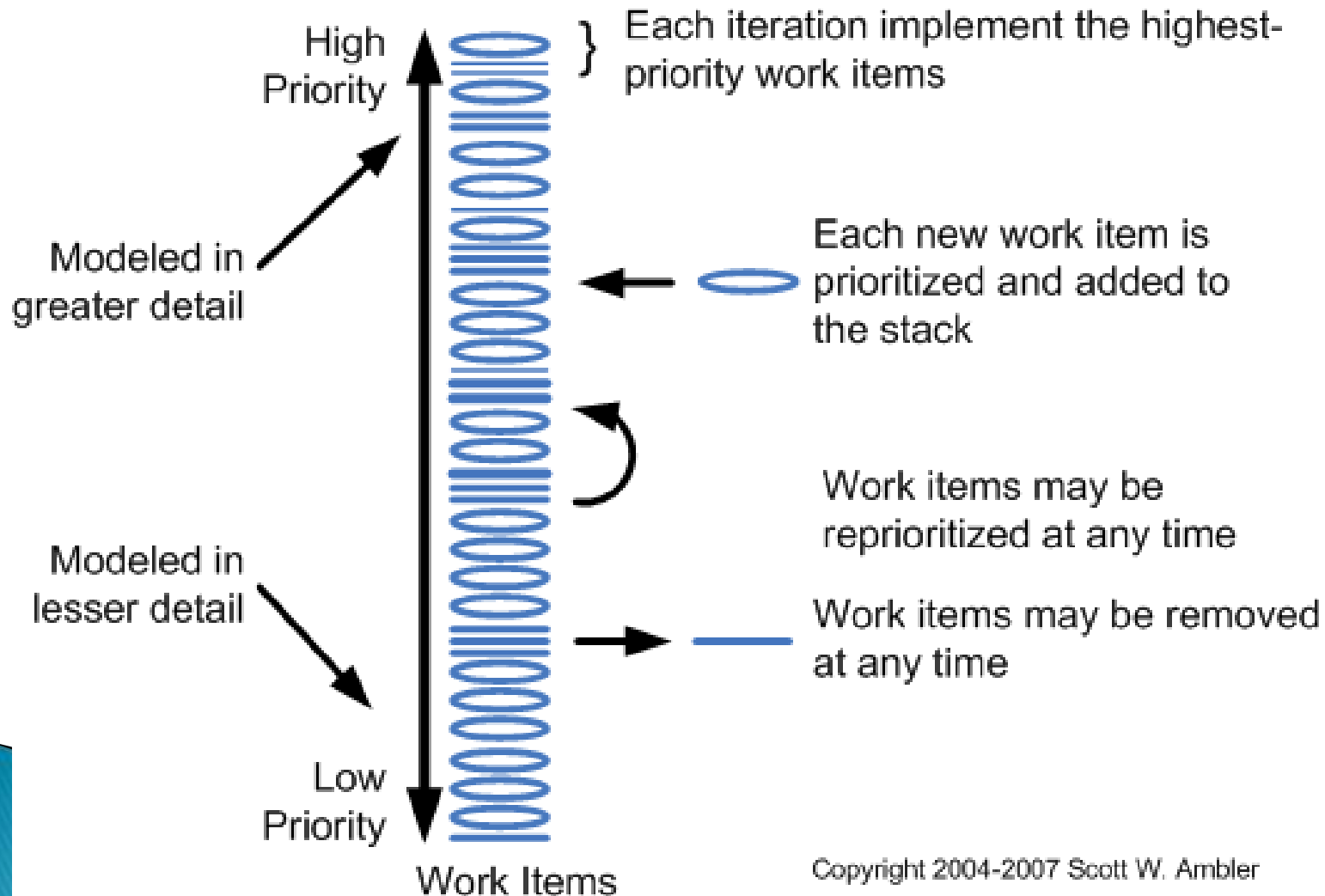
- ▶ Principiile Lean Software Development (LSD)
 1. Eliminarea lucrurilor nefolositoare
 2. Amplificarea învățării
 3. Decide cât mai târziu posibil
 4. Termină cât mai curând posibil
 5. Oferă **responsabilități** membrilor echipei
 6. Construiește un **proiect integru**
 7. Construiește văzând tot **proiectul în ansamblu**

Model Driven Development

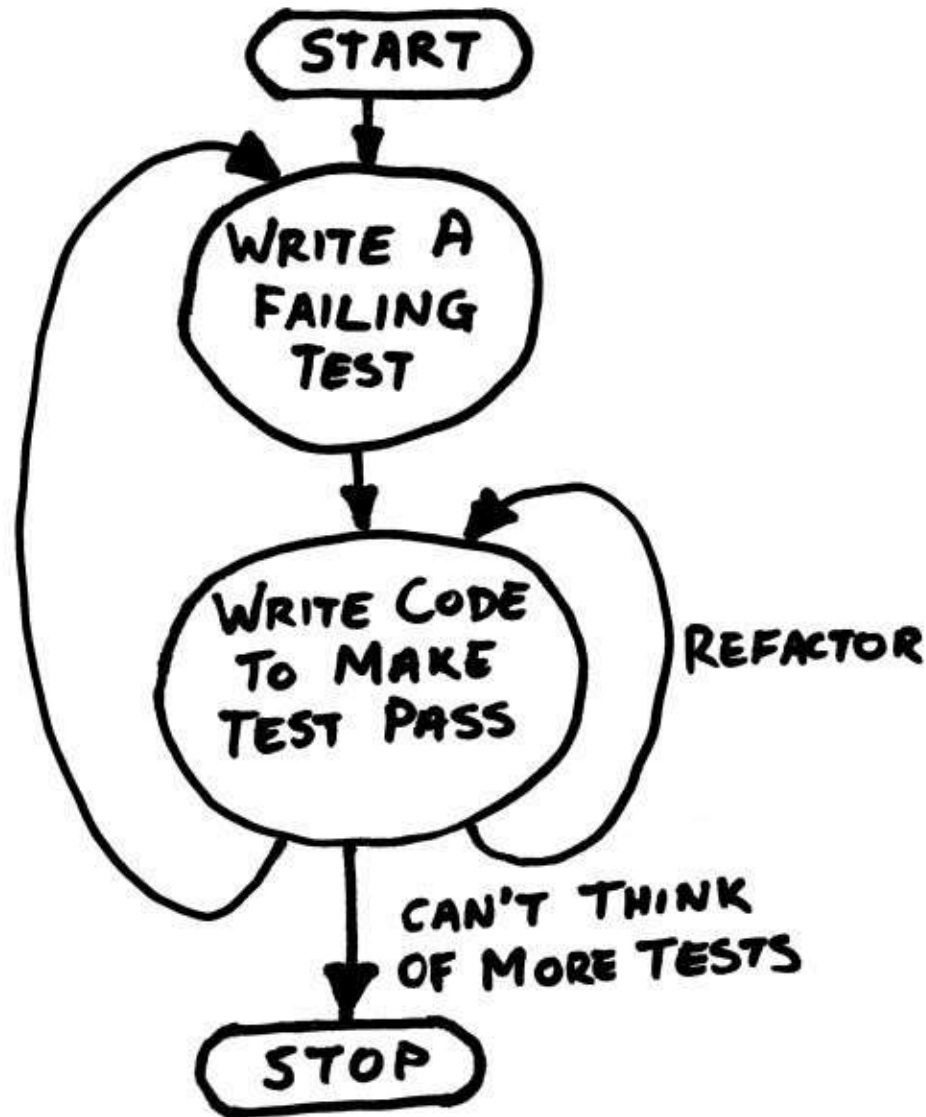
- ▶ *Model Driven Development* (MDD) is a paradigm for writing and implementing computer programs **quickly, effectively and at minimum cost**
 - ▶ MDD is an approach to software development where **extensive models are created before source code is written**
 - ▶ A primary example of MDD is the Object Management Group (OMG)'s Model Driven Architecture (MDA) standard
- 

Agile MDD – Iteration Modeling

▶ Thinking Through What You'll Do This Iteration



Test Driven Development



Myths

▶ Clients

- A general description of the objectives is sufficient to begin writing program
- Requirements are constantly changing, but the software is flexible and can easy adapts

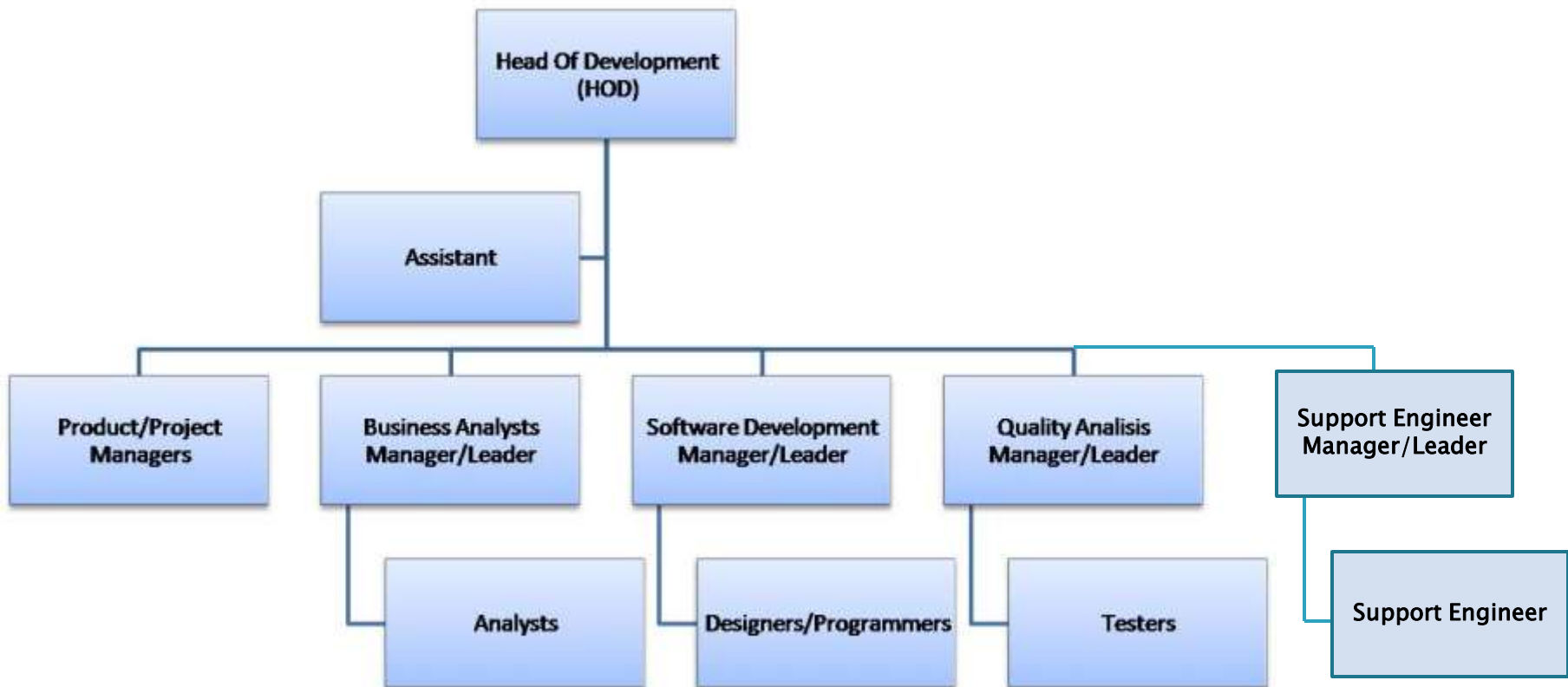
▶ Developers

- Once the program is written and it is functional, our role has ended
- Until the program doesn't work, we can not assess the quality
- The only good product is the functional program
- Software Engineering will create voluminous and unnecessary documentation and will cause delays

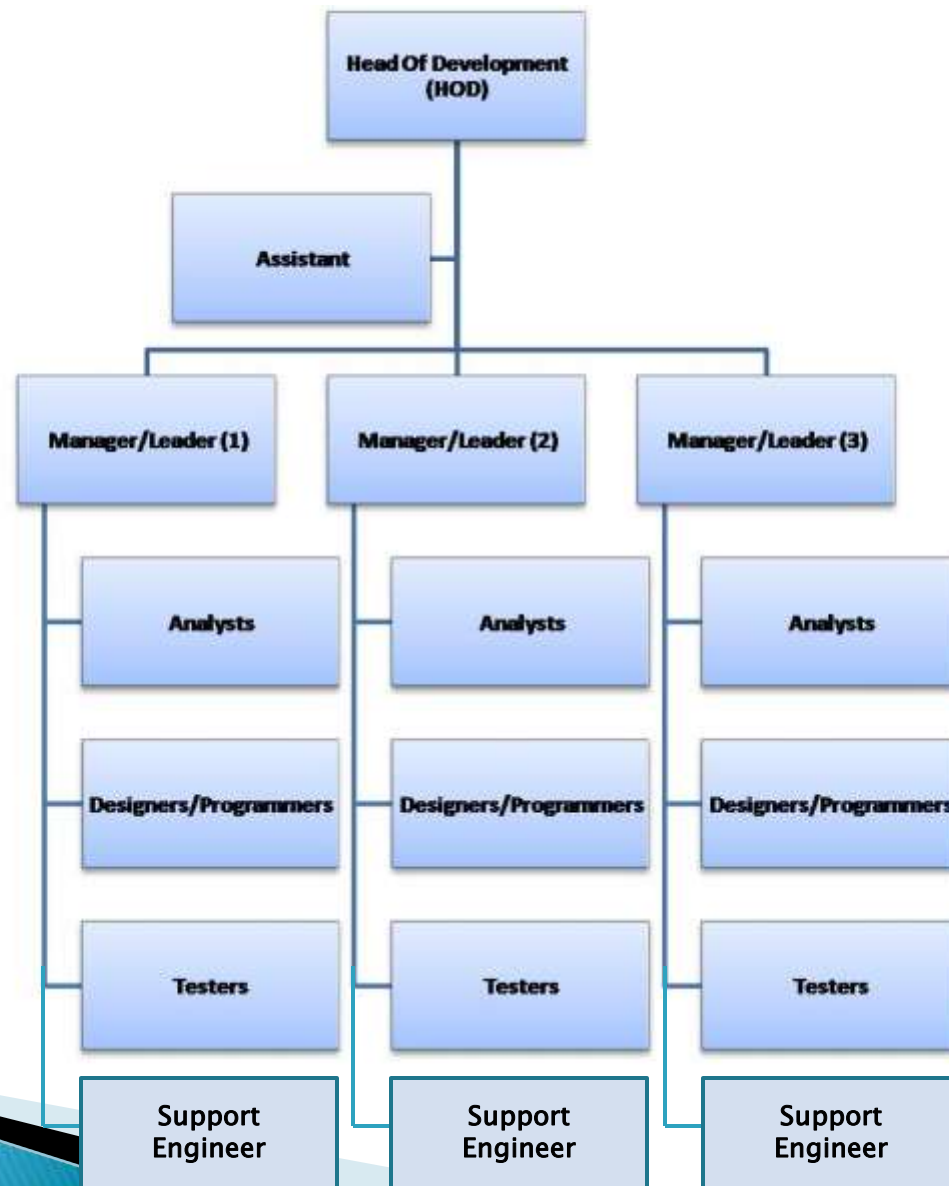
Cum vom face noi?

- ▶ O comunicare foarte bună cu CLIENTUL care face parte din echipă (SCRUM)
- ▶ După fiecare etapă veți obține un produs finit care de regulă nu va putea fi refăcut la pașii următori (Cascadă)
- ▶ Ca membru al “echipei” vă voi sprijini cât mai mult posibil (LEAN)
- ▶ Fiecare va fi încurajat să facă ce îi place mai mult (XP)
- ▶ NU am să aduc modificări continue în cerințele mele (NU AGILE)
- ▶ (NU) vom face un studiu de risc ((NU) MODEL ÎN SPIRALĂ)

Ierarhia dintr-o firmă de software – Compartimente



Ierarhia dintr-o firmă de software – Proiecte



Etapele dezvoltării programelor

- ▶ În engleză: **Software Development Life Cycle**
- ▶ Analiza cerințelor (Requirements analysis)
- ▶ Proiectarea architecturală (Architectural design)
- ▶ Proiectarea detaliată (Detailed design)
- ▶ Scrierea codului (Implementation)
- ▶ Integrarea componentelor (Integration)
- ▶ Validare (Validation)
- ▶ Verificare (Verification)
- ▶ Întreținere (Maintenance)

Cum începe un proiect?

- ▶ Un client dorește să-și
 - Îmbunătățească productivitatea
 - Rezolve o problemă de personal
 - Facă reclamă la produsele pe care le vinde
 - Gestioneze mai ușor activitatea sucursalelor din țară
- ▶ Un proiect interesant
- ▶ O idee, nevoia de a-mi gestiona cheltuielile zilnice, etc.
- ▶ **Din acest punct urmează Ingineria Cerințelor!**

Ingineria cerințelor (Requirement analysis) – Definiție

- ▶ Procesul înțelegerii nevoilor clientului și a așteptărilor acestuia de la aplicația noastră
- ▶ O etapă bine definită din ciclul de viață al dezvoltării unui produs (Software Development Life Cycle)
- ▶ La ce ne așteptăm de la o aplicație să facă
- ▶ Cum ar trebui sistemul să se comporte și care sunt caracteristicile acestuia

Ingineria cerințelor – Exemplu

- ▶ Realizați un program C++ care să realizeze suma a două matrici citite din fișier.
- ▶ +:
 - Se specifică limbajul
 - Știm că citirea se face din fișier
- ▶ -:
 - Nu știm ce să facem cu două matrici care nu au aceleași dimensiuni
 - Ce facem cu rezultatul?

Ingineria cerințelor – Cine?

- ▶ Datorită multitudinii de tipuri de interacțiuni care pot exista între utilizatori, procese de business, dispozitive hardware, etc., pot exista diverse tipuri de cerințe, de la aplicații simple, la aplicații complexe
- ▶ Procesul de analiză a cerințelor presupune alegerea și documentarea acestor tipuri de cerințe, și construirea documentelor ce vor constitui baza construirii sistemului
- ▶ **Cine se ocupă? Project Manager, Program Manager sau Business Analyst**

Ingineria cerințelor – De ce?

- ▶ Studiile făcute demonstrează că atenția insuficientă acordată analizei cerințelor este **cea mai des întâlnită cauză în cadrul proiectelor vulnerabile**
- ▶ Foarte multe organizații au cheltuit sume imense pe proiecte software care în final nu făceau ceea ce se dorea inițial de la ele
- ▶ În momentul de față foarte multe companii investesc timp și bani pentru a face o analiză a cerințelor eficientă

Ingineria cerințelor – Pași

1. Stabilirea limitelor aplicației
2. Găsirea clientului
3. Identificarea cerințelor
4. Procesul de analiză a cerințelor
5. Specificarea cerințelor
6. Gestionarea cerințelor

IC – Limitele aplicației (1)

- ▶ Ca prim pas, are ca scop identificarea modului în care această nouă aplicație **se va integra în mediul** pentru care va fi concepută
- ▶ Care va fi **scopul** aplicației
- ▶ Care vor fi **limitele** aplicației

IC– Găsirea clientului (2)

- ▶ **Obiectivul** ultimilor ani: *Cine este utilizatorul (clientul) care va folosi efectiv aplicația ?*
- ▶ **Ca rezultat**, vom ști exact ce persoane vor fi direct sau indirect afectate de realizarea acestui produs
- ▶ Vom ști pe cine să întrebăm pentru eventuale clarificări

IC – Identificarea cerințelor (3)

- ▶ Cerințele se colectează de la mai multe grupuri ce au fost identificate în etapa anterioară
 - ▶ Se identifică ce anume doresc aceștia ca aplicația să realizeze
 - ▶ **Nivelul de detaliere** depinde de:
 - Numărul și de dimensiunea grupurilor
 - Complexitatea procesului de business
 - Dimensiunea aplicației
 - ▶ **Probleme întâlnite în această etapă**
 - Ambiguități în înțelegerea proceselor
 - Inconsistență în înțelegerea aceluiași proces
 - Date insuficiente
- Modificări în cerințe după începerea proiectului

IC – Cine face identificarea cerințelor ? (3)

- ▶ Această persoană trebuie să interacționeze direct cu multe grupuri de lucru
- ▶ Are de a face cu idei contradictorii
- ▶ Trebuie să aibă abilități de comunicare și de lucru cu oamenii
- ▶ Trebuie să aibă cunoștințe de programare
- ▶ În final trebuie să cadă de acord cu clientul în privința cerințelor

IC – Metode folosite în identificarea cerințelor (3)

- ▶ Interviuri cu viitorii utilizatori și cu grupuri de utilizatori
 - ▶ Folosirea documentației existente (manuale de utilizare, diagrame ale organizației, specificații de sistem, etc.)
 - ▶ Metode:
 - Prototipuri
 - Diagrame “Use case”
 - Diagrame de flux a datelor și a proceselor
- Interfețe utilizator

IC – Procesul de analiză a cerințelor (4)

- ▶ Se face o analiză structurată care folosește tehnici specifice:
 - “animarea” cerințelor
 - Raționament automat
 - Privire critică din punct de vedere al cunoașterii
 - Verificarea consistenței
 - Raționament analogic și bazat pe exemple

IC – Specificarea cerințelor (5) 1

- ▶ Se face într-un mod **clar, neambiguu**
- ▶ **Scrierea unui document** în care se specifică cerințele este **obligatoriu!**
- ▶ Acest document va circula între toate persoanele implicate în această fază: *client, grupuri de utilizatori, echipele de dezvoltare și de testare*
- ▶ Documentul va fi folosit la:
 - Validarea cerințelor de către client
 - Contractul dintre client și echipa de dezvoltare
 - Bază pentru proiectarea sistemului de către dezvoltatori
 - Bază pentru planificări
 - Sursă pentru realizarea scenariilor de testare

IC – Specificarea cerințelor (5) 2

- ▶ Trebuie să surprindă **viziunea clientului** despre produs
- ▶ Reprezintă rezultatul colaborării dintre **utilizator** (care nu e un expert) și **analistul de sistem** (care surprinde situația în termeni tehnici)
- ▶ E posibil ca specificarea cerințelor să se facă în două documente separate:
 - **Cerințele utilizator** – scrise în clar folosind cazuri de utilizare (pentru utilizator)
 - **Cerințele sistemului** – descrise folosind un model matematic sau programatic (pentru dezvoltatori și pentru testerii)

IC – Specificarea cerințelor (5) 3

- ▶ În cerințele utilizatorului **nu trebuie** să apară noțiuni tehnice (protocol de comunicare, criptarea folosind MD5, http, IP, etc)
- ▶ În cerințele sistemului **trebuie** să apară formatul de export al datelor (XML), adresa serverului de pe care se fac citiri, locul în care se depozitează fișierele log

IC – Specificarea cerințelor (5) 4

- ▶ Nivelul de detaliere:
 - **Ridicat** – presupune multă muncă, uneori inutilă (este mai precis și mai clar)
 - **Scăzut** – poate fi vag (nu ajută în procesul de dezvoltare și testare)
- ▶ Exemplu:
 - Realizați un program care să facă suma a două matrici.
 - Realizați un program C# care să aibă clasa **Matrice** cu attributele **n,m** de tip **int** reprezentând numărul de linii și de coloane și **matrice** de tip **int[3][3]** reprezentând elementele matricii. Metodele disponibile în clasa **Matrice** sunt

IC – Specificarea cerințelor (5) 5

► Tipuri de cerințe:

- **Cerințe utilizator:** legate de locul unde va fi folosit sistemul, eficiență, durata de viață a produsului (*produsul va fi folosit de compartimentul financiar*)
- **Cerințe funcționale:** despre modul în care se fac anumite calcule, modul în care se manipulează datele (*impozitul pe salar este de 16 %*)
- **Cerințe de performanță:** modul în care anumite funcții sunt apelate cantitativ, calitativ (*sistemul va permite 1000 de interogări pe secundă*)
- **Constrângeri:** nu se va permite ca două persoane să introducă simultan date în tabele

IC – Gestionarea cerințelor (6)

- ▶ Este un proces continuu care surprinde toate aspectele identificării cerințelor și în plus asigură verificarea, validarea acestora
- ▶ Pentru a fi utilă trebuie să asigure neambiguitatea cerințelor, eliminarea erorilor și completarea omisiunilor

Scenarii de utilizare 1

- ▶ Folosesc **actori** (elemente cu care programul interacționează):
 - Utilizatori umani
 - Elemente software (Ex: program care prelucrează informațiile colectate de pe Internet)
 - Elemente hardware (Ex: cititor de coduri de bare, telefoane mobile, etc.)
- ▶ Folosesc **scenarii (use case)**
 - Acestea descriu cum interacționează actorul cu sistemul
 - Cum reacționează sistemul în urma acestor acțiuni
 - Care e rezultatul vizibil pentru actori

Scenarii de utilizare 2

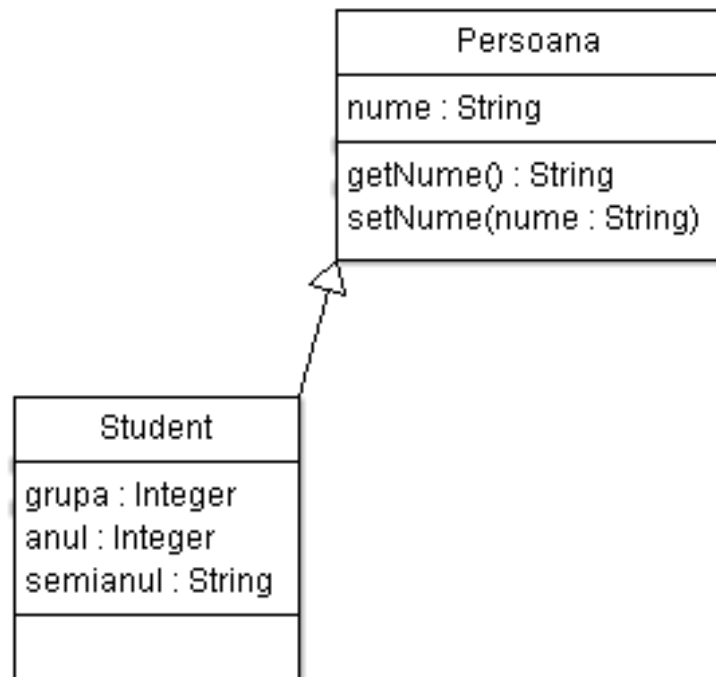
- ▶ Ce nu conțin acestea:
 - Diagrame de clase
 - Structura modulară a programului
 - Tipul datelor de intrare și de ieșire
- ▶ Use Case – Tipuri de conținut:
 - **Pe scurt** – descrie principalul caz de succes
 - **Cazual** – conține ce ar trebui făcut în caz că se întâmplă ceva
 - **Detaliat** – se prezintă pe larg toate situațiile posibile

Use Case – Exemple

- ▶ **Pe scurt:** Programul trebuie să poată aduna 2 matrici
- ▶ **Cazual:** Programul trebuie să poată aduna 2 matrici dacă au același număr de linii și de coloane, **altfel** se va afișa un mesaj de eroare corespunzător
- ▶ **Detaliat:** Programul trebuie să poată aduna două matrici de numere întregi citite de la tastatură, **dacă** au același număr de linii și de coloane, iar matricea rezultată se va afișa într-un fișier “rezultat.txt” câte o linie pe rând. **Altfel** se va afișa un mesaj de eroare corespunzător într-un fișier “mesaj.txt” aflat în directorul curent. (*Mai trebuie specificat ceva?*)

UML – Diagrame de clase

- ▶ Relația de generalizare



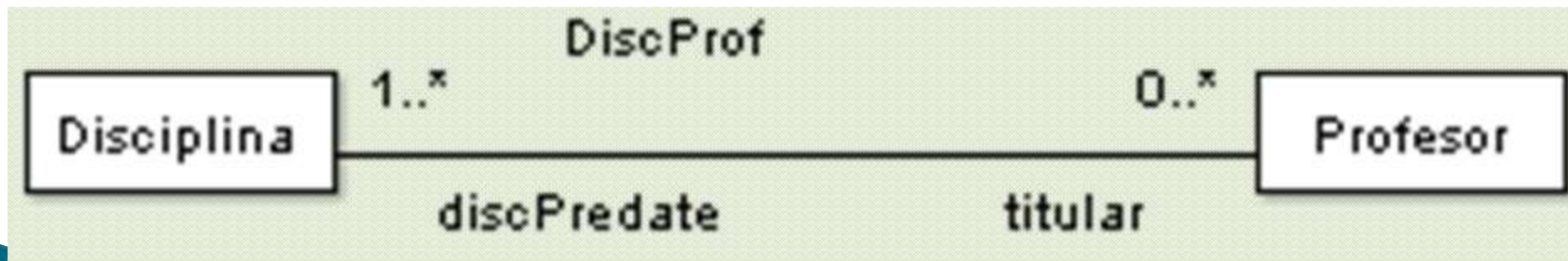
Relația de asociere 1

- ▶ Relația Student – Disciplină
 - **Student**: urmez 0 sau mai multe discipline, cunosc disciplinele pe care le urmez;
 - **Disciplină**: pot fi urmată de mai mulți studenți, nu cunosc studenții care mă urmează

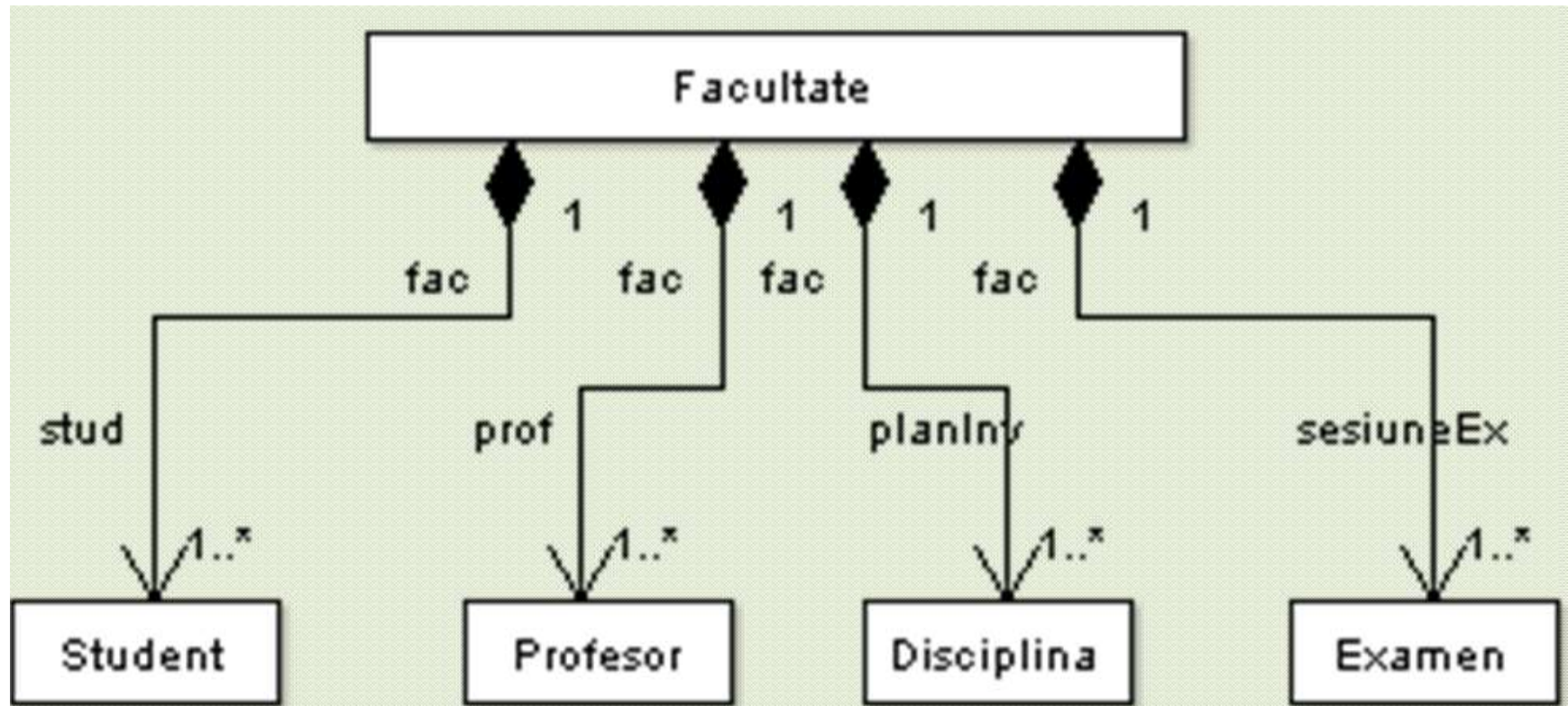


Relația de asociere 2

- ▶ Relația Disciplină – Profesor
 - **Disciplină**: sunt predată de un profesor, îmi cunosc titularul
 - **Profesor**: pot preda mai multe discipline, cunosc disciplinele pe care le predau



Relația de Compoziție (“hasA”)



ArgoUML

- ▶ Link: <http://argouml-downloads.tigris.org/argouml-0.34/>
- ▶ Varianta “zip” trebuie doar dezarhivată
- ▶ Trebuie să aveți instalat Java
 - În Path sa aveți c:\Program Files\Java\jdk1.6.0_03\bin
 - Variabila
JAVA_HOME=c:\Program Files\Java\jdk1.6.0_03\

Bibliografie

- ▶ Anil Hemrajani, Agile Java Development with Spring, Hibernate and Eclipse, 2006
- ▶ Dorel Lucanu, Principii POO

Links

- ▶ XP: <http://www.extremeprogramming.org/rules.html>
- ▶ Agile: <http://agilemanifesto.org/>
- ▶ Scrum: <http://jeffsutherland.com/oopsla/schwapub.pdf>
- ▶ Lean: http://www.projectperfect.com.au/info_lean_development.php
- ▶ V-model: <http://en.wikipedia.org/wiki/V-Model>,
http://en.wikipedia.org/wiki/V-Model_%28software_development%29
- ▶ Project Management White Paper Index:
http://www.projectperfect.com.au/wp_index.php
- ▶ Requirements analysis process:
<http://www.outsource2india.com/software/RequirementAnalysis.asp>
- ▶ ImageCup 2009:
<http://fiistudent.wordpress.com/2008/12/10/imagine-cup-2009-ce-ar-fi-daca-intr-o-zi-am-ajunge-toti-la-muzeu/>
- ▶ Curs 2 IP – Ovidiu Gheorghieș:
<http://www.infoiasi.ro/~ogh/files/ip/curs-02.pdf>
- ▶ Software house: http://en.wikipedia.org/wiki/Software_house