

Înainte de toate

- Prezentare asistent
- Prezentare/parcursare pagină [MPS Need to Know](#)
- Prezentare/parcursare pagină [Reguli generale și notare](#)

Componentă echipă

Există mai multe modele de lucru în echipă

- **model descentralizat/anarhic** - nu există o demarcare clară a rolurilor; funcționează în proiectele free/open-source bazate pe comunități cu o motivație internă
- **modele ierarhice** - există responsabilități clare pentru fiecare membru al echipei; în funcție de ierarhie se distribuie responsabilități și se transmit rapoarte
 - în general există un *Project Manager (PM)* al cărui rol este de coordonare, distribuție task-uri, motivare, feedback etc.
 - pot exista mai mulți *Lead Developeri (Team Leaderi - TL)* ale căror responsabilități sunt coordonarea, training-ul, oferirea de sfaturi tehnice, revizia codului și proiectarea unei anumite componente
 - dezvoltatorii (*Developeri - D*) participă la implementarea componentelor; în general, fac parte din echipe distincte dedicate unei anumite componente a proiectului
 - testerii (**T**) (pot fi tot dezvoltatori) verifică respectarea cerințelor, funcționarea în regim normal și în regim de stres a aplicației
- **combinație** - încearcă rezolvarea problemei: "conducătorii nu știu implicațiile deciziilor lor, iar lucrătorii sunt demotivați de decizii luate aiurea", dar fără să elimine ierarhia [the alternative to hierarchy or anarchy](#)
 - În proiectele mari pot apărea mai multe roluri. Vom avea o abordare mai detaliată în [laboratorul 2](#).

Case study: Strategie de PM pentru dezvoltarea jocurilor video

Timpul estimat pentru aceasta activitate este de 45 de minute.

O companie internațională de software la comanda are în vedere realizarea de proiecte proprii legate de industria jocurilor video. Compania nu are experiența în domeniul jocurilor, dar are deja mai multe produse legate de industria divertismentului (film, cărți). Compania a avut până acum clienți și o piață matură, dar vor să-și extindă produsele către toate categoriile de vârstă. Proiectul lor trebuie să includă și cercetare de piață a jocurilor video, implicând mai mulți profesioniști de marketing și IT. Primele obiective ale proiectului sunt cercetarea pieței actuale și proiectate, dezvoltarea și administrarea de sondaje online pentru colectarea datelor, grupuri focus pentru a ajuta la înțelegerea pieței potențiale și a grupurilor țintă, analizarea datelor, stabilirea dacă această companie ar trebui să intre sau nu pe această piață și cum să elaboreze un plan pentru a merge mai departe. Să presupunem că ați devenit membru al echipei prezentate mai sus. Sunteți managerul de proiect.

- Alegeți-vă o echipă, precizați rolurile fiecăruia, justificați-le. Ce experiență este necesară pentru fiecare rol?
- Precizați care sunt skill-urile pe care trebuie să le aveți ca project manager.
- Să presupunem că în urma cercetării pieței, hotărâți să demarați procesul de dezvoltare al primului vostru joc video. Ați dori ca întregul proces să dureze doi ani.

Care sunt tema & tipul jocului, in urma analizei pietei? Realizati o planificare (etape, persoane responsabile, timp, buget) pentru a dezvolta jocul video ales.

De făcut pentru laboratorul următor

Să vă împărțiți pe semigrupe. Să nu uitați ca repartizarea pe [forumul de discuții generale](#) este deschisa. Mai multe informații găsiți la secțiunea [Semigrupe de laborator](#) și pe pagina de [Facebook](#) a echipei.

Laborator 2 - Rolurile în echipă. Redactarea specificațiilor de proiect.

Rolurile în cadrul echipei de proiect

Cu cât proiectul este mai mare, cu atât rolurile din echipă sunt mai diversificate (mai specializate pe o anumite arie de activități).

Într-un **proiect mic**, principalele roluri din echipă sunt:

1. **Project Manager / Manager-ul Proiectului**
 - responsabil de succesul/eșecul proiectului
 - planifică & monitorizează evoluția proiectului
 - ia decizii
 - adesea, are background tehnic (pentru a înțelege ciclul de viață al proiectului)
 - întotdeauna, are abilități soft (de conducere, de motivare și stimulare, de conștientizare a cauzelor, de decizie, de negociere, de previziune, de inovație)
 - certificări PMI (Project Management Institute): PMP (Project Management Professional)
2. **Team Leader / Liderul echipei** (Dezvoltator Lider, QA Lider)
 - responsabil de execuția conform standardelor a activităților planificate
 - monitorizează activitatea echipei sale
 - intervine pentru a corecta, pentru a îndruma (coaching/support)
 - raportează managerului statusul real al proiectului
3. **Software Developer / Dezvoltator software**
 - responsabil de implementarea software a cerințelor
 - în metode de dezvoltare agile contribuie și la design, arhitectură, specificații
4. **Tester** (Inginerul Calității)
 - responsabil de verificarea funcționalităților și a performanțelor produsului
 - scrie scenarii de test, le execută, analizează rezultatele ⇒ rapoarte de testare

Într-un **proiect mare**, pe lângă rolurile de mai sus, apar și altele, precum:

1. **System Architect / Arhitectul Sistemului**
 - responsabil de arhitectura produsului software
 - captează interesele partenerilor de business și le transpune alegând arhitectura potrivită
2. **Technical Writer / Scriitorul tehnic**
 - responsabil de documentația tehnică a proiectului
 - primește documente tehnice de la ingineri, pe care le editează spre a fi corecte, clare și conforme cu standardele în vigoare
3. **Analyst / Analist** (Analist de Business, Analist de Sisteme Business, Analist de Sistem, Analist de Cerințe)
 - responsabil de înțelegerea corectă a cerințelor de business
 - studiază specificațiile clienților, clarifică toate detaliile proiectului cu clientul și cu partenerii (stakeholders)
 - redactează specificațiile aplicației software
4. **Consultant**
 - analizează fezabilitatea ofertelor de proiecte
 - propune soluții pentru diverse probleme legate de ciclul de viață al proiectelor
 - evaluează gradul de atingere a obiectivelor proiectelor

5. **Experți business:** SME (Subject Matters Expert), Manager-ul Produsului, Manager-ul de program

Gestiunea timpului

Orice membru al echipei și project manager-ul în special trebuie să aibă skill-uri de organizare a timpului pentru a obține maximum de rezultate. Aspecte precum planificarea task-urilor, stabilirea obiectivelor, evitarea întreruperilor, prioritizarea activităților trebuie avute constant în vedere pentru eficientizarea muncii.

Redactarea specificațiilor de proiect

După primirea specificațiilor clientului și după analiza de profunzime a acestora, contractorul redactează *documentele de specificație* ale soluției oferite, de obicei sub formă de:

- **Software Requirements Specification / Specificarea Cerințelor Software** (SRS/SCS)
- **Software Design Description / Descrierea Design-ului Software** (SDD/DDS)

Este **esențial** pentru reușita proiectului ca aceste documente de specificații să fie complete și absolut clare deoarece:

- acestea reprezintă parte a contractului dintre client și contractor;
- pe baza acestora se pot determina clar cerințele proiectului, rezolvându-se astfel eventualele conflicte dintre parteneri
- pe baza acestora se vor realiza estimările de timp și buget și planificările de proiect;
- pe baza lor se va realiza dezvoltarea soluției și testarea funcționalităților și performanțelor ei;
- pe baza lor se va realiza evaluarea finală a proiectului (a gradului de atingere a obiectivelor).

Documentul SRS cuprinde descrierea completă a comportamentului sistemului software, adică:

- a interacțiunilor dintre utilizator și sistem (**cerințe funcționale**). Acestea se descriu cu ajutorul *use-cases* (cazurilor de utilizare).
- a constrângerilor de proiectare și dezvoltare (**cerințe non-funcționale**). Acestea se referă la restricții de performanță, de securitate, de fiabilitate etc.

Conținutul unui SRS/SCS

- Scopul documentului (*Document purpose*)
- Conținutul documentului (*Document overview*)
- Descrierea generală a produsului (**General description of the product**)
- Situația curentă (*The current situation*)
- Misiunea proiectului (*Purpose of the product*)
- Contextul proiectului (*Product context*)
- Beneficii (*Benefit*)
- Cerințe funcționale (**Functional requirements**)
- Actori (**Actors**)
- Diagrama de sistem (**System boundary**)
- Descrierea cazurilor de utilizare (**Use cases description**)
- Cerințe nefuncționale (**Non-functional requirements**)
- Cerințe de interfață (*User Interface Requirements*)
- Cerințe de performanță (*Performance Requirements*)
- Cerințe de fiabilitate (*Availability & Reliability*)
- Cerințe de securitate (*Security Requirements*)

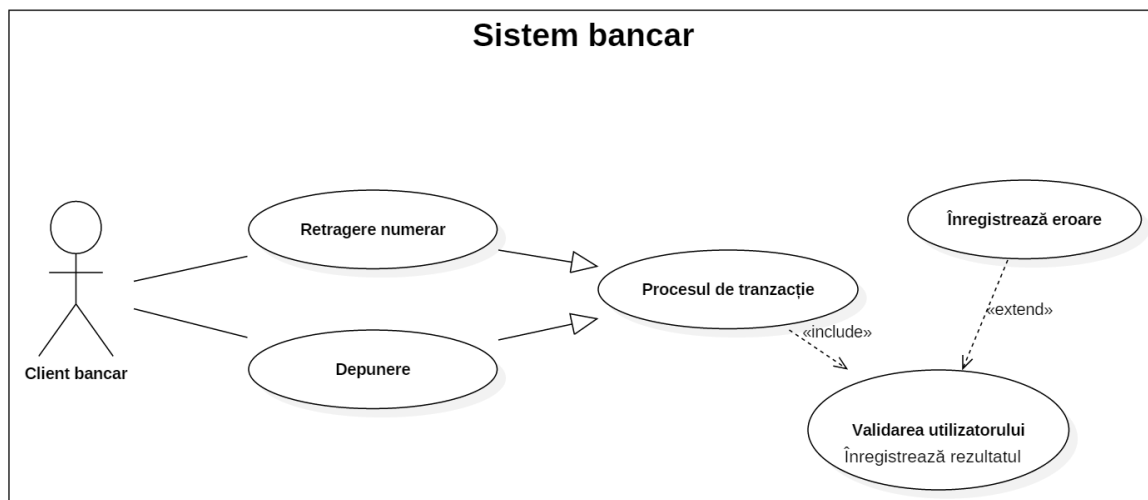
System Boundary

Se mai numește și *Use-case diagram* / *Diagrama de cazuri de utilizare*.

Este:

- o diagramă UML obținută în urma analizei cerințelor utilizatorului
- schema funcționalităților oferite de sistemul software, în termeni de actori, cazuri de utilizare și relații între acestea.

Mai multe detalii, poți afla [aici](#) și [aici](#).



Astfel de diagrame se pot realiza în [Visio](#), [Visual Paradigm](#) sau [Dia](#).

Exemplu de document SRS

[Exemplu de Document SRS - Alumni database](#)

[Exemplu de Document SRS - Simulator cursa formula 1](#)

Arhitectura proiectului

Stabilirea arhitecturii unui sistem software implică:

1. *alegerea șabloanelor arhitecturale* potrivite (se pot combina mai multe, se pot personaliza)
2. în funcție de șabloanele arhitecturale alese, *proiectarea structurii* sistemului software la nivel de:
 - **componente** specializate pe o gamă de servicii (module și interfețe),
 - **conexiuni** dintre componente.

Există mai multe tipuri șabloane arhitecturale (**architectural patterns**), precum:

- **Tehnică de calcul distributivă**
 - sistemul software e alcătuit dintr-o serie de componente software ce rulează pe mașini de calcul diferite, ce comunică prin rețea
 - mașinile de calcul interacționează pentru a oferi cât mai rapid un răspuns corect la cererea utilizatorului
- **Client-server**
 - arhitectură distribuită
 - conține componente furnizoare de servicii = componente *server* și componente ce solicită servicii = componente *client*
- **Arhitectură bazată pe evenimente**
 - se bazează pe producerea, detectarea, consumarea și reacția la evenimente
- **Front-end și back-end**

- front-end = interfață de colectare a datelor de la utilizator și de procesare a acestora pentru a fi aduse la formatul prevăzut de componenta back-end
- back-end = componenta software de procesare a datelor furnizate de front-end
- **Modelul pe trei nivele**
 - arhitectură de tip client-server
 - conține: nivel de prezentare (browser web), nivel de logică business (middleware - server de application: "motor" de rulare a paginilor web dinamice scrise în ASP.NET, JSP/Java, PHP, Perl, Python, Ruby), nivel de bază de date (server de bază de date)
 - browserul trimite cererile către serverul de aplicație, care le servește interogând și modificând conținutul bazei de date și, de asemenea, generează pagina de răspuns care va fi trimisă înapoi browserului.

Exemplu de descriere a arhitecturii unei aplicații: **Platformă de gestiune a datelor medicale electronice**

Software Design Document (SDD)

- document de **specificație** a soluției pentru sistemul software descris în **SRS**
- răspunde la întrebarea: *cum va fi construit sistemul software pentru a avea comportamentul descris în SRS?*
- prezintă metodologii, tehnologii, participanți și resurse implicate în proiect
- se poate redacta numai după finalizarea SRS-ului fiind un răspuns la cerințele prezentate în SRS
- este redactat de o echipă de software designers (proiectanți de sistem) și analiști business, pe baza SRS-ului și a experienței acestora
- reprezintă ghidul de construire a soluției folosit de echipa de dezvoltare a proiectului
- reprezintă un tool de analiză a întregului proiect în faza de început cât și de monitorizare pe parcurs

Secțiuni SDD

Un SDD are următoarele secțiuni:

1. **Scopul documentului (Document purpose)**
2. **Obiective (Objectives)**
3. **Conținutul documentului (Document overview)**
4. **Modelul datelor (Data Design)**
 - prezintă *structurile de date* importante, *formatele fișierelor* folosite în cadrul soluției și *schema bazei de date*.
 - Structurile de date pot fi:
 - **globale** (structuri de date disponibile tuturor componentelor arhitecturii)
 - **de legătură** (structuri de date trimise ca argumente între componente pentru a asigura fluxul informației la nivel de aplicație)
 - **temporare** (structuri de date folosite temporar).
 - Schema bazei de date este descrisă prin:
 - **diagrama schemei** bazei de date
 - **descrierea semnificației tabelelor** și, pentru fiecare tabelă, descrierea semnificației coloanelor și indicarea cheilor primare și referențiale.
5. **Modelul arhitectural (Architectural Design)**
 - este o structură ierarhică alcătuită din componente interconectate
 - prezintă arhitectura sistemului – atât descriptiv, cât și sub forma unei diagrame de arhitectură
 - descrie:

- fiecare componentă a arhitecturii,
- restricțiile de implementare ale componentelor,
- interacțiunea dintre componentele sistemului.
- poate fi reprezentat prin :
 - **diagrame de componente** (pentru proiecte mari) - le-am numit "generic" în laboratorul 2: diagrame de arhitectură
 - **diagrame de clase**, în care relațiile ierarhice se bazează pe generalizare și specializare (pentru proiecte mici).

6. Modelul interfeței cu utilizatorul (*User Interface Design*)

- prezintă *ferestrele* aplicației și *succesiunea* lor în cadrul sistemului.

7. Elementele de testare (*Testing Issues*)

- identifică *componentele critice* ale aplicației (componente a căror performanță influențează decisiv performanța globală a aplicației)
- propune *alternative* de proiectare a componentelor critice (pentru a fi folosite în cazul insuccesului soluției propuse inițial).

Exemplu de Document SDD

Exemplu de document SDD mai vechi: [Exemplu de Document SDD](#)

Exemplele de mai jos sunt mai noi. Unul se remarcă prin organizarea clară a informațiilor și structură/organizare, dar și prin concizie, însă Scopul și Obiectivele nu sunt în totalitate corecte. Cel de-al doilea se remarcă prin forma simplă și schematică, dar care include toate secțiunile importante (și chiar extra) într-un mod concis.







Exemple mai noi

Exerciții

Metoda pălăriilor gânditoare (30 - 40 minute)

Împărțiți-vă în echipe de 6 persoane.

Asistentul va propune câte o situație pentru fiecare echipă. Fiecare membru al echipei alege una dintre pălării (alegeri diferite) și aduce argumente specifice acestora în rezolvarea problemei.

	The White Hat calls for information known or needed. "The facts, just the facts."
	The Yellow Hat symbolizes brightness and optimism. Under this hat you explore the positives and probe for value and benefit.
	The Black Hat is judgment - the devil's advocate or why something may not work. Spot the difficulties and dangers; where things might go wrong. Probably the most powerful and useful of the Hats but a problem if overused.
	The Red Hat signifies feelings, hunches and intuition. When using this hat you can express emotions and feelings and share fears, likes, dislikes, loves, and hates.
	The Green Hat focuses on creativity; the possibilities, alternatives, and new ideas. It's an opportunity to express new concepts and new perceptions.
	The Blue Hat is used to manage the thinking process. It's the control mechanism that ensures the Six Thinking Hats® guidelines are observed.

Team Roles Test (15 minute)

Într-o clasificare de roluri pe echipă apar următoarele, descrise mai jos.

Președinte: Președintele are un rol puternic de coordonator. Punând accent pe proceduri, președintele va încerca să aducă laolaltă și să păstreze echipa împreună. El sau ea comunică și se ocupă de membrii echipei într-un mod respectuos și deschis.

Expert: Expertul are abilitățile și expertiza necesare pentru sarcina specifică la îndemână. El sau ea se concentrează puternic pe sarcină și poate intra într-o atitudine defensivă atunci când alții interferă cu munca lui sau ei. Expertul preferă să lucreze singur, iar membrii echipei dețin adesea o mare încredere în acesta/aceasta.

Executiv: Executivul este uneori menționat și ca organizator. Executivul este în general disciplinat și dornic să-și facă treaba. Este eficient, practic și sistematic. Executivii sunt bine organizați și harnici și transformă rapid ideile unei echipe în acțiuni concrete și planuri practice.

Supraveghetor: Supraveghetorul este în general foarte ambițios și energic. El sau ea poate să pară nerăbdător(oare) și impulsiv(ă). Supraveghetorul este un motivator puternic și îi va provoca pe alții în momentele vitale. Cu toate că acțiunile acestuia pot părea uneori oarecum emoționale, ele joacă un rol crucial în conducerea echipei spre reușită.

Cercetător: Cercetătorul este, în general, un extrovertit prin natură. El sau ea este vesel(ă), gregar(ă). Cercetătorul este, de asemenea, un bun investigator, interesat și curios despre lucruri. Deoarece doresc să improvizeze și să comunice cu alții, vor avea o mică problemă în a prezenta ideii echipei și în a dezvolta noi contacte.

Analist: Analistul are tendința de a fi rezervat și critic. Analistul va reacționa, de asemenea, la planuri și idei într-un mod rațional și sensibil. El sau ea va favoriza o abordare prudentă a problemelor și le va evalua în funcție de exactitatea lor înainte de a acționa.

Integrator: Integratorul este foarte conștiincios și se simte responsabil pentru realizările echipei. Integratorii sunt preocupați de erori și au tendința de a-și face multe griji din cauza naturii lor de control. Integratorul este, de asemenea, cunoscut sub denumirea de finisor, deoarece este folosit cel mai eficient la sfârșitul unei sarcini, pentru a lustrui și a controla munca pentru erori, supunându-l la cele mai înalte standarde de control al calității.

Inovator: Inovatorul este adesea generatorul creativ al unei echipe. Are o imaginație puternică și o dorință de a fi original. Inovatorul preferă să fie independent și tinde să abordeze sarcinile într-un mod științific. Ca individ creativ, inovatorul poate juca un rol crucial în modul în care o echipă abordează sarcini și rezolvă probleme.

Jucător de echipă: Jucătorul echipei e grijuliu, evită conflictele și promovează armonie. Fiind cineva căruia îi place să ajute alte persoane, jucătorul echipei este în general considerat agreabil și prietenos. El sau ea este diplomatic și subliniază solidaritatea și coeziunea echipei. Fiecare dintre noi are ponderi pentru fiecare rol. Faceți o estimare a rolului pentru care considerați că aveți cea mai mare pondere. Apoi faceți o estimare a rolului fiecărui membru al echipei din care faceți parte (fiecare face estimarea proprie).

Apoi faceți fiecare **testul de aici**. Vedeți ce rezultate obțineți atât voi cât și restul echipei și vedeți cât de bine ați estimat rolul vostru și al fiecărui membru al echipei.

Lucru la proiect (35-40 minute)

Echipele vor fi formate din 4-6 studenți. Apoi urmează stabilirea unui manager de proiect și a celorlalte roluri pe baza testelor efectuate mai sus (1 x Project Manager, 1x Team Leader, 1 x Tester, 1-3 x Dezvoltatori)

În echipă gândiți-vă cum veți aborda proiectul. Puteți folosi hârtie fizică, un fișier .txt sau .doc/.odt, un document Google sau o pagină wiki sau puteți trimite pe e-mail primele idei. Urmăriți:

- Ce soluții software vă propuneți să utilizați?
- Ce arhitectură de proiect urmăriți (în linii mari)?
- Care sunt responsabilitățile macro în cadrul echipei?
- Care este planul pentru următoarea săptămână (ce aveți de făcut fiecare)?
- Care este agenda următoarei întâlniri de proiect (din laboratorul următor)? Dau dacă planificați o întâlnire înainte de următorul laborator.
- Ce veți folosi pentru comunicare/colaborare?
 - Un director [Google Drive](#) în care veți pune documente de organizare. Va fi partajat cu asistentul.
 - O listă de discuții, un grup de Facebook sau ce doriți voi pentru comunicarea în cadrul echipei.

Să puneți la punct de pe acum resursele pe care le veți folosi.

Nu vă recomandăm să vă gândiți **acum** la un timeline al celor cinci săptămâni de lucru la proiect, dar ar trebui să-l aveți în vedere în următoarele zile ce să știți:

- ce aveți de făcut
- cine ce face
- care este termenul pentru fiecare acțiune

Bonus: Testul culorilor

[Testul culorilor](#) este un test care prezintă personalitatea cuiva pe 4 culori:

- Mov - Loial, Demn de încredere, Pregătit, Organizat, Îngrijitor
- Portocaliu - Fermecător, Spontan, Are impact, Optimist, Îndrăzneț, Definitiv
- Albastru - Flexibil, Arătos, Călduros, Milos, Imaginativ
- Verde - Analitic, Relaxat, Inventiv, Rațional, Teoretic

Pentru fiecare culoare de mai sus estimați ce scor veți obține între 0 și 100.

După ce faceți estimarea faceți și testul. Comparați estimările și rezultatele.

Cum să ne organizăm pentru a atinge obiectivele definite în SRS conform planurilor din SDD?

În managementul de proiect, există o serie de utilitare de organizare a lucrului în proiecte, precum:

- **WBS**
- **Grafic Gantt**

WBS (Work Breakdown Structure)

- structură arborescentă cu rolul de a determina și de a grupa **sarcinile proiectului** în funcție de categoria din care fac parte
- ajută la organizarea proiectului prin clarificarea obiectivelor, a **elementelor de lucru** mai generale (*work packages*) și mai specifice (*work elements*)
- pe baza WBS, se pot face estimări de durată, de resurse umane și de buget → *suport pentru construirea diagramei Gantt*
- ajută la identificarea relațiilor de precedență dintre elementele de lucru (care elemente de lucru sunt independente, care sunt restricționate să aibă loc înainte de altele, etc.) - elemente vizibile în diagrama Gantt.
- WBS reprezintă graful acțiunilor necesare unui proiect grupând acțiunile în categorii de interes
- oferă suport pentru identificarea tuturor acțiunilor necesare și prioritizarea lor în funcție de efort, cost, durată etc.

Modul de construire a WBS

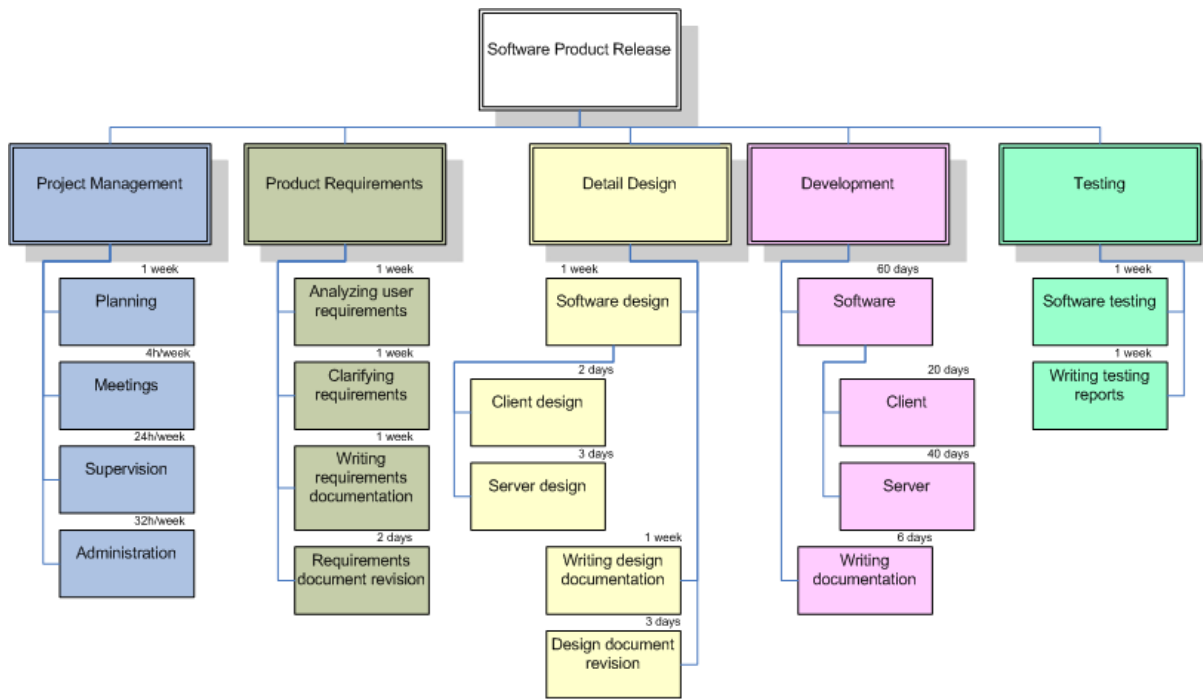
- nodurile arborelui conțin elementele de lucru, care pot fi produse, date, servicii
- rădăcina arborelui coincide cu obiectivul proiectului: sistemul software de dezvoltat
- nivelul al doilea al arborelui: subsistemele componente ale sistemului
- tot așa în adâncimea arborelui, în sensul că orice nod părinte înseamnă un element de lucru general, iar copiii reprezintă elemente de lucru componente ale lui
- deci, se detaliază efortul de dezvoltare a proiectului, de la obiectivul general la elemente de lucru și sarcini practice.
- nivelul de detaliere al unui WBS nu este foarte ridicat; astfel, adâncimea arborelui nu depășește, adesea, mai mult de 4-5 niveluri.

Reguli

- un WBS trebuie să cuprindă prin elementele sale **tot** efortul necesar dezvoltării proiectului (100% din muncă)
- elementele de muncă de pe același nivel din arbore nu se acoperă total sau parțial (nu sunt *overlapping*)
- frunzele din arbore au adesea asociate resurse, durate și costuri.

Tool-uri de creare a WBS-urilor: [WBS Schedule Pro](#), [Online Visual Paradigm](#), [Planhammer.io](#)

Exemplu de diagramă WBS a unui proiect software:



Resources:

Project Manager:
PM_Name

Requirements team:
Leads_name,
Members_name

Design team:
Leads_name,
Members_name

Development team:
Leads_name,
Members_name

Testing team:
Leads_name,
Members_name

Grafic Gantt

- grafic pentru **planificarea temporală** a sarcinilor proiectului
- construit pe baza activităților identificate în **WBS**
- indică data de început și de sfârșit a activităților din WBS și dependențele dintre activități:
 - start-to-start
 - finish-to-start
 - finish-to-finish
- întotdeauna se va ține cont de potențialele întârzieri și vor incluse în planificare sub o formă sau alta (review, delay, etc.)
- graficul Gantt este unul orientativ și starea sa se poate altera în timpul proiectului în funcție de situație
- acțiunile în grafic sunt trecute ca începând cel mai curând posibil, urmând să se asigure o marjă de eroare până la momentul cel mai târziu posibil
- este un tool bun pentru monitorizarea stării proiectului în timp, în funcție de gradul de realizare a acțiunilor până la un moment dat
- adesea, conține și următoarele elemente:
 - o linie verticală ce marchează timpul prezent
 - alte linii verticale cu milestone-urile proiectului
 - starea activităților exprimată prin procente (100% înseamnă că activitatea a fost realizată complet).

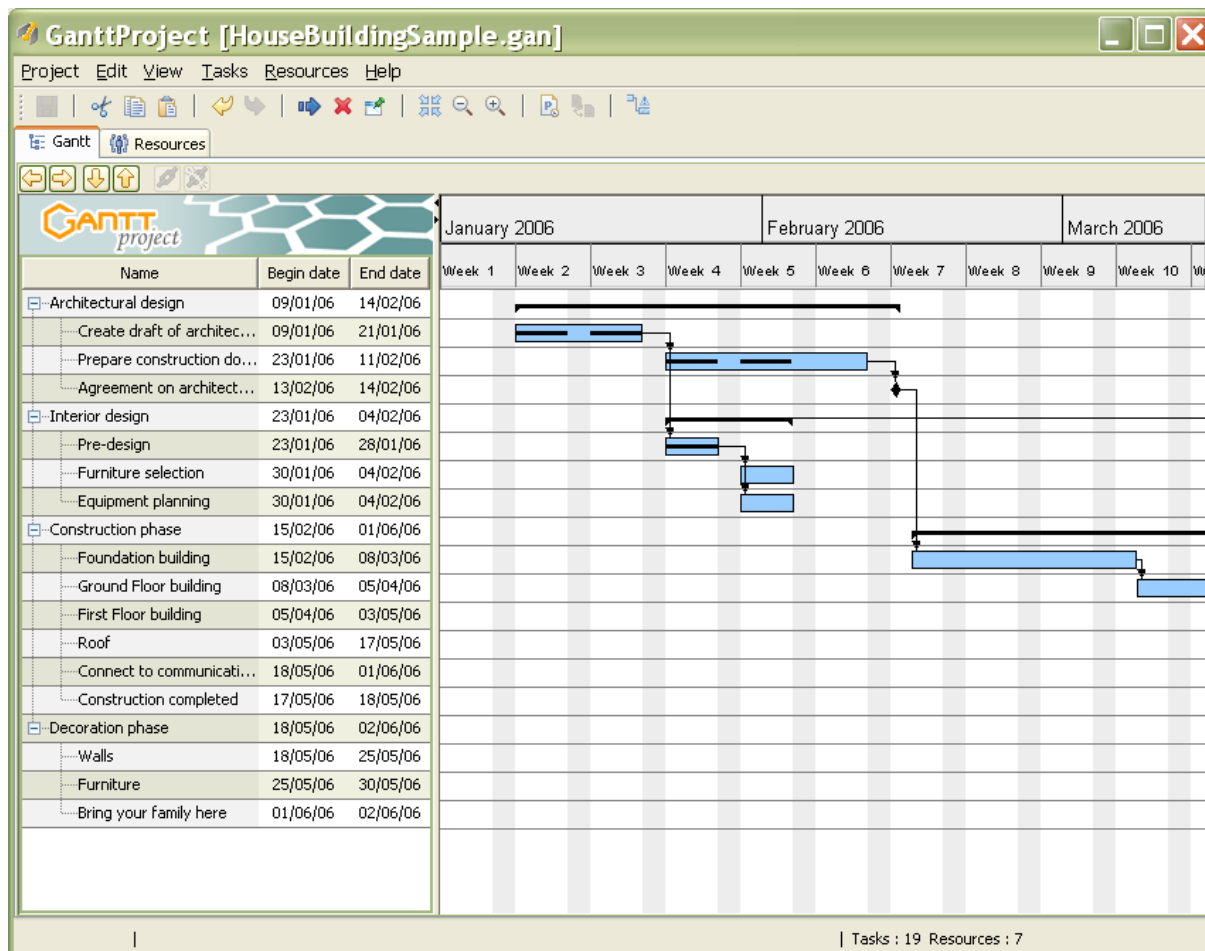
Organizarea unui proiect înseamnă parcurgerea etapelor:

1. Determinarea activităților proiectului (WBS)

2. Estimarea duratelor activităților și a consumului de resurse (astfel încât costul total să fie mai mic decât bugetul alocat)
3. Planificarea activităților în funcție de dependențe și de eventualele restricții legate de resurse (*Gantt*)
4. Stabilirea de *milestone-uri*.

Tool-uri de creare a diagramelor Gantt: [Gantt Project](#), [Team Gantt](#), [Gantt](#), [Gantt Excel](#), Microsoft Project.

Exemplu de diagramă Gantt a unui proiect de construcție de casă:



Milestone

- deadline de finalizare a unei etape importante și decisive a proiectului
- adesea presupune predarea unor livrabile către beneficiar
- în vederea respectării milestone-urilor fixate, adesea se impune luarea unor decizii cu impact major asupra evoluției proiectului.
- deși reprezintă puncte cheie în cadrul unui timeline, din varii motive acestea pot fi decalate dacă acest lucru ajută și nu au un impact negativ asupra proiectului.

Controlul versiunii (Git)

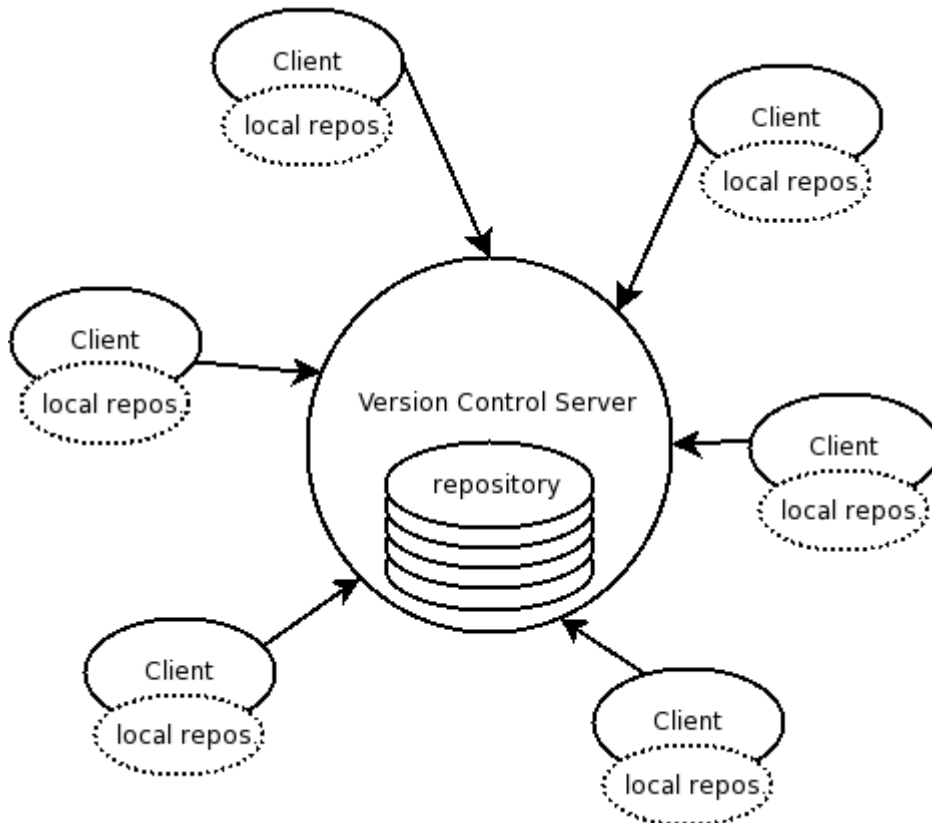
Sisteme pentru controlul versiunii (*Version Control Systems* - **VCS** - sau *Source Code Management* - **SCM**) sunt aplicații care permit lucrul colaborativ pe diverse fișiere, în special fișiere cod sursă. Sistemele pentru controlul versiunii sunt practic obligatorii în cadrul unui proiect cu dezvoltatori multipli. Astfel de sisteme rețin istoricul modificărilor efectuate de fiecare dezvoltator și folosesc comenzi specializate care să faciliteze transmiterea acestor modificări între dezvoltatori.

Exceptând sistemele de gestiune a surselor, prezentate mai detaliat în continuare, și alte aplicații folosesc versiuni:

- Wiki Engines - pentru fiecare modificare se rețin doar schimbările de la versiunea anterioară; orice modificare poate fi anulată;
- Google Docs - versiuni pentru fiecare modificare;
- MS Office, OpenOffice - permit atașarea unor numere de identificare pe documentele editate.

Principiul de funcționare a sistemelor de gestiune a codului este comun:

- Sursele sunt păstrate, de obicei, într-un **repository (depozit)** aflat pe un server accesibil tuturor dezvoltatorilor. Acest repository constituie mecanismul principal de sincronizare a surselor dezvoltatorilor.

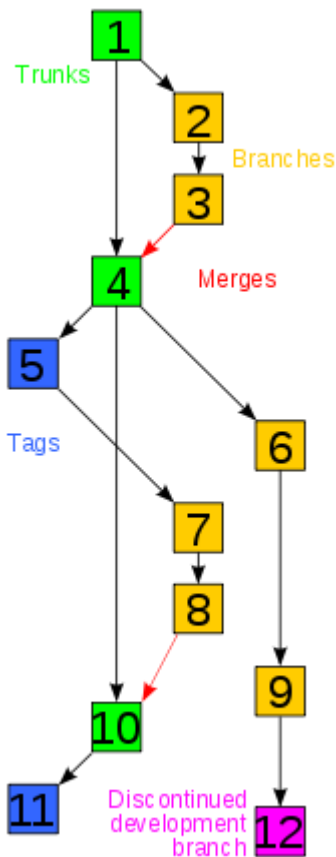


- Fiecare dezvoltator obține o copie a repository-ului, denumită **copie locală**. Copia rezidă pe sistemul dezvoltatorului. Dezvoltarea se va realiza în acest director.
- Există două operații care permit comunicația cu repository-ul:
 - **update** sau **pull** înseamnă actualizarea copiei locale cu informațiile din repository; este posibil ca alți dezvoltatori să fi comis schimbări în repository;
 - **commit** sau **push** înseamnă transmiterea/comiterea modificărilor locale în repository; ceilalți dezvoltatori vor folosi comanda **update** pentru actualizarea copiei locale cu acele informații.

Câteva cuvinte cheie utile în lucrul cu sisteme de control al versiunii sunt (denumirile sunt în engleză pentru că aceasta este forma uzuală de utilizare):

- **repository** locul (centralizat) în care se găsesc sursele și informațiile de modificare;
- **working copy** copia locală a repository-ului folosită de un dezvoltator;
- **check-out** operația de creare a unei copii locale pornind de la repository (clonarea repository-ului);
- **commit** (vezi mai sus)
- **update** (vezi mai sus)
- **branch** o ramură de dezvoltare (un fork) care poate fi dezvoltată în paralel față de ramura principală de dezvoltare (vezi [imaginea de mai jos](#));
- **trunk** ramura principală de dezvoltare (vezi [imaginea de mai jos](#));

- **tag** o referință la un snapshot al surselor la un moment dat în timp;
- **merge** o operație de unificare a două seturi de schimbări petrecute distinct;
- **conflict** situație în care două modificări sunt efectuate din surse diferite și sistemul nu poate găsi o soluție de unificare (merge) a acestor schimbări; dezvoltatorul va trebui să rezolve conflictul fie prin unificarea schimbărilor, fie prin selectarea unei singure variante



Există două tipuri de sisteme pentru gestiunea codului sursă:

- sisteme centralizate ([Subversion](#), [CVS](#), [perforce](#));
- sisteme distribuite ([Git](#), [Darcs](#), [Mercurial](#), [Bazaar](#)).

Detalii despre diferențele dintre acestea (mai degrabă între doi dintre cei mai cunoscuți reprezentanți, Subversion și Git) găsiți

[aici](#).

Git

[Git](#) este unul dintre cele mai folosite sisteme de versionare distribuite. Cele mai importante concepte din Git sunt următoarele:

- repository - există un repository remote și oricâte repository-uri locale
- commit - conținut modificat
- branch - lanț de commit-uri

La începutul dezvoltării unui proiect folosind Git sunt necesare câteva configurări:

- numele, adresa de e-mail, editorul utilizat pentru mesajele de commit

```
$ git config --global user.name "Andrei Maruseac"
$ git config --global user.email "andrei.maruseac@gmail.com"

$ git config --global core.editor "vim"
```

Dacă nu este folosit argumentul `-global`, configurările sunt făcute doar pentru repository-ul curent.

- tipuri de fișiere care nu vor fi comise prin crearea unui fișier `.gitignore`. Un exemplu de astfel de fișier este:

```
$ cat .gitignore
*~
*.swp
*.swo
*.o
*.obj
*.a
*.so
*.dll
*.lib
*.gz
*.bz2
*.zip
```

Fișiere `.gitignore` specifice limbajului de programare folosit puteți găsi pe [GitHub](https://github.com).
Comenzile Git folosite uzual în dezvoltarea proiectelor sunt:

- `man git-$comandă`, `man gittutorial` - pagini de manual pentru Git
- `git init` - inițializarea unui repository
- `git clone` - clonează un branch într-un director nou
- `git fetch` - preia commit-urile care nu există în repository-ul local
- `git merge` - integrează modificările din două branch-uri
- `git pull` - efectuează operațiile `fetch` și `merge` între branch-ul specificat și cel curent
- `git status` - arată ce fișiere au fost modificate de la ultimul commit și ce fișiere se află în staging area (se vor comite la comanda `git commit`)
- `git add` - adaugă fișiere în staging area
- `git commit` - comite modificările din staging area
- `git branch` - afișează toate branch-urile existente
- `git checkout` - dacă primește ca argument un branch, acesta devine branch-ul curent; dacă primește un nume de fișier ca parametru, se renunță la modificările de la ultimul commit din acest fișier
- `git push` - trimite modificările în repository-ul global
- `git log` - arată istoricul commit-urilor din branch-ul curent
- `git diff` - arată modificările față de un anumit commit sau între două commit-uri
- `git format-patch` - realizează patch-uri din ultimele commit-uri pentru a putea fi trimise prin e-mail
- `git am` - aplică patch-uri în branch-ul curent

Exerciții

Git (30 de minute)

Porniți în Linux și parcurgeți tutorialul [Git Immersion](#). Realizați primele 9 laboratoare din tutorial.

Dacă nu aveți instalat Git sau Ruby, din contul utilizatorului `root` instalați folosind `apt-get install git ruby rake`.

După realizarea tutorialului, creați un Repository al echipei. Repository-ul poate fi privat sau public, cum doriți, dar trebui să adăugați asistentul vostru la repository.

Lucru la proiect (60 de minute)

- **Obligatoriu** Creați un repository Git pe [GitHub](#). Repository-ul poate fi privat sau public, cum doriți, dar trebui să adăugați asistentul vostru la repository.
- Stabiliți task-urile din WBS pentru proiectul vostru. Realizați de asemenea o diagrama Gantt utilizând unul dintre tool-urile menționate (ex: [Grafice Gantt](#))
- **Optional** [Trello](#) pentru gestiunea task-urilor

În limita timpului parcurgeți și exercițiile de mai jos.

Opțional: GitHub (20 min)

Dacă aveți timp faceți și acest exercițiu.

Realizați următoarele tutoriale de pe GitHub: [Set up git](#), [Create A Repo](#) și primii 2 pași de aici: [Fork A Repo](#). Creați un branch nou, mutați-vă pe acesta, faceți o modificare, și comiteți modificarea. Realizați un patch cu modificarea respectivă, mutați-vă pe branch-ul inițial, și aplicați patch-ul. Trimiteți modificarea pe repository-ul central (fork-ul vostru de pe GitHub).

Laborator 4 - Utilitare pentru managementul și dezvoltarea proiectelor software

Aplicații web pentru MPS - GitHub

Aplicațiile web pentru managementul proiectelor software integrează mai multe aplicații folosite pentru a asigura o bună comunicare între membrii echipei și monitorizare a evoluției proiectului: wiki, source-code browser, tichete (buguri și solicitări de implementare), calendar (roadmap) etc. [Trac](#), [Redmine](#), [Google Code](#) și [GitHub](#) reprezintă cele mai cunoscute platforme ale acestei categorii de aplicații.

GitHub oferă următoarele funcționalități:

- crearea de organizații: de pe Select button-ul cu numele utilizatorului se selectează opțiunea "Create Organization"
- crearea de echipe: din interfața pentru organizație se accesează meniul "Teams" și se crează o echipă folosind butonul "New Team"
- crearea de proiecte publice sau private: din meniul din dreapta "Your repositories" se accesează butonul "New Repository" și se selectează tipul acestuia
- crearea unei pagini pe wiki: de pe pagina unui proiect se selectează din meniu "Wiki" și se apasă butonul "New Page"
- crearea de milestone-uri: de pe pagina unui proiect se selectează din meniu "Issues", se comută pe tab-ul "Milestones" și se apasă butonul "Create a new milestone"
- vizualizarea repository-ului și a commit-urilor
- realizarea de fork-uri: se apasă butonul "Fork"
- realizarea și acceptarea de pull request-uri
- crearea de branch-uri
- operația de "merge" între 2 branch-uri
- realizarea de code review
- bugtracking
- crearea și vizualizarea rapoartelor (issues)
- realizarea unor grafice despre evoluția proiectelor
- vizualizarea activităților din cadrul proiectului (creare de rapoarte (issues), știri, upload de document, commit-uri în repository, editări de wiki etc.)
- posibilitatea de a urmări evoluția altor proiecte: se apasă butonul "Watch" de pe pagina proiectului respectiv

Pentru a vă familiariza cu funcționalitățile sale de bază, accesați [pagina de ajutor](#) a platformei GitHub.

În caz de hazarde naturale, instrucțiunile sunt disponibile [aici](#).

Modelare - Netbeans, Eclipse

Netbeans și Eclipse reprezintă un set de IDE-uri cu funcționalități complexe ce îi ajută, în special, pe dezvoltatorii de Java. Una din funcționalitățile importante ale IDE-urilor o reprezintă posibilitatea creării de diagrame UML. În urma creării acestor diagrame se poate genera cod automat, reducând astfel efortul de dezvoltare a aplicației. Pentru C++/C# cel mai adesea se folosește Visual Studio.

Diagrame - MS Visio, Dia

În activitatea de proiectare se realizează numeroase grafice pentru a descrie componentele și modulele aplicației, legăturile între acestea și cazuri de utilizare.

Una dintre cele mai cunoscute aplicații pentru realizarea de diagrame este [Microsoft Visio](#). Alternativa lightweight a acestuia este [Dia](#). Alternativa din familia OpenOffice este [OpenOffice Draw](#). Toate aceste aplicații conțin un set de imagini/module predefinite (utilizatori, sisteme de calcul, elemente de rețelistică, forme predefinite de imagini de componente). Recomandăm folosirea acestor aplicații pentru realizarea diagramelor și graficelor în realizarea documentației pentru toate proiectele unde este necesar. Aplicațiile de mai sus oferă o interfață grafică user-friendly și permit exportarea într-un număr vast de formate.

Pentru modelarea arhitecturii (în special pentru SDD) amintim [StarUML](#) (cu suport pentru MacOSX, Linux, Windows) și [DrawIO](#), soluție online care poate exporta rezultatul atât ca imagine cât și ca pdf. Schema este salvată sub forma unui XML, ceea ce face posibilă salvarea și încărcarea cu ușurință a fișierelor de pe device-ul local direct în aplicația online și invers, pentru editare ulterioară. Alte aplicații: [Visual Paradigm](#) sau [Astar](#)

Limbaje de programare, biblioteci, framework-uri, coding style

În cadrul procesului de proiectare și dezvoltare a unui proiect software, trebuie să se aleagă limbajul/limbajele de programare și bibliotecile folosite, framework-urile utilizate și alte aplicații care vor înlesni procesul de dezvoltare, testare și mentenanță.

Una din ideile importante în alegerea unui limbaj sau biblioteci este că nu există un super-limbaj sau un super-framework care să rezolve toate problemele, indiferent de natura acestora. Domeniul de software engineering s-a dezvoltat tocmai pentru a găsi cele mai bune opțiuni și proceduri pentru o situație dată.

Alegerea unui limbaj, a unui framework sau a unor utilitare pentru dezvoltare/testare se realizează, în general, ținând cont de două aspecte:

- cât de potrivit este acel limbaj sau framework pentru proiectul dat;
- cât de acomodați sunt viitorii dezvoltatori cu acel limbaj/framework.

Folosirea unui limbaj/framework adecvat dar pe care utilizatorii nu-l cunosc și pentru care partea de training durează mult, poate produce frustrări în rândul viitorilor dezvoltatori și poate genera ineficiență și, în consecință, un produs necompetitiv.

Ca recomandare generală, limbajele de programare se pretează la anumite situații:

- aplicații low-level, care necesită eficiență, se vor realiza, de obicei în C
- aplicații care au nevoie de dezvoltare rapidă vor fi realizate, de obicei, într-un limbaj cu facilități object-oriented de forma Java sau Python
- aplicații middleware vor folosi biblioteci și API-uri specializate (de forma OpenMP, MPI, RMI, Corba) și limbaje specializate
- aplicațiile web vor folosi framework-uri și limbaje specializate (Ruby on Rails, PHP, JSP/Servlets)

Un alt aspect important în proiectarea și dezvoltarea unei aplicații îl constituie formatul fișierelor de configurare. Acestea pot fi:

- fișiere binare - ocupă puțin spațiu, sunt relativ ușor de parsat, dar nu sunt portabile
- fișiere text format propriu - pot fi editate de utilizator, nu reprezintă un format standard
- fișiere .ini - reprezintă un fișier text în format standard
- fișiere .xml - format standard răspândit cu dezavantajul că ocupă mult spațiu și parsarea fișierului consumă memorie

Indiferent de limbajul, bibliotecile și framework-urile folosite, pentru facilitarea parcurgerii, înțelegerii și îmbunătățirii codului se recomandă folosirea unui set de reguli, încadrate de

obicei, într-un document (de obicei fișier text) de tip "Coding Style" sau "Coding Standards". Exemple de astfel de documente sunt:

- [Linux Coding Style](#)
- [GNU Coding Standards](#)
- [Kernel Normal Form](#)
- [PEP 8 -- Style Guide for Python Code](#)
- [pear Coding Standards](#)
- [Code Conventions for the Java Programming Language](#)

În general, convențiile de codare diferă dar urmăresc aspecte comune:

- indentarea codului
- cod lizibil prin folosirea de spații și linii libere
- cod comentat și comentarii relevante
- nume relevante pentru variabile
- funcții cu dimensiune "umană" (nu kilometrice)
- consecvența stilului de codare (dacă se indentează codul cu TAB, se va indenta cu TAB peste tot; dacă se indentează cu 4 spații, se vor folosi 4 spații peste tot)

Documente de raportare

Folosirea unor documente de raportare periodice reprezintă o acțiune de bază în coordonarea și evaluarea activităților din cadrul unui proiect. Astfel de documente de raportare se vor referi la sarcinile planificate și realizate în perioada anterioară. Vor descrie, de asemenea, probleme apărute și soluții la acestea.

Documentele de raportare pot fi furnizate la nivel de săptămâna, lună, semestru sau an, funcție de care variază dimensiunea documentului și detaliile cuprinse. Exemple de șabloane de documente de raportare săptămânală se găsesc [aici](#), [aici](#) și [aici](#). Documentul de raportare săptămânală este realizat de project manager prin analiza activităților planificate/efecuate în cadrul săptămânii anterioare.

Documentele de raportare a activității au o structură tabelară în care sunt descrise principalele activități care s-au desfășurat pe parcursul perioadei raportate. Fiecare activitate este descrisă printr-un nume, data planificată pentru încheiere, responsabilul/responsabilii activității, starea curentă (în lucru, suspendată, dependentă de un rezultat, încheiată) și procentul realizat. Exemple de șabloane de documente de raportare a activității se găsesc [aici](#) și [aici](#). Documentul de raportare a activității este completat de project manager pe baza informațiilor furnizate de membrii echipei și responsabilii fiecărei activități.

Evaluarea membrilor echipei este importantă în cadrul unui proiect. Evaluarea are rolul de a determina gradul de implicare și competență a membrilor echipei și de a lua decizii de actualizare a distribuției sarcinilor pentru eficientizarea realizării acestora. Un exemplu de formular de implicare a membrilor echipei este [acesta](#). Evaluarea este realizată de project manager.

Exerciții

Lucru la organizarea proiectului (50 de minute)

Pentru un tutorial de GitHub, parcurgeți [ghidul aferent](#).

- **Creat infrastructură de prezentare pe GitHub**
 - fișier **readme** cu echipa (Nume, prenume, grupă, (opt.) link către CV);
 - pagină de **wiki** asociată proiectului;

- prezentarea generală a proiectului. Evitați repetarea cerinței. Link către cerință;
- Rolurile membrilor echipei (dezvoltatori, tester, team lead/manager, technical writer, analist, consultant);
- upload la imaginea arhitecturală;
- resurse folosite;
- milestone-uri;
- Stabilit și documentat în GitHub:
 - limbaj/limbaje de programare folosite;
 - framework-uri, biblioteci;
- Creat infrastructură de bază în repository
 - directoare pentru surse, documentație, branch-uri, subdirectoare ale acestora;
 - creat fișier **text** de tipul **CodingStyle**; un fișier pentru fiecare limbaj; poate fi inspirat din fișiere similare din proiecte mari;
 - fișierul de CodingStyle poate apărea pe wiki;
- Ca în cadrul oricărui laborator, va debuta cu un scrum meeting (prezentarea de 1 minut a activităților realizate de fiecare membru al echipei) (opțional).
- Detaliile tehnice stabilite pe github vor fi adăugate și în SDD.

Pe perioada dezvoltării proiectului, asistentul de laborator va avea acces la repository-ul Git al echipei. La fel, se va oferi acces la directorul Google Drive (sau wiki-ul sau forma colaborativă folosită pentru structurarea informațiilor interne echipei: documente, evenimente, minute de întâlniri, responsabilități, sarcini).

Responsabilități în echipă (15 minute)

Fiecare membru al echipei scrie pe o foaie mică (A6) două elemente: ce rol are în cadrul echipei și ce tehnologie preferă. Apoi asistentul spune cele două elemente iar membrii echipei spun despre cine era vorba.

Exercitiu (15 minute)

Stabiliți recenzori pentru pull request-uri. Va exista un recenzor pentru fiecare pull request (poate fi unul singur, pot fi mai mulți). Acesta va comenta codul comis și, dacă totul e în regulă, îl va accepta.

Oricine are permisiuni de push în repository dar e recomandată urmărirea [workflow-ului GitHub](#) și folosite pull request-uri (sau merge request-uri pe [GitLab](#)) pentru faza de recenzie. Este o formă de [code review](#) și o bună practică în dezvoltarea aplicațiilor.

De făcut acasă

Continuați lucrul la proiect. Urmăriți:

1. Să existe cineva care are un rol de a monitoriza activitățile și respectarea termenelor.
2. Să fiți pe aceeași pagină, să știți unde vreți să ajungeți.
3. Să redactați agenda fiecărei întâlniri.
4. Să știe fiecare ce are de făcut. Să discutați și să agreeți acțiuni, responsabilități și termene.
5. Să recenzați codul scris de alții.
6. Să comunicați intern dacă întâmpinați probleme sau dacă nu vă puteți îndeplini la timp atribuțiile.

Laborator 5 - Comunicarea în cadrul proiectului

Comunicarea

Ca în orice domeniu, comunicarea poate să aibă loc, fie pe cale verbală, fie pe cale scrisă.

Comunicarea verbală este interactivă, rapidă, dar nu ajunge la nivelul de detaliere posibil prin comunicarea scrisă. Comunicarea verbală transmite sensuri și simboluri mai curând decât informație exactă. De exemplu: dacă toată lumea este de acord ca la forma "patrat" să îi zică cerc atunci acesta este sensul general acceptat de către toți. Capcana tipică este de a crede că toată lumea a înțeles același lucru. Detalii esențiale pot dispărea rapid din memorie. Fiecare persoană umple acele goluri existente într-un mesaj cu elemente proprii din alte experiențe. Comunicarea trebuie să fie exactă, specifică fără generalizări, omisiuni, ambiguități. Tipic, o persoană poate reține circa 50% din ce s-a discutat în primele 20 de minute, iar după aceea procentul scade rapid (<http://www.mtholyoke.edu/acad/intrel/speech/differences.htm>). Comunicarea verbală este utilă în special în situații decizionale, când interlocutorii sunt deja familiari cu subiectul, și este recomandat să fie urmată de un rezumat scris.

Comunicarea scrisă este mult mai susceptibilă la erori de emisie și/sau recepție. Problemele pot apărea în funcție de formularea aleasă și de interpretarea dată de cititor. În lipsa posibilității de interacțiune în vederea lămuririi unui aspect, confuziile și erorile de comunicare pot apărea ușor. (vezi interpretări ale legilor). Ca avantaje, informația poate fi foarte detaliată, exactă și mai ales e stabilă în timp. Comunicarea scrisă este utilă pentru transmiterea efectivă de informație și fixarea informației schimbate în comunicările verbale.

Comunicarea verbală

Următoarele reguli sunt utile în cadrul comunicării verbale:

- **Nu întrerupeți pe cel care vorbește la un moment dat.**
 - Există 2 posibilități: aveți ceva valoros de zis sau doar o observație care nu ar aduce valoare discuției. În ambele cazuri, în afară de lipsa de politețe, și potențialele tensiuni create din partea persoanei respective, o să aduceți și un minus în termen de eficiență a discuției.
 - Soluția recomandată, pentru a nu întrerupe discuția, dar și a păstra ideile bune care v-au venit, este să aveți un carnetel / laptop în care să notați aceste idei, pe care le puteți expune când vorbitorul a terminat ce avea de spus.
- **Nu vorbiți mai mult de 5 minute fără pauză.**
- **Separați clar ideile distincte.**
 - Spre deosebire de un material scris, unde ideile distincte pot fi separate ușor în mod vizual, folosind diverși despărțitori, în comunicarea verbală, trecerea bruscă de la un subiect la altul poate reprezenta o problemă.
- **Nu stați cu spatele la cei cărora vă adresați.**
- În cadrul grupului de lucru puteți stabili un set de **reguli suplimentare de comunicare**.
 - O bună practică este să existe un set de semne stabilit pentru a facilita transmiterea de mesaje vorbitorului fără o întrerupere verbală. Exemple de mesaje pentru care pot fi transmise folosind anumite semne:
 - off-topic - pentru a avertiza vorbitorul că a divagat de la subiect;
 - scurtează - dacă vorbitorul discută prea mult pe o singură temă;
 - vreau să iau cuvântul - pentru a cere o întrerupere, pentru a aduce un comentariu important;

- subiect închis - dacă se consideră că subiectul este clar și se poate trece la următorul punct în discuție.
- **Fiti asertivi.**
- În cadrul discuției nu încercați să vă impuneți punctul de vedere și ascultați și părerile celorlalți încercând să înțelegeți argumentele lor.

Comunicarea în scris

Deși se preferă, în general, comunicarea verbală în cadrul dezvoltării unui proiect software (în special în cadrul metodologiilor Agile), există numeroase situații în care este necesară transmiterea unor informații în formă scrisă.

Întotdeauna când comunicați, trebuie să aveți în vedere respectul pentru timpul celorlalți. Acest lucru înseamnă că trebuie să comunicați ce aveți de transmis scurt și clar.

Înainte de a trimite un mesaj, fie că este vorba despre un e-mail sau despre un document cu un raport, puneți-vă următoarele întrebări:

- Ce vreau să transmit?
- Cui vreau să transmit? (persoanele de la HR probabil nu vor fi interesate de ce bug ați descoperit în kernel)
- Ce rezultat / ce răspuns aștept? (dacă PM-ul trimite un mail în care anunță că există un mare bug în aplicație, dar nu numește un responsabil / un deadline / nu propune o soluție, mesajul nu va fi eficient).

Liste de discuții & e-mailuri

Reguli pentru listele de discuții:

- Folosiți subiecte clare, nu subiecte de forma "Urgent", "Important", "Ceva", "Pentru mâine", "Reply". Nu transmiteți mesaje fără subiect.
- Folosiți etichete la început de subiect pentru a încadra domeniul mesajului: (Exemplu: [Testare] / [Debug] / [Intalnire Joi])
- Evitați subiecte de tipul "Re: Re: Re: Re: Re: [...]".
 - Exemplu de Subject prost: "Intalnire"
 - Exemplu de Subject bun: "[Intalnire] Intrevedere luni, 15 noiembrie 2010, 16:00 pentru stabilire plan de testare"
- Scrieți mesaje scurte și la obiect; nu trimiteți mesaje gigant (nu le va citi nimeni) sau diluate (se va căuta ideea dorită în cadrul paragrafelor).
- Dacă sunt mai multe idei într-un e-mail, separați-le vizual cât mai bine (prin paragrafe noi, prin gruparea în liste).
- Dacă vreți să transmiteți două topic-uri diferite, trimiteți două mesaje.
- Dacă aveți un mail mai lung, puteți scrie un rezumat în prima parte, pentru ca cel care citește să înțeleagă contextul și ce urmează să citească.
- Dacă un e-mail are mai mult de 500 de cuvinte, cel mai probabil era nevoie de un fișier.
- Dacă răspundeți la un mail dintr-un lung șir de răspunsuri, ștergeți din când în când conținutul mesajelor vechi, pentru a nu ajunge la un mail de 2000 de rânduri din care doar 20 mai sunt de actualitate.
- Dacă răspundeți la un mail cu mai multe puncte, răspundeți între acele puncte.
- Dacă aveți link-uri lungi de inclus în mail, le puteți indexa cu [1] [2] [3] și include la sfârșitul mailului, pentru a nu intercala în mesaj linkuri lungi.
- Nu folosiți ALL CAPS. ALL CAPS denotă un ton răstit, și nimeni nu apreciază asta.
- Evitați transmiterea de atașamente pe o listă. Încarcă inutil căsuța poștală a tuturor membrilor. Publicați-le pe un site și trimiteți link-ul aferent în e-mail.

- Dacă un mesaj se adresează numai unei minorități a membrilor unei liste de discuție, trimiteți mesajul privat acelor membri. Nu trimiteți un mesaj unor persoane neinteresate de subiect (nu trimiteți spam 😊)

Change logs

Este recomandat să existe o evidență a derulării unui proiect. Deși în general, există o motivație scăzută pentru evidența schimbărilor, este util să existe un set de fișiere de tip jurnal/log care să conțină date despre:

- bug-fix-uri;
- modificări efectuate;
- decizii de proiectare/alegere de componente.

Majoritatea aplicațiilor de (software) project management (inclusiv GitHub) conțin facilități de înregistrare a activității utilizatorilor, precum: adăugarea unui issue, rezolvarea unei probleme, posting-ul unui mesaj, editarea unei pagini wiki.

Comentarii cod

Există două moduri de a comenta cod: un mod "asistat" și unul "liber".

Modul de comentare "asistat" este acela în care programatorul urmărește o sintaxă ce poate fi citită și interpretată de un sistem automat de generare de documentație.

- Pentru Java, un astfel de sistem de documentare a codului este <http://java.sun.com/j2se/javadoc/>
- Pentru .NET există o soluție similară: <http://msdn.microsoft.com/en-us/library/b2s063f7.aspx>
- Acest mod de comentare are totuși limitările sale, întrucât este conceput mai mult pentru a documenta clasele, interfețele, metodele, legăturile dintre acestea, parametrii care se transmit, în general informații care nu explică implementarea.

Pentru documentarea implementării, modul "liber", nu există un sistem clar definit, dar următoarele reguli sunt, în general, bine de urmărit:

- Când vrei să explici o linie, puneți comentariile înaintea liniei respective, nu în dreapta

- *// un select simplu pentru a prelua din BD toate campurile din tabelul primit ca parametru*
- `ResultSet rs = st.executeQuery("select * from " + tableName);`

```
ResultSetMetaData metadata = rs.getMetaData();
```

- Dacă o bucată de cod este incompletă/neoptimizată și există șanse ca ulterior să se mai lucreze la ea, folosiți cuvinte cheie:
 - **BUG/FIX:** Pentru a marca un bug care nu a fost încă eliminat, sau un fix. De multe ori un fix introduce bug-uri noi, astfel este bine să documentați locurile în care ați făcut modificări și ați schimbat comportamentul programului.
 - **TODO:** Puteți folosi acest cuvânt cheie pentru a marca un punct din program unde se pot aduce îmbunătățiri/ unde mai trebuie completat ceva. Există medii de dezvoltare care pot chiar recunoaște astfel de cuvinte cheie și pot crea o listă de TODO-uri.

```
// TODO: check data input
```


- Comentați și explicați cât mai clar; nu mergeți pe ideea "lasă că știu eu ce am vrut să fac aici".
- Nu exagerați cu diverse comentarii inutile. Exemplu:

```
▪ // increment i
```

```
i++;
```

- La [această adresă](#) găsiți sfaturi legate de comentarea corespunzătoare a codului.

Pentru amuzament, consultați pagina [What is the best comment in source code you have ever encountered?](#)

Întâlnirile

Întâlnirile reprezintă un mod de comunicare vital pentru realizarea unui proiect, fiind folosite în special pentru a lua decizii în anumite situații, pentru a găsi soluții și pentru a analiza progresul actual al proiectului precum și planul de lucru pentru perioada următoare.

Ca regulă generală, pentru ca o întâlnire să fie eficientă, aceasta trebuie să fie cât mai scurtă și să implice doar oamenii direct interesați / în cunoștință de cauză. De exemplu, în metodologia SCRUM (metodologie de tip Agile), se recomandă întâlniri cu timp fix de 15 minute. Nu se permite depășirea timpului. Astfel, participanții au fiecare un timp limitat pentru a își expune ideile, și sunt astfel obligați să își structureze și rezume ideile înainte de întâlnire.

Întâlnirile SCRUM au doar caracter informativ. Astfel, fiecare participant prezintă statusul acțiunilor de care este responsabil fără să se dezbată eventualele probleme. Problemele ce nu pot fi rezolvate la sursă se scalează către superior.

Programarea întâlnirilor

Planificarea întâlnirilor trebuie să fie efectuată de comun acord, pentru a nu interfera cu programul celorlalți / pentru a nu apărea suprapuneri. [Doodle](#) este unul din cele mai cunoscute site-uri create pentru acest scop.

Se recomandă, de asemenea, folosirea aplicațiilor de tip calendar pentru stabilirea întâlnirilor și pentru notificarea participanților. Există aplicații Desktop precum Microsoft Outlook, Evolution, Mozilla Sunbird sau aplicații web precum Google Calendar. Mai multe informații despre aplicații de calendar sunt prezentate pe [Wikipedia](#).

Planificarea și organizarea unei întâlniri

Pentru ca o întâlnire să fie eficientă, trebuie ca cel care organizează întâlnirea să înțeleagă răspunsul la următoarele întrebări, în funcție de care să iau deciziile aferente. *Care este scopul întâlnirii?* - Este important ca o întâlnire să aibă la bază un scop precis. *Care este rezultatul scontat în urma întâlnirii și cum poate fi el cuantificat?* - Este bine să existe un target clar stabilit, sub forma unui output ce trebuie să rezulte din întâlnire. La sfârșitul întâlnirii, situația proiectului trebuie să fie diferită decât la începutul întâlnirii: o decizie luată, un set de probleme clarificate, un impas depășit, etc. *Cine trebuie să participe la întâlnire?* - Este vital pentru succesul unei întâlniri ca la aceasta să participe persoanele interesante / persoanele care au răspunsuri / persoanele care cunosc proiectul/aspectul care va fi discutat.

De asemenea, atât din respect pentru timpul celor implicați, dar și pentru a fi o întâlnire eficientă, este recomandat ca înainte de întâlnire, invitații să fie informați de:

- scopul întâlnirii
- aspectele ce vor fi discutate
- rezultatele așteptate

- durata estimată a întâlnirii

În plus, organizatorul întâlnirii ar trebui să:

- planifice o locație adecvată
- creeze o agendă de discuție

Înainte de începerea discuțiilor:

- O persoană trebuie să fie desemnată să noteze **o minută** cu cele discutate.
- Coordonatorul întâlnirii va enumera pe scurt ce urmează a fi discutat.

Pe parcursul întâlnirii:

- Coordonatorul întâlnirii trebuie să încurajeze discuția.
- Pentru a fi eficientă, discuția trebuie să fie axată pe argumente.
- Coordonatorul este cel ce va "arbitra discuția".
- El trebuie să aibă și rol de "focus-man", și anume să aibă grijă ca discuția să se concentreze pe temele importante, să nu se divage de la scop.
- Dacă se ajunge la un impas din lipsă de informații, se poate programa o întâlnire ulterioară, până la care anumiți membri ai echipei vor avea ca task obținerea informațiilor necesare.

La sfârșitul întâlnirii:

- Se trece prin agenda întâlnirii, pentru a verifica dacă au fost acoperite toate aspectele.
- Se trag concluziile.
- Minuta redactată pe parcursul întâlnirii este distribuită factorilor de interes / echipei.

Se recomandă existența unui document sau pagini wiki care să fie completată înainte, în timpul și după întâlnire. O posibilă structură a acestei pagini este:

```
* Topic întâlnire

* Dată/timp întâlnire

===== Agendă =====

-- se completează la planificarea întâlnirii

* item1

* item2

===== Participanți =====

-- se completează înainte de începerea discuțiilor

* participant1

* participant2
```

```
===== Discuții =====
```

```
-- minuta întâlnirii pe parcursul discuției
```

```
* Inițiale1: ...
```

```
* Inițiale2: ...
```

```
===== Decizii/concluzii =====
```

```
-- imediat după încheierea discuției
```

```
* ...
```

```
* ...
```

Link-ul la acea pagină sau documentul aferent este transmis participanților după încheierea întrevederii.

Brainstorming

Brainstorming este o tehnică de grup creativă proiectată pentru a genera un număr mare de idei pentru soluția unei probleme.

Un set de utilitare precum [FreeMind](#) și [Mind Manager](#) sunt folosite pentru brainstorming/mind-mapping.

Decizii

În orice proiect există momente de impas în care o decizie trebuie luată, dar există mai multe opțiuni, și membrii echipei au păreri distincte.

În aceste situații, cel mai important lucru de avut în vedere este faptul că în multe situații, o decizie greșită poate fi mai puțin dăunătoare decât un impas pe termen lung. Cu alte cuvinte, leader-ul trebuie să își asume uneori o decizie pentru a scoate echipa din impas. Chiar dacă decizia este greșită, o decizie luată în timp util poate fi corectată în momentul în care lucrurile merg prost; pe de altă parte, dacă echipa rămâne în impas, nu se ajunge nicăieri și proiectul nu înaintază. Mai rău, se poate ajunge în situația în care o decizie este impusă în momentul în care echipa nu mai are opțiuni din lipsă de timp.

Decizia în grup

Există mai multe metode prin care un grup poate să ajungă la o decizie, fiecare având avantajele și dezavantajele ei, în funcție de situație:

* **Competiția ideilor** - fiecare opțiune este combătută până când toată lumea ajunge la aceeași părere * **Compromisul** - este adoptată o idee de mijloc, care echilibrează avantajele și dezavantajele * **Votul** - utilă mai ales pentru decizii obiective (Exemplu: decizii de design)

* **Impunerea** - utilă în caz de impas. Dacă proiectul stagnează, conducătorul trebuie să își asume decizia pentru a merge mai departe * **Încrederea** în persoana care cunoaște cel mai bine domeniul / are cea mai multă **experiență**

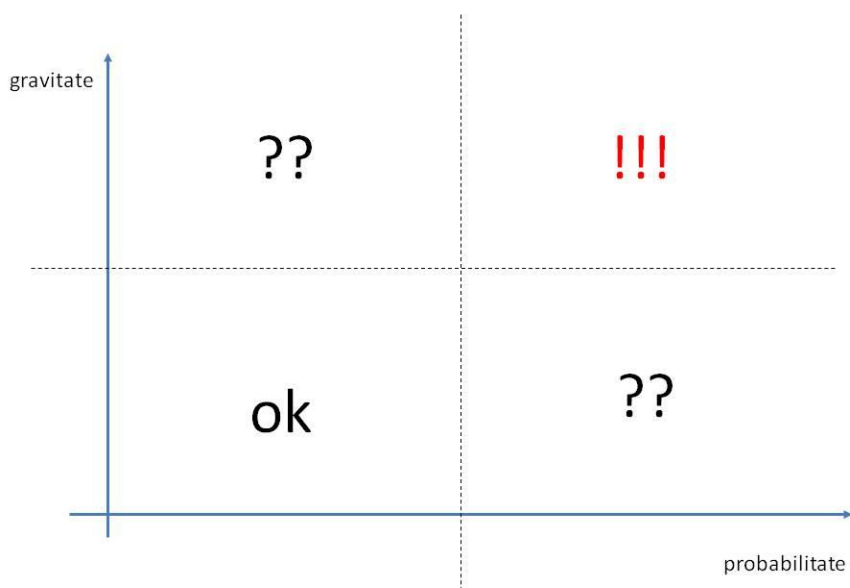
Riscul

Uneori, o decizie greu de luat depinde de evaluarea unui risc. În cadrul unui proiect care "merge în necunoscut" apar o serie de riscuri. Unele variante prezintă anumite riscuri, alte variante de decizie prezintă alte riscuri. Problema este cum decidem ce variantă să alegem, pentru a minimiza riscul.

Riscul are în general 2 aspecte:

- probabilitatea - Care sunt șansele să aibă loc evenimentul?
- gravitatea - Cât de grav este dacă are loc?

O metodă general recomandată de analiză a riscului este de a alocă fiecărui risc identificat 2 note, una pentru probabilitate și una pentru gravitate. Riscul poate fi analizat fie grafic, fie ca o sumă a celor 2 note. Dacă folosim o scară de la 1 la 10, și un eveniment primește nota 16, în mod cert acesta trebuie considerat. În cazul în care se cunosc probabilitatea riscului și impactul pe care îl are acesta în cadrul proiectului (număr de zile sau costuri), riscul va fi calculat ca produsul dintre probabilitatea de apariție a riscului și efectul produs.



CheckList

- Cât de importantă este decizia? Care sunt consecințele deciziei? Riscurile?
- Cât de clară este imaginea pe care o avem acum asupra problemei? Avem toate datele problemei?
- Au fost cercetate toate opțiunile?
- Au fost implicați / consultați membrii echipei cu cea mai mare expertiză în problema de față?

Rezolvarea conflictelor

Deși, în general, se dorește evitarea conflictelor, uneori pot apărea divergențe între membrii unei echipe. Pentru bunul mers al lucrurilor, este recomandat ca aceste divergențe să fie discutate și aplanate cât mai din timp. Orice problemă de tip conflictual neabordată la timpul

ei poate escalada și va avea tendința de a se manifesta în momentele critice ale proiectului, când un conflict este ultimul lucru de care are nevoie echipa.

În cazul în care apar divergențe între doi membri ai echipei, este responsabilitatea team leader-ului/PM-ului, eventual cu ajutor din partea unei persoane de la HR să aplaneze conflictul înainte ca acesta să afecteze echipa.

În cazul unui conflict manager - subordonat, lucrurile se complică, și un factor determinant în rezolvarea unui astfel de conflict este dat de maturitatea managerului.

Există mai multe modalități de abordarea a unui conflict

Abordarea directă/dictatorială

Deși este cea mai "incomodă" poate fi și cea mai eficientă. Presupune determinare din partea liderului și este necesar ca orice critică adusă în cadrul confruntării să fie constructivă. Această abordare se bazează pe un stil de abordare logic, ingineresc, de rezolvare a unei probleme. Problema este expusă direct, sincer, și liderul încearcă să prezinte soluția logică. Dezavantajul metodei este dat de nivelul de sinceritate și de exprimarea directă necesară, care uneori nu va fi privită bine de toți cei implicați.

Negociere

Avantajul metodei este dat de faptul ca teoretic, prin negociere se poate ajunge la un compromis acceptat de toți factorii implicați. În realitate, de multe ori la final toți cei implicați pot ajunge nemulțumiți de ce au trebuit să cedeze.

Aplicarea regulilor

Sunt cazuri în care un membru al echipei generează un conflict prin nerespectarea regulilor / standardelor. Uneori, nu este oportună regenerarea / schimbarea regulilor, și deși cu un cost mare de antipatie din partea liderului, trebuie impuse regulile echipei / companiei.

Ocolire

În cazul în care miza este prea mică / nu este ceva urgent, liderul poate alege fie să amâne rezolvarea, fie să accepte o soluție chiar dacă aceasta nu este optimă. Deși această abordare are dezavantaje evidente, este foarte utilă totuși în momentul în care echipa stagnează blocată pe o discuție pe un detaliu nesemnificativ.

Empatizarea

Uneori, oamenii pot avea o idee similară, dar sunt blocați într-o discuție contradictorie pe detalii. În astfel de situații, liderul este cel care poate sublinia similitudinile între soluțiile propuse și le poate exploata pentru a sublinia că ideile nu sunt atât de diferite și, în final, poate negocia un consens.

Exerciții

Întâlnire de proiect (15 minute)

Realizați o întrevedere în cadrul echipei în care discutați despre starea curentă a proiectului. Urmăriți recomandările pentru întâlnire. Aveți în vedere:

- Să țineți o minută a întâlnirii.
- Să stabiliți care au fost concluziile întâlnirii legate de starea proiectului.
- Să stabiliți care au fost deciziile pentru viitor.
- Să stabiliți care sunt sarcinile și responsabilitățile din timpul întâlnirii.

Publicați minutele întâlnirii într-un document Google sau pe o pagină de wiki și transmiteți-le asistentului. Cereți feedback asistentului legat de desfășurarea proiectului. Evaluați-vă și apoi cereți și asistentului evaluarea legat de încadrarea în timp, cât mai e până la termen, cât mai aveți în desfășurarea proiectului.

Să aveți în vedere o astfel de întâlnire la începutul fiecărui interval de laborator.

Comentarii în cod (10 minute)

Analizați următoarele situații în care sunt adăugate comentarii în cod. Care dintre ele trebuie evitate? Cum le-ați îmbunătăți?

```
int d; // days until deadline
// if the student is at least 18 years of age
if (student.Age >= 18)
{
    // send meeting invitation to the student
    notificationService.SendMessageTo(student, meetingInvitation);
}
else // if the student is younger than 18 years
{
    // sends a meeting invitation to the student's legal guardian
    notificationService.SendMessageTo(student.Parent, meetingInvitation);
}
// student is eligible for blood donation
if (student.Age >= 17 && student.Weight >= 58.0 && student.Height >= 1.55)
{
    ScheduleBloodDonatingSessionWith(student);
}
```

Organizare de proiect (30 de minute)

- Discutați intern în cadrul echipei de proiect și decideți ce veți documenta în cadrul proiectului, unde veți realiza acest lucru (wiki, pagină web, Google Doc) și cine se va ocupa din proiect de realizarea documentației.
- Discutați pe marginea organizării proiectului. Urmăriți crearea de activități, sarcini, cu responsabilități și timeline în cadrul proiectului (pana in acest moment, ar trebui sa aveti deja un plan al acestora). Pentru aceasta puteți folosi un document spreadsheet, puteți folosi issue tracker pe GitHub sau puteți folosi soluții de realizarea de diagrame de tip WBS (Work Breakdown Structure).

Lucru la proiect (30 de minute)

Lucrați la proiect în conformitate cu planificarea realizată, urmărind sarcinile individuale. Puteți face brainstorming, puteți lua decizii, puteți scrie documentație, puteți scrie cod, puteți face code-review, puteți stabili noi sarcini (management), puteți face planificări, puteți face pair programming. Orice vă este util și important în buna desfășurare a proiectului.

Introducere

Premisele unui **proiect de succes** din perspectiva contribuției **PM**-ului sunt:

- *estimările corecte* ale duratelor și resurselor necesare îndeplinirii sarcinilor (planificări potrivite)
- *monitorizarea și evaluarea* consecventă a proiectului având drept ghid planul proiectului (Obiectivitate, Informare, Prevedere)
- *controlul* permanent al evoluției proiectului (Direcția și sensul către succes)
- *comunicare* clară și motivantă
- *coeziune* cu echipa.

Monitorizarea, evaluarea și controlul proiectului alcătuiesc **instrumentul de bază** utilizat de **PM** în mod **iterativ** pentru a asigura **evoluția** proiectului în consonanță cu planurile de **succes**.

Monitorizarea proiectului (5 min)

- este realizată de PM
- cuprinde activități precum:
 1. **colectarea de informații** privind evoluția proiectului și a factorilor de influență (echipă, context)
 2. **diseminarea** concluziilor evaluării și a schimbărilor decise către echipă.

Ce monitorizează PM-ul (colectează informații despre):

- statusul proiectului în raport cu planul proiectului
- costurile înregistrate în raport cu planul
- evoluția factorilor de risc
- evoluția contextului proiectului
- cantitatea și calitatea muncii îndeplinite de fiecare membru al echipei
- coeziunea și comunicarea în cadrul echipei.

Cum monitorizează PM-ul:

- prin întâlniri de proiect
- cu ajutorul tool-urilor ajutătoare de Software Project Management (Redmine, Trac): sistem de Ticketing, News, forumuri
- cu ajutorul listelor de discuție
- cu ajutorul rapoartelor de testare
- prin comunicare directă cu membrii echipei.

În urma monitorizării, PM-ul deține informațiile necesare evaluării proiectului, urmând ca după evaluarea propriu-zisă, să facă publice echipei concluziile evaluării și deciziile luate.

Recomandări pentru etapa de diseminare a concluziilor evaluării și a deciziilor:

1. PM-ul trebuie să evite să facă comparații între membrii echipei sau să propună ca modele sau anti-modele diverse persoane. Acest lucru conduce la distrugerea coeziunii echipei și la generarea de tensiuni, invidii, antipatii în cadrul echipei (NU "*de ce nu poți fi ca ...*", "*X este cel mai grozav / slab*")
2. PM-ul trebuie să dea mereu ca model sau anti-model tipul generic de membru al echipei, conform standardelor de lucru
3. PM-ul trebuie să trateze toți membrii implicați în proiect obiectiv, egal și nediscriminatoriu – altfel, membrii echipei se pot simți nerespectați, defavorizați, nedoriți

4. se recomandă ca PM-ul să acorde feedback constant membrilor echipei pentru ca aceștia să fie conștienți de situație, să poată acționa rapid fiind în cunoștință de cauză și să nu fie tensionați la auditurile periodice

Important: Monitorizarea presupune colectarea periodică de informații astfel încât PM-ul să fie la curent cu starea proiectului și a echipei. Monitorizarea nu presupune urmărirea permanentă și insistentă a activităților; acest lucru poate deranja echipa și transforma PM-ul într-un "spion" sau o "moară stricăță". O monitorizare bună presupune colectarea de informații cu deranjare cât mai mică a echipei, cât mai puțin intruzivă; echipa are ca motivație să lucreze la proiect nu să răspundă la toate întrebările de detaliu ale PM-ului.

Evaluarea proiectului

- este realizată în principal de *PM*. Contribuie membrii echipei, managementul superior, clienții și subcontractorii
- are loc după etapa de colectare de informații a monitorizării
- evaluarea proiectului se referă la:
 1. **determinarea statusului proiectului** în raport cu planul proiectului (respectarea deadline-urilor, calitatea proiectului și încadrarea în planul de buget)
 2. **evaluarea echipei.**

Planul proiectului cuprinde următoarele:

1. *WBS*-ul detaliat al proiectului (asocierea activități – resurse - durate)
2. planificarea temporală a activităților din *WBS* (*diagrama Gantt*)
3. *planul de risc* al proiectului (ce riscuri sunt, cum pot fi preîntâmpinate, planuri de acțiune în cazul apariției lor).

Determinarea statusului proiectului (5 min)

Pentru a evalua obiectiv și acurat statusul proiectului, se folosește **metrica SPI**.

SPI (Schedule Performance Index):

- este o măsură a eficienței evoluției proiectului
- se calculează ca raport între **valoarea obținută** (EV – earned value) și **valoarea planificată** (PV – planned value): **$SPI = EV / PV$**
 - EV răspunde la întrebarea "Cât de mult s-a realizat până în acest punct din ce s-a planificat?"
 - PV răspunde la întrebarea "Cât de mult s-a planificat pentru a se realiza până în acest punct?"

Interpretarea SPI-ului se face astfel:

- dacă SPI are o valoare *egală cu 1*, atunci starea proiectului este *conform planului* ⇒ proiectul are șanse favorabile de a se încheia cu succes
- dacă SPI are o valoare *supraunitară*, atunci starea proiectului este *înfloritoare*, mai bună chiar decât cea planificată ⇒ proiectul are șanse mari de a se încheia cu succes. În acest caz, este nevoie de analiza situației, determinarea factorilor de succes, stimularea și diseminarea lor și actualizarea planului proiectului.
- dacă SPI are o valoare *subunitară*, atunci proiectul este *în urmă față de planificări* ⇒ proiectul poate deveni un eșec dacă nu se acționează corectiv prompt. În acest caz, este nevoie de analiza situației, determinarea factorilor de insucces, a acțiunilor corective și actualizarea planului proiectului.

Exemplu de calcul SPI

Pentru proiectul de dezvoltare a unor biblioteci Java ce oferă servicii auxiliare față de cele oferite de Sun, pentru etapa dezvoltării bibliotecii de servicii auxiliare de email, s-au planificat activitățile:

- proiectare bibliotecă 100% 4
- dezvoltare bibliotecă 50% 4
- testare servicii oferite de bibliotecă 30% 1
- redactare rapoarte de testare 30% 1.

(activitățile au notate alături informațiile: procentul cât s-a realizat până la deadline și o pondere egală cu efortul necesar de împlinire a activității raportat la efortul total).

În acest caz, calculul SPI este defapt calculul unei medii ponderate:

$$SPI = (4 \cdot 100 + 4 \cdot 50 + 1 \cdot 30 + 1 \cdot 30) / (10 \cdot 100) = 0.66 \Rightarrow \text{proiectul este în întârziere.}$$

Întârzieri ale proiectului (5 min)

SPI-ul poate avea **valori subunitare** în următoarele cazuri:

- *estimările* PM-ului de timp și resurse necesare au fost *departe de realitate*
- *echipa nu și-a îndeplinit sarcinile* planificate (când un membru al echipei conștientizează faptul că nu poate termina la timp sarcina atribuită, are datoria de a anunța imediat PM-ul pentru ca acesta să poată încerca evitarea întârzierii prin plasarea de noi resurse sau prin alte metode)
- a intervenit un *context nefavorabil* (poate s-au dezlănțuit factori de risc ori s-au creat alte circumstanțe nefavorabile proiectului).

Premisele unei estimări greșite sunt următoarele:

- neînțelegerea scopului propus
- necunoașterea activităților de desfășurat pentru a împlini sarcina
- estimarea pe baza unei alte păreri (îndoielnice)
- estimarea necoordonată între activități
- subestimarea următoarelor probleme:
 - timpul de pregătire / învățare
 - timpul consumat pentru comunicare
 - întâlnirile de proiect
 - control
- Optimism nefondat - nimic nu va merge prost; totul va fi conform planului
- Ignorarea realității – totul merge bine; nimic nu se va schimba în cursul proiectului.

Do you know what Murphy's Law is?

"If it can go wrong it will".

Do you know what Sod's Law is?

"It will go wrong in the worst possible way".

Recomandări pentru estimări corecte:

- pentru a estima cât mai corect timpul și numărul de resurse necesare unei sarcini, este recomandat să se determine clar și detaliat *ce activități* presupune acea sarcină și *ce efort* este asociat cu fiecare activitate.
- O regulă destul de răspândită în estimarea duratei unui proiect este **calculul om/lună**:
 1. se estimează cât i-ar lua în luni unui singur om să realizeze proiectul (X luni)
 2. se scoate rădăcina pătrată din X și se obține Y, adică $Y = \sqrt{X}$ = mărimea aproximativă a echipei, precum și $Y = \text{durata proiectului (în luni)}$.

Se recomandă ca evaluarea proiectului să fie realizată **săptămânal**, pentru proiectele mici și pentru cele medii/mari, **lunar**. Astfel, PM-ul va fi în deplină cunoștință de cauză și va putea acționa **corectiv** la timp.

Metode de abordare a întârzierilor proiectelor (5 min)

Pentru proiectele **întârziate**, se poate alege una din următoarele **abordări**:

1. a fost o *întâmplare* și se va munci mai mult în perioada imediat următoare pentru a recupera, deci proiectul nu va fi afectat
2. *nu se poate face nimic*, așa că terminarea proiectului se amână (în caz că este posibil) cu timpul necesar terminării task-urilor
3. se *asignează* încă niște *persoane* care să ajute la terminarea taskurilor în timp util.

Fiecare dintre cele trei metode are avantaje și dezavantaje:

1. pentru prima abordare, dacă nu se reușește să se reintre în program, proiectul va acumula întârzieri, crescându-i șansele de eșec
2. a doua abordare nu este tot timpul disponibilă deoarece nu se poate întârzia oricât cu terminarea proiectului, acesta putând să ajungă irelevant. De exemplu, dacă firma X lucrează în paralel cu firma Y la un produs, chiar dacă firma X face produsul bine, dacă îl termină cu întârzieri față de firma Y, s-ar putea să nu mai aibă clienți.
3. pentru ultima abordare, urmează o dezbatere mai detaliată.
 - **Legea lui Brook** spune că dacă se adaugă persoane la un proiect care este în întârziere, proiectul va fi și mai mult întârziat.
 - alți autori (*Abel-Hamid și Madnick* în cartea "*Software Project Dynamics: An Integrated Approach*") au protestat față de această lege:
 - ei consideră că adăugarea de persoane nu implică neapărat că proiectul va fi și mai mult întârziat
 - ei susțin că adăugarea de persoane într-un stadiu incipient, când persoanele au mai mult timp să cunoască proiectul și să se acomodeze cu echipa ar putea fi benefică, pe când adăugarea de persoane când proiectul este deja în întârziere nu va aduce niciun beneficiu.
 - o *problemă* în adăugarea de persoane este *redistribuirea de sarcini*. Trebuie ca tot proiectul să fie redistribuit pentru noua echipă. Acest proces ar putea să fie foarte complicat și să se piardă mult timp astfel.
 - altă *problemă* este faptul că *efortul de comunicare și sincronizare* a unui număr crescut de membri ai echipei ar putea deveni în anumite proiecte inutil de mare, întârziind clar avansul proiectului.

Înainte de a adăuga noi resurse umane pentru un proiect, **PM-ul** trebuie să determine din **ce categorie** face parte proiectul dezvoltat:

1. **Proiect cu etape nepartiționabile** (indiferent de câte persoane lucrează la acest proiect, el se va termina tot în același timp)
2. **Proiect perfect partiționabil** (cu cât se adaugă mai multe persoane, cu atât proiectul este terminat mai repede)
3. **Proiect care necesită comunicare** (se poate observa o îmbunătățire cu adăugarea de persoane, dar comunicarea implicată face ca efectul să fie mai scăzut ca în cazul proiectelor perfect partiționabile)
4. **Proiect cu interconexiuni complexe** (există un minim în acest graf – care corespunde duratei minime a proiectului și numărului optim de membri ai echipei; dacă

se adaugă mai mulți membri, defapt, se îngreunează munca și timpul de terminare a proiectului crește)

Evaluarea echipei (5 min)

- este o activitate esențială pentru că echipa *influențează major evoluția proiectului*
- se referă la **evaluarea întregii echipe**, dar și la **evaluarea individuală** a membrilor echipei
- astfel de evaluări se realizează periodic pentru a determina eventualele probleme spre a le soluționa și preveni, pentru a recompensa și stimula membrii echipei
- în companii, evaluările individuale detaliate se realizează la terminarea proiectelor și în cadrul auditurilor anuale; ele sunt incluse în arhivele individuale de performanță și constituie baza deciziilor de promovare, asignare de sarcini și bonusuri.
- evaluarea *individuală* se începe prin a îi cere persoanei să se *autoevalueze* (sunt șanse mari să se descopere informații noi despre persoana respectivă – se poate evalua foarte corect, încearcă să mușamalizeze, etc.)
- evaluarea individuală devine un proces mai complicat și laborios cu cât poziția persoanei este mai înaltă în companie: un membru al echipei va fi evaluat de ceilalți membrii ai echipei și de PM; PM-ul va fi evaluat de către membrii echipei, de clienți, de subcontractori, de managementul superior.

În faza de evaluare, PM-ul realizează:

- **concluzionarea** asupra statusului proiectului
- **evidențierea problemelor** și a potențialelor surse de probleme
- evidențierea activităților împlinite cu un *real succes*, deci a resurselor umane implicate și a contextului de lucru

Important: Nu există forme de evaluare perfectă. Orice formă de evaluare este contestabilă. Orice PM va adăuga propriile informații și intuiții în cadrul evaluării, fără a le prioritiza însă. Evaluarea și feedback-ul sunt necesare pentru buna evoluție a proiectului și trebuie tratate cu seriozitate. Evaluarea presupune atât determinarea părților bune cât și a celor slabe. Ulterior evaluării, concluziile sunt de forma: aici stai bine, aici mai ai de lucrat.

Control (5 min)

- este realizat în principal de PM. Contribuie clienții, managementul superior.
- are loc după finalizarea evaluării proiectului
- este o acțiune constantă

În faza de control, PM-ul realizează:

- **raportare** (starea proiectului, planuri de acțiune/schimbare) către nivelul superior de management și către client
- **găsirea de soluții** pentru probleme și de măsuri de evitare a potențialelor probleme și de contracarare a acestora în cazul în care nu pot fi prevenite
- găsierea de modalități de **stimulare a factorilor de succes** (recompensarea celor mai buni din echipă, îmbunătățirea contextului general de lucru în funcție de modelul observat)
- **clarificarea planurilor** viitoare de acțiune
- **actualizarea planului** proiectului, deci implementarea deciziilor de schimbare (corective sau stimulente)

Urmează etapa de diseminare a concluziilor evaluării și a deciziilor de schimbare către echipă.

Important: Controlul este o buclă de feedback pentru acțiunile de monitorizare și de evaluare, așa cum reiese și din schema de mai jos. Controlul presupune "ținerea trenului pe șine". Atunci când apar probleme, PM-ul trebuie să decidă modul de soluționare; de dorit este să prevină apariția acestora, având din timp obținute informațiile necesare. PM-ul trebuie să evite să se transforme într-un "control freak" dar trebuie să evite să fie "devil may care"; echipa trebuie lăsată să lucreze, iar PM-ul să intervină doar când este cazul; eventual poate decide să lase pe cineva să greșească ca proces de învățare. Un PM care vrea să controleze prea mult va demotiva echipa, membrii considerând că sunt tratați ca niște copii; un PM care are un nivel de control redus va fi considerat lipsit de interes și de autoritate și echipa va fi, la fel, demotivată.

Exerciții

Evaluare inițială și control (20 min)

Se dorește realizarea următoarelor proiecte:

1. nucleu de sistem de operare pentru un sistem embedded
 2. server web centrat pe performanță (performance-oriented)
 3. utilitar de project management
- Fiecare proiect trebuie realizat în interval de 6 luni.
 - Estimați numărul de membri ai echipei care să realizeze respectivul proiect (oameni-lună; man-month)
 - Care sunt principalele elemente ale unui formular de evaluare a proiectului în luna 1, 2, 4, și 6 (la sfârșit)?
 - Presupunând că din cauza unor estimări nereușite, PM-ul își dă seama în luna a 4-a că proiectul va dura 8 luni. Ce decizie ați lua în acel moment?

Situații de intervenție (15 minute)

Împărțiți-vă în echipe de 2-3 persoane și decideți cum ați reacționa dacă ați fi PM în cadrul următoarelor situații:

- Una dintre echipele proiectului merge vizibil mai greu decât altele. Datele arată că sunt la 75% din task-uri realizate, iar celelalte echipe la 100%. Membrii echipei lucrează la fel de mult ca alții, dar au o pregătire inferioară. Ce decizie luați pentru a spori eficiența echipei?
- Team lead-ul unei echipe, un om experimentat și muncitor, vă roagă să îi degrevați din activități întrucât soția lui a născut și vrea să petreacă mai mult timp alături de ea. Această degrevare va avea impact în evoluția proiectului. Ce propuneți?
- În trecerea pe lângă bucătăria firmei auziți o discuție între doi colegi despre un alt coleg. Auziți că acest coleg este interesat de o ofertă de la o firmă concurentă. Îl cunoașteți bine și știți că este o persoană valoroasă în cadrul firmei și că plecarea sa ar afecta desfășurarea curentă a proiectului.

Lost At Sea (40 minute)

Împărțiți-vă în echipe de 3-4 persoane pentru a rezolva problema [Lost At Sea](#).

1. Alegerea individuală a kit-ului de supraviețuire (10 minute)

Completați secțiunea Your Individual Ranking, urmărind instrucțiunile din enunț.

2. Alegerea kit-ului de supraviețuire în cadrul echipei (20 minute)

Discutați în cadrul echipei despre alegerile fiecărui membru și încercați să ajungeti la un compromis pentru a crea lista finală.

3. Discutați varianta finală (10 minute)

Dupa ce primiți ranking-ul corect al fiecărui element din lista, puteți să aflați scorul individual și scorul echipei. Scorul se calculează ca diferență în modul dintre ranking-ul individual/echipei și ranking-ul corect. Discutați diferențele obținute. Ce v-a făcut să vă schimbați părerea în cadrul echipei?

(alternativa) Wilderness survival (40 minute)

Stabiliți grupuri de 5-7 persoane pentru a încerca să supraviețuiți în sălbăcie: <https://drive.google.com/open?id=17RA8kyyWcfxwheEK-SBRCKrd7T8ylqwl>

1. Alegerea individuală a răspunsului corect (10 minute)

2. Alegerea variantei de supraviețuire în cadrul echipei (10 minute)

Discutați în cadrul echipei despre alegerile fiecărui membru și încercați să ajungeti la un compromis pentru a crea lista finală.

3. Discutați varianta finală (10 minute)

Dupa ce primiți răspunsul corect pentru fiecare întrebare, puteți să aflați scorul individual și scorul echipei. Scorul se calculează numărând răspunsurile corecte din varianta individuală, respectiv din munca în echipă. Discutați diferențele obținute. Ce v-a făcut să vă schimbați părerea în cadrul echipei?

Pentru laboratorul următor

- De finalizat proiectul.
- De planificat o întâlnire de final a echipei și sintetizat observații după finalizarea proiectului: ce a mers bine, ce nu a mers bine, ce poate fi făcut mai bine în viitor.

Reamintim că deadline-ul hard al proiectului este laboratorul 7.

Documentația

Orice aplicație software dispune de mai multe forme de documentație. Principalele forme sunt:

- documentația utilizatorului (*user documentation* sau *user manual*)
- documentația tehnică a produsului.

Pe parcursul dezvoltării aplicației se pot genera și alte forme de documentație: raportare, administrativă, de natură legală.

Persoanele care se ocupă de realizarea documentației tehnice a unei aplicații / a unui produs sunt numite *technical writers*. Documentația tehnică va cuprinde cerințele software, proiectarea sistemului și arhitectura sistemului. Documentația tehnică este utilă utilizatorului avansat sau administratorului sau unui membru al serviciului de mentenanță a aplicației. Scopul acestei documentații este de a oferi informații/detalii despre funcționarea internă a aplicației și despre legătura între componente.

Documentația de utilizator este, în general, descrisă în cadrul unui manual de utilizare a aplicației. Documentația este destinată oricărui tip de utilizator și este orientată pe a prezenta funcționalitățile aplicației și modul de folosire a acesteia fără a insista pe modul în care aceasta este proiectată. Exceptând formatele standard de manual de utilizare, formele online de ajutor (forumuri, liste de discuții), FAQ-uri, how-to-uri, tutoriale, pot fi, de asemenea considerate documentație de utilizator.

În general, documentația se realizează pe un set de șabloane (template-uri) predefinite. Totuși, o dată cu dezvoltarea Web-ului, un număr din ce în ce mai mare de articole de documentație se găsesc pe wiki-uri, pagini web specializate, blog-uri, în special cele destinate dezvoltatorilor. Forme de documentare precum [MSDN Library](#), [Java Documentation](#), [GNU libc](#) sunt accesibile online.

Generarea documentației tehnice

În proiectele software, un rol foarte important pentru reușita aplicației și buna înțelegere între membrii echipei este modul în care se documentează funcțiile, metodele, clasele și bibliotecile expuse. Pentru aplicații de tip bibliotecă software, este necesar ca utilizatorului acestora (un dezvoltator) să îi fie bine definite și documentate funcțiile și interfețele expuse.

Forme de documentare a codului, interfețelor includ pagini de manual, pagini web, tutoriale, documente LaTeX, references etc. În general, majoritatea formelor de documentare utilizate oferă o interfață web ([MSDN library](#), [pagini de manual](#), [pagini info](#), [reference lib](#), [Java API](#), [Python Standard Library](#), [PHP](#)) pentru a facilita accesul și citirea acestora.

O formă posibilă de documentare a codului, utilizată cu succes de [Java API](#) este generarea automată de documentație din surse folosind utilitarul [Javadoc](#). Folosind un stil special de comentare a surselor, care nu afectează lizibilitatea codului, aplicațiile de generare automată a documentației obțin, în diverse formate, cel mai răspândit fiind HTML, informații despre utilizarea funcțiilor/claselor/metodelor expuse.

Javadoc

[Javadoc](#) este o aplicație produsă de Sun Microsystems care permite generarea de documentație automată pentru surse Java. O descriere amplă a tag-urilor care sunt folosite pentru obținerea documentației găsiți [aici](#). În exemplul de mai jos este prezentat modul în care este folosit Javadoc pentru generarea documentației pentru [acest fișier Java](#). Rezultatul obținut este prezentat [aici](#).

```
razvan@valhalla:~school/2009-2010/mps/labs/lab-06$ javadoc -author -version
MpsTest.java

Loading source file MpsTest.java...

Constructing Javadoc information...

MpsTest.java:16: cannot find symbol

symbol   : class myException

location: class MpsTest

    public MpsTest() throws myException
                               ^

Standard Doclet version 1.6.0_16

[...]
```

Doxygen

[Doxygen](#) permite generarea automată a documentației într-o diversitate de formate pentru un număr mare de limbaje de programare.

[Aici](#) găsiți documentația generată pentru [acest fișier Python](#) folosind [acest fișier de configurare](#).

Pentru obținerea documentației se urmează trei pași:

1. generarea unui fișier de configurare:

2. razvan@valhalla:~school/2009-2010/mps/labs/lab-06\$ doxygen -g mps-doxy.conf
- 3.
4. Configuration file `mps-doxy.conf' created.
- 5.
6. Now edit the configuration file and enter
- 7.
8. doxygen mps-doxy.conf
- 9.

to generate the documentation for your project

10. editarea fișierului de configurare; directiva `INPUT` este folosită pentru a selecta fișierele/directoarele care se doresc parsate

11. rularea doxygen pentru obținerea documentației:

```
12. razvan@valhalla:~school/2009-2010/mps/labs/lab-06$ doxygen mps-  
doxy.conf  
  
13. Notice: Output directory `out-doxy/' does not exist. I have created it  
for you.  
  
14. Searching for include files...  
  
15. Searching for example files...  
  
16. Searching for images...  
  
17. Searching for dot files...
```

```
[...]
```

Patterns și anti-patterns

Domeniul ingineriei software are o dinamică deosebită. Fiecare limbaj de programare sau paradigmă înseamnă folosirea altui set de reguli și noțiuni. Cu toate acestea există un set de reguli de bază care trebuie urmărite pentru a asigura succesul unei aplicații. Aceste reguli au ca scop integrarea facilă a aplicației cu alte aplicații, folosirea de interfețe ușor de înțeles, adăugarea ușoară de noi caracteristici. Un set de caracteristici pe care trebuie să le îndeplinească o aplicație sau un modul sunt:

- **utilitatea** - aplicația trebuie să fie utilă, să aibă sens crearea și folosirea acesteia;
- **simplitatea** - evitarea complexității inutile ([feature creep](#) sau [software bloat](#));
- **claritatea** - interfața modulului/aplicației trebuie să fie ușor de înțeles;
- **generalitatea** - modulul trebuie să aibă un comportament așteptat într-un număr cât mai mare de situații și trebuie să se adapteze;
- **automatizarea** - modulul respectiv trebuie să execute o acțiune care să elibereze o persoană de un efort ulterior (folosită în scripting).

Caracteristicile de mai sus sunt descrise în cartea [The Practice of Programming](#) de Brian Kernighan și Rob Pike.

Aceste caracteristici sunt generice la nivelul oricărui limbaj de programare. Apariția [limbajelor de nivel foarte înalt](#) oferă un context prin care dezvoltatorul nu trebuie să se preocupe de modul de proiectare/dezvoltare a aplicației. Aceste limbaje sunt, însă, foarte specifice astfel încât pentru majoritatea limbajelor este nevoie ca dezvoltatorul să aibă cunoștințe și experiențe în folosirea tehnicilor de tip "best practice" în inginerie software.

Patterns & best practices

Tehnicile de tip "best practices" și "(design) patterns" se referă la mecanisme de proiectare și implementare a unei aplicații/modul pentru ca aceasta să respecte cât mai bine caracteristicile de claritate, generalitate, simplitate. În acest sens se definesc o serie de principii/mecanisme pe care dezvoltatorul/programatorul este recomandat să le urmeze.

Noțiunea de "design pattern" (șablon de proiectare) se referă în special la limbajele orientate obiect. Un design pattern este o formă de abstractizare a unei soluții la o problemă într-o formă generică pentru a permite utilizarea soluției în alte situații. Exemple de [design patterns](#) sunt:

- [Abstract factory](#) - folosirea unei clase abstracte/interfețe care grupează alte fabrici (factories) de obiecte;
- [Lazy Initialization](#) (tehnică folosită și în alte situații) - întârzierea creării unui obiect până în momentul în care este folosit;
- [Singleton](#) - permite instanțierea unui singur obiect dintr-o clasă dată;
- [Iterator](#) - folosirea unui mecanism de parcurgere secvențială a unui obiect agregat fără a expune reprezentarea internă;
- [Null Object](#) - obiect cu valoarea neutră (nulă);
- [Visitor](#) - separarea implementării unui algoritm de obiectul asupra căruia acționează;
- [Decorator](#) - adăugarea unor noi funcționalități unui obiect existent în mod dinamic.

La un nivel generic, există noțiuni de "development methodology/philosophy" sau tehnici/patterns pentru dezvoltarea aplicațiilor cu un nivel de genericitate ridicat - folosite de un spectru larg de limbaje de programare, nu doar cele orientate obiect. Între acestea amintim:

- [Code and fix](#) este o tehnică de eliminare a fazei de proiectare pentru module mici; această metodă presupune implementarea și testarea imediată modulului;
- [Continuous integration](#) se referă la folosirea de tehnici care să minimizeze timpul de integrare; tehnica presupune colaborarea membrilor unei echipe prin folosirea unui repository, a unor aplicații de compilare automată și prin realizarea de commit-uri dese și funcționale în repository;
- [Don't repeat yourself](#) (DRY) se referă la implementarea unei funcționalități într-un singur modul de unde va fi folosită de celelalte module; în acest fel, modificările se vor realiza într-un singur loc fără a necesita schimbări în alte zone;
- [Hollywood principle](#) (*don't call us, we'll call you*)/[Inversion of control](#) (IOC) se referă la inversarea ordinii uzuale de apel între module; modulul inferior în ierarhie poate apela modulul superior pentru a facilita efortul de implementare; folosirea de callback-uri sau evenimente reprezintă forme ale acestui principiu;
- [KISS \(Keep it simple, stupid\)](#) accentuează importanța simplității, care ar trebui să fie un țel important al aplicației și că aspectele de complexitate nenecesară trebuie evitate;
- [MoSCoW](#) este o metodă de prioritizare a acțiunilor în cadrul unui proces de dezvoltare; majusculele din MoSCoW se referă la cele 4 clase de priorități: **M** - Must have this, **S** - Should have this, **C** - Could have this if doesn't affect anything else, **W** - Won't have this time, but Would like in the future;
- [Quick-and-dirty](#), de obicei conotată negativă, se referă la implementarea rapidă a unei metode/clase/modul fără a ține cont de nivelul de generalitate a acesteia; este utilă în situații în care se dorește testarea/verificarea unei anumite componente și nu este justificată implementarea unei metode cât mai bune;
- [Release early, release often](#) (RERO) încurajează livrările frecvente și timpurii ale unei aplicații pentru a crea o buclă de feedback între utilizatori/testeri și dezvoltatori; se permite astfel dezvoltarea rapidă a aplicației în conformitate cu cerințele utilizatorilor pentru software;
- [You ain't gonna need it](#) (YAGNI) este o tehnică folosită în [extreme programming](#) conform căreia programatorii nu ar trebui să adauge funcționalitate până când nu este necesară;
- [Separation of concerns](#) (SoC) se referă la împărțirea unei aplicații în componente cu funcționalități distincte, care nu se suprapun; tehnici de bază pentru obținerea SoC sunt modularitatea și încapsularea.

Un set important de principii, menționate în [The Practice of Programming](#) se referă la proiectarea interfeței expuse de un modul. O interfață proiectată corespunzător va facilita integrarea altor module și scalarea facilă a aplicației.

- ascunderea detaliilor de implementare (încapsulare, abstractizare)
- alegerea unui set ortogonal de primitive
- nu lucra pe la spatele utilizatorului
- același lucru trebuie realizat în toate situațiile (consecvență)

Anti-patterns

În ingineria programării anti-patterns sunt metode de proiectare (organizațional, software etc.) care, deși par evidente și simple sunt de fapt greșite, mai complexe decât e necesar sau suboptime. O listă de anti-pattern-uri clasice se găsesc pe [wikipedia](#). Câteva anti-pattern-uri frecvente printre programatorii mai puțin experimentați sunt:

- [Call Super](#) - necesitatea de a chema o metodă suprascrisă dintr-o clasă de bază;
- [God Object](#) - existența unui obiect cu o funcționalitate foarte largă;
- [Object Orgy](#) - delimitare neconcludentă a variabilelor interne a unui obiect;
 - se recomandă ca variabilele interne să fie accesate prin mutatori și accesorii (*white listing*);
 - de exemplu, în Java:

```
▪ void setVisible(boolean); // mutator  
▪ boolean isVisible(); // accesor
```

- [Sequential Coupling](#) - necesitatea de a chema anumite metode într-o anumită ordine. Metodele respective pot fi incluse în altă metodă care să le apeleze în ordinea corectă;
- Yet Another Useless Layer - adăugare de niveluri de abstractizare nenecesare;
- [Blind faith](#) - încrederea că o anumită funcție este corectă fără a fi testată;
- Accumulate and fire - transmiterea de argumente unei funcții folosind variabile globale, nu parametri;
- [Voodoo programming](#) & [Deep magic](#) - implementarea unei soluții vag înțelese sau folosirea de utilitare necunoscute (vezi http://www.codemaestro.com/reviews/9_sqrt);
- [Magic numbers](#) & strings - folosirea constantelor și a sirurilor de caractere fără o explicație (un comentariu sau denumire de variabilă);
- [Copy and paste programming](#) - lipsa unei generalizări și copierea codului cu mici modificări (vezi Blind Faith);
- [Reinventing the wheel](#) - nefolosirea codului deja existent (cum ar fi third-party software);
- [Programming by permutation](#) - încercarea de a repara un bug prin mici modificări aleatoare fără a înțelege exact unde este greșeala.

Modele de dezvoltare

Modelele/Metodologiile de dezvoltare sunt, în general, clasificate după secvențialitatea acțiunilor. Există, astfel, două clase mari de metodologii de dezvoltare: liniare și iterative. În cadrul celor liniare, fiecare etapă a proiectului se bazează pe etapa anterioară. În cazul modelelor iterative, se realizează o primă parte a etapelor de proiectare, implementare și testare urmând ca, ulterior, să se treacă la faza următoare.

Modelul Waterfall este reprezentantul clasic al metodologiilor liniare. Fazele unui proiect (cerințe, proiectare, implementare, verificare, validare) se succed. Deși oferă o organizare a etapelor ce compun activitățile unui proiect, modelul Waterfall este în general privit ca o opțiune nepotrivită în cadrul proiectelor software. Principalul său dezavantaj este rigiditatea. Majoritatea proiectelor software nu pot defini de la bun început contextul în care se va implementa aplicația. Schimbarea rapidă a condițiilor, constrângerilor, opțiunilor pe parcursul desfășurării activităților proiectului impune iterarea procesului de proiectare/implementare/testare. O extensie a modelului Waterfall este [modelul V](#).

Prototiparea software presupune crearea de prototipuri, versiuni incomplete ale programului care va fi realizat. Aceste prototipuri vor fi analizate și evaluate. Unele prototipuri vor fi aruncate, altele vor fi modificate, iar altele pot asigura baza pentru implementarea efectivă a unui modul/componente. RAD (Rapid Application Development) este o metodologie de dezvoltare software care include dezvoltarea iterativă și crearea de prototipuri.

Modelul spirală presupune realizarea unor cicluri de proiectare și prototipizare. În fiecare ciclu se parcurg 4 faze: determinarea obiectivelor, evaluarea riscurilor, dezvoltarea și verificarea livrabilelor, planificarea pentru următoarea iterație. Modelul spirală este, în general, folosit de companiile mari și activitățile complexe.

Modelul de dezvoltare agilă (agile software development) se referă la un set de tehnici bazate pe dezvoltare iterativă, în care soluțiile și cerințele evoluează prin intermediul colaborării între membrii diverselor echipe. Metodele agile încurajează inspecțiile frecvente, munca în echipă, auto-organizarea, responsabilitatea. Agile Manifesto a menționat noile valori ale modelului:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

[Scrum](#) este o metodă iterativă/incrementală pentru gestiunea unei aplicații complexe, folosită, în general, la un loc cu agile development. Scrum presupune o serie de întâlniri/sprint-uri colaborative precum: daily scrum, scrum of scrums, sprint planning meeting, sprint-ul efectiv, sprint review meeting.

Un codesprint sau [hackathon](#) sau codefest este o activitate în care un grup de dezvoltatori se întâlnesc și programează colaborativ. Un hackathon presupune prezența unui număr considerabil de programatori și desfășurarea de activități pe parcursul câtorva zile până la maxim o săptămână.

Interfețe cu utilizatorul

Cele două forme comune de interfață cu utilizatorul sunt interfața mod text (CLI - *Command Line Interface*) și interfața mod grafic (GUI - *Graphical User Interface*). O dată cu dezvoltarea Internet-ului, se poate considera o nouă formă de interfață de natură grafică, denumită WebUI - *Web User Interface*.

Evoluția și răspândirea sistemelor de calcul și a aplicațiilor au dus la o creștere accelerată a interfețelor grafice și web în ultimele două decenii. Preponderent aplicațiile vor oferi o interfață grafică, deși interfața în linia de comandă rămâne utilizată pentru automatizare și eficiență sau de persoane cu profil tehnic (dezvoltatori, administratori de rețea/sistem).

Nivelul de intuitivitate, ușurință în folosire și eficiență a celor trei tipuri de interfețe variază dar există un set de decizii de proiectare care trebuie avute în vedere în momentul utilizării unei anumite interfețe.

Regula importantă pe care trebuie să o respecte orice interfață, indiferent de tipul acesteia este "oferă-i utilizatorului ceea ce se așteaptă". O interfață care va inversa rolurile butoanelor *OK* și *Cancel* va încurca utilizatorul.

Command Line Interface

O aplicația care oferă o interfață în linia de comandă, va avea, în funcție de natura aplicației, un număr mai mare sau mai mic de argumente. În general argumentele sunt:

- **argumente "pure"**: nume de fișiere, nume de utilizatori, identificatori
- **opțiuni**; opțiunile pot fi clasificate
 - în funcție de prezența unui argument
 - opțiuni simple (`ls -a`, `ps -e`);
 - opțiuni cu argumente `lsof -p 1001` (opțiunea `-p` este asociată cu un pid ce reprezintă PID-ul unui proces);
 - în funcție de "lungimea argumentului"
 - opțiuni format scurt (`-t`, `-l`, `-p`, `-n`);
 - opțiuni format lung (`--tcp`, `--listening`, `--program`, `--numeric`).

Argumentele trebuie să aibă nume intuitive și să permită folosirea tuturor opțiunilor importante ale aplicației. Opțiunile trebuie documentate și se recomandă o formă de ajutor rapid în acest sens (`/?` pe Windows sau `--help` pe Unix). Pentru parsarea argumentelor se recomandă folosirea bibliotecii [getopt](#), disponibilă și pentru [Python](#), [shell](#), [Perl](#), [PHP](#), [Ruby](#).

Graphical User Interface

O interfață grafică (GUI) folosește elemente descrise de WIMP: window, icon, menu, pointing device. Interfața grafică permite folosirea obiectelor anterioare pentru a obține un rezultat sau efect.

Interfața grafică are ca principal scop oferirea unei interfețe ușor de folosit și ergonomice. Paleta de culori folosită, fonturile, dimensiunea obiectelor și plasarea acestora sunt criterii importante pentru definirea interfeței grafice.

Aplicațiile grafice masive (de tipul desktop environment) dispun, în general, de un set de documente denumite [human interface guidelines](#). Acestea oferă dezvoltatorilor și proiectanților un set de recomandări pentru a îmbunătăți experiența utilizatorilor. Aceste recomandări dictează regulile de utilizabilitate a aplicației. Exemple sunt:

- [Apple Human Interface Guidelines](#)
- [Windows Vista Human Experience Guidelines](#)
- [GNOME Human Interface Guidelines](#)

Implementarea interfețelor grafice este facilitată de prezența [toolkit-urilor grafice](#), denumite și *widget toolkit*, precum [Microsoft Foundation Class](#) pe Windows, [GTK+](#) pentru aplicații GNOME, [Qt](#) pentru KDE, [wxWidgets](#) ca interfață portabilă între platforme.

Web User Interface

Interfețele web (WebUI, WUI) sunt interfețele folosite în cadrul browserelor de paginile Web din Internet. O interfață web funcționează în contextul unui sistem client-server în care serverul este furnizorul paginii web care trebuie redată în client (browser). Pe lângă folosirea HTML ca mecanism de bază de reprezentare a paginii și comunicare client/server, noile aplicații dispun de tehnologii precum Ajax, Adobe Flex etc.

O interfață web trebuie să respecte, de asemenea, standarde de utilizabilitate. Totuși, o interfață web dispune de constrângeri suplimentare precum faptul că trebuie să funcționeze

În contextul mai multor browsere și trebuie să țină cont de conexiunea/comunicarea client/server.

Prezentarea Proiectului (45 min)

Prezentarea temei celui de-al doilea proiect (10 minute)

Evaluare informală a echipei și stabilirea noilor echipe (30 de minute)

- Pe o foaie albă A4 trageți o linie verticală pe mijloc; veți obține două coloane. Pe una scrieți + în antet, pe alta scrieți -.
- Solicitați ajutorul colegilor pentru a vă lipi, cu scotch-ul din laborator, foaia pe spate.
 - Lipiți o foaie și pe spatele asistentului.
- Treceți pe la fiecare persoană cu care interacționați și marcați puncte bune și puncte slabe (+ și -). Ar trebui să treceți cel puțin pe la asistent și pe la membrii subechipei voastre.
 - Da, o să fie haos, dar așa este jocul 😊
 - Punctați atât elemente de natură profesională cât și de natură personală.
- Nu e o problemă dacă marcați un punct slab/tare pe care l-a marcat și altcineva.
- După runda de "feedback", dezlipiți-vă foaia, uitați-vă la ea, analizați-o și apoi, pe rând, fiecare spune ce concluzii trage din feedback/evaluare.
- Observație: Unele elemente vor fi, probabil, mai puțin relevante; e cum sunteți percepuți de cei din jur. Asumați acest lucru și exprimați-vă părerea.
 - Feedback-ul are și rol corectiv. E important să spuneți cât de mult considerați că vă caracterizează și dacă doriți și cum doriți să "corectați" anumite puncte.

În urma experienței dobândite și a acestei evaluări, stabiliți echipele (4-6 membri) pentru proiectul următor.

Ce înseamnă "integrare"? (5 min)

Cu puține excepții, proiectele software nu sunt izolate și auto-suficiente, ci depind de alte sisteme, fiind părți dintr-un întreg mai mare. Utilizatorii se așteaptă să poată folosi transparent toate funcțiile acestui întreg, indiferent de componenta software din care o anumită funcție face parte. Pentru aceasta, diferitele componente trebuie conectate împreună. **Acest proces este integrarea.** De obicei, ea constă în "cablarea" datelor și informației între componente. Această "cablare" presupune de obicei:

- definirea datelor de care diferitele componente au nevoie
- definirea transferului acestor date (sursă, storage comun, destinație)
- definirea interfețelor prin care componentele să comunice între ele
- definirea hardware-ului pe care o componentă software va fi instalată ("deployed")
- în cazul în care componenta în cauză are ea însăși subcomponente conectate, pașii de mai sus trebuie repetați pentru subcomponente (*ex. un același proiect software poate avea nevoie de mai multe tipuri de hardware: serverul fizic de baze de date, servere, middle tier - fiecare având instalate doar anumite module și nu tot proiectul*)

Integrarea: un task dificil?

De obicei, **DA**. Motivele sunt mai multe:

Cerințe divergente

Cerințele integrării sunt de multe ori în contradicție cu cerințele de arhitectură ale proiectului. *Ex. din motive de performanță, o componentă software ar putea avea nevoie de un storage local mai curând decât de date remote. Integrarea acestei componente într-un sistem ar putea însă presupune accesul remote la acele date, printr-o interfață.*

Evoluția sistemului

Un sistem evoluează în timp: se adaugă noi funcționalități, noi componente, noi date. De multe ori, sistemul nu mai este optimal pentru cerințele noii componente - sau componentele mai vechi se dovedesc depășite de noile cerințe.

Complexitatea

Un sistem are adeseori o complexitate imposibil de stăpânit de către un singur om. Integrarea poate fi deci un proiect în sine, necesitând conlucrarea unei întregi echipe.

Impactul

Integrarea unei componente noi poate afecta funcționarea celorlalte. *De exemplu: o componentă nouă X accesează o componentă preexistentă, Y. Componenta X poate transfera date eronate care să provoace inconsistențe la nivelul datelor din Y, sau să scoată la iveală buguri preexistente (și anterior necunoscute) din componenta Y.*

Securitatea

De asemenea, mai ales în mediul de Web, pot apărea *breșe de securitate*. În funcție de modul de cuplare al componentelor, aceste breșe de securitate pot fi mai mult sau mai puțin periculoase. *De exemplu: accesul direct a unei componente de pe server la o bază de date poate introduce o breșă în securitatea bazei de date. Compromiterea serverului atrage după sine și compromiterea datelor din baza de date.*

Abordări de Integrare

Integrarea stea

Integrarea "stea" (sau "spaghetti") presupune interconectarea fiecărei componente cu toate celelalte, refolosind la maximum funcționalitățile existente.

Avantajul este economia: refolosirea la maximum a funcționalităților existente în fiecare sistem duce la simplificarea componentelor, care trebuie doar să implementeze interfețe și nu să duplicate funcționalități.

Dezavantajul rezidă însă la nivelul evoluției sistemului: pe măsură ce crește numărul de componente, crește și numărul de conexiuni: fiecare componentă va avea $O(N)$ conexiuni, deci numărul total va fi:

$$\frac{n(n-1)}{2}$$

Treptat, un asemenea sistem ajunge să "scape de sub control". Adăugarea de noi funcționalități - care era foarte simplă în stadiile inițiale - devine inefficientă din cauza overheadului de conectare.

Integrarea verticală

Integrarea verticală presupune crearea de componente cvasi-autonome. În acest scop, fiecare componentă va duplica părți din funcționalitatea celorlalte, inclusiv storage.

Avantajul este că, odată ce componentele au fost concepute și implementate în acest fel, ele pot fi integrate în sistem cu costuri și impact minim. Dezavantajul constă în costul mai ridicat al componentelor, care nu refolosesc funcționalități și module, ci mai curând le duplicatează.

Integrarea orizontală

Integrarea orizontală - sau [Enterprise Application Integration](#) (EAI) - presupune crearea unei componente specializate ([middleware](#)) care este dedicată comunicației între celelalte componente. Această soluție permite reducerea numărului de conexiuni (interfețe) la una singură pentru fiecare subsistem, urmând ca subsistemul middleware să asigure translatarea unei interfețe în alta (respectiv să "ruteze" datele de la o componentă la alta).

Avantajele sunt multiple: pe de o parte componentele au nevoie de o singură interfață, pe de altă parte ele pot refolosi funcționalitățile celorlalte. Componentele rămân separate, putând fi înlocuite sau dezvoltate fără afectarea celorlalte.

Patternuri de integrare orizontală

Din punct de vedere funcțional, într-o integrare de tip orizontal apar doua patternuri tipice, de obicei complementare:

- **Intermediere:** intermedierea conectivității componentelor. Ori de câte ori are loc un eveniment (sau o cerere) într-una din componente, aceasta este distribuită corespunzător către celelalte componente prin interfețele acestora. Componentele folosesc structuri proprii pentru mesajele cu care comunică; mesajele sunt transformate din forma expeditorului într-o structură specifică destinatarului.

- **“Federation”**: legarea sistemului cu mediul exterior prin intermediul interfețelor publice și ascunzând interfețele private. Componentele cad de acord asupra structurii mesajelor și toate folosesc aceeași structură.

Din punct de vedere al propagării evenimentelor sau cererilor, apar de asemenea două situații tipice de transmitere a mesajelor:

- **Sincronă**: Este acea comunicare în care toate părțile participă în același timp la curgerea mesajelor. Aceasta necesită transmiterea mesajului de un participant X, blocarea lui până la primirea răspunsului, prelucrarea mesajului de un participant Y, trimiterea răspunsului înapoi la X. Un exemplu este convorbirea la telefon, în care e necesar ca cel cu care vreau să vorbesc să fie lângă telefon în momentul în care sun. Avantajul acestui tip de comunicare îl reprezintă simplitatea, dar apar dezavantaje precum pierderea timpului, blocarea unui thread în așteptarea răspunsurilor, suprasolicitarea unui destinatar comun.
- **Asincronă**: În această abordare transmitătorul X doar trebuie să aibă grijă că mesajul ajunge în canalul de transmisie; canalul având grijă ca mesajul să ajungă cu bine. Destinatarul, sau receptorul are o coadă în care primește mesajele și le prelucrează la viteza maximă de care dispune. În acest timp expeditorul face alte activități, nefiind blocat în așteptarea unui răspuns. Exemplu ar fi lăsarea unui mesaj în căsuța vocală a celui cu care dorim să comunicăm. Avantajele sunt evidente: separarea funcțiilor, randament maxim, evitarea suprasolicitării; dar toată arhitectura trebuie gândită într-un mod mai complex bazat pe evenimente.

Din punct de vedere al tipurilor de date transmise, acestea pot fi transferate ca și:

- **Fișiere**: O aplicație scrie un fișier pe care o alta îl citește mai târziu. Aplicațiile trebuie să cadă de acord asupra numelui de fișier și locației, formatului de fișier, momentului când acesta va fi scris și citit, și cine va șterge fișierul.
- **Bază de date comună**: Aplicații multiple împărtășesc aceeași schemă de baze de date, situată într-o singură bază de date fizică. Deoarece nu există duplicate ale bazei de date, datele nu trebuie să fie transferate la o aplicație la alta.
- **Apelare de proceduri la distanță**: O aplicație expune anumite funcționalități astfel încât acestea să poată fi accesate de la distanță de alte aplicații, ca și proceduri la distanță. Comunicarea are loc în timp real și sincron.
- **Mesaje**: Aplicațiile trimit mesaje printr-un canal de mesaje comun. Alte aplicații pot citi mesajele de pe canal la un moment ulterior. Aplicațiile trebuie să cadă de acord asupra unui canal, precum și formatului mesajelor. Comunicarea este asincronă.

Service Oriented Architecture (SOA)

SOA reprezintă un set de guidelines de arhitectură pentru sisteme **scalabile** și **extensibile**. Ideea de bază este ca fiecare componentă software să își exporte funcționalitățile sub forma unui **serviciu** accesibil remote, ceea ce deschide calea sistemelor complexe, flexibile și refolosibile. SOA nu este propriu zis o rețetă, ci o colecție de principii:

1. **Incapsulare** - componentele nu au restricții de arhitectură internă
2. **Cuplare slabă** - componentele trebuie să aibă dependențe reciproce cât mai mici
3. **“Contract de servicii”** - componentele aderă la o specificație comună de comunicare și conlucrare reciprocă
4. **Abstractizare** - componentele nu expun public logica lor internă
5. **Refolosire** - împărțirea pe servicii este gândită în vederea refolosirii
6. **Combinabilitatea** - serviciile pot fi combinate și încapsulate sub forma unui serviciu
7. **Autonomie** - serviciile au control total asupra logicii lor interne

8. **Optimizare** - între două servicii echivalente, se va prefera serviciul mai bun din punct de vedere calitativ

9. **Descoperire** - serviciile exportă informații care permit descoperirea și folosirea lor de către alte servicii

10. **Relevanță** - serviciile trebuie să aibă valoare pentru end-useri

Exemplu: Flickr.com, Google.com, Facebook.com - toate sunt gândite ca SOA, exportând servicii prin intermediul unui API. Consecințe: funcționalități ca Google search sau Google maps sunt integrabile în contexte (pagini sau aplicații Desktop) în afara lui Google. În mediul web, acest gen de servicii integrabile în cvasi-orice context se numesc [widgets](#).

Platforme pentru integrare (5 min)

Există multe [soluții software](#) care încearcă să automatizeze integrarea diverselor componente software. Cele mai cunoscute sunt următoarele două:

Application servers

Un [application server](#) este un framework care integrează componentele în jurul unui web server. Componentele comunică prin intermediul serverului. Exemple:

- Apache Geronimo (Apache)
- JBoss (RedHat)
- WebSphere Application Server (IBM)
- WebLogic Server (Oracle)

Enterprise Service Bus

Un [enterprise service bus](#) este un framework având la bază un layer generic de comunicare între diferite componente software. Comunicarea se face prin intermediul *mesajelor*. Exemple:

- WebSphere Enterprise Service Bus (IBM)
- Enterprise Service Bus (Oracle)
- Progress Sonic ESB

Integration testing (5 min)

[Testarea integrării componentelor](#) se face pentru verificarea funcționalității, performanței și stabilității sistemelor compuse. Este un sistem tip cutie neagră, în sensul că testarea nu se face la nivelul componentelor sistemului, ci la nivelul interfețelor acestora. Cazurile de succes și eroare sunt testate prin inputuri (parametri și date) simulate corespunzător.

Scenariile de testare urmăresc să verifice felul în care componentele interacționează. Această etapă de testare este *ulterioară* testării individuale a componentelor (i.e. unit testing) și presupune că toate componentele sunt deja funcționale și corespund specificațiilor.

Există trei [abordări](#) principale pentru testarea integrării:

Bottom-up

Testarea bottom-up presupune testarea componentelor de la nivelul cel mai jos, urcând progresiv către componentele de nivel înalt până către ansamblul sistemului. La nivelul cel mai de jos, se integrează procedurile sau funcțiile în modulele respective, apoi sunt testate. După aceea, se integrează modulele și se face o nouă rundă de testare. Acest tip de testare este foarte eficient în găsirea bugurilor, și de asemenea ușor de monitorizat (se poate ști în fiecare moment ce procent din teste a fost terminat). Are însă marele dezavantaj că nu testează logica și flow-urile principale de date până foarte târziu în cadrul proiectului, ceea ce face ca eventualele greșeli de design să fie depistate relativ târziu. De asemenea, nu permite un eventual release parțial, cu funcționalități limitate.

Top-down

Testarea top-down presupune integrarea și testarea modulelor începând cu cele de la nivelul cel mai înalt. Avantajul este că logica și flowurile principale de date sunt testate mai devreme, deci orice greșeală de design este depistată în timp util. Dezavantajul constă însă în overheadul mare necesar pentru test cases și pentru simulările de date, în condițiile în care sistemele nu sunt terminate și funcționale. Alt dezavantaj este că modulele de nivel jos sunt testate relativ târziu în cadrul proiectului. Nici acest model nu permite un release parțial, cu funcționalități limitate.

"Umbrella" sau "Sandwich"

Este o combinație a celor două metode de mai sus. În prima fază, se testează funcțiile de nivel jos, ca în cazul abordării bottom-up. Ieșirile funcțiilor sunt apoi integrate și testate în maniera top-down. Este o metodă mai puțin sistematică decât oricare dintre cele două anterioare, și mai puțin exactă, însă avantajul este că permite release-uri parțiale, cu funcționalități limitate, înainte de terminarea întregului proiect.

Exerciții

Story telling (30 de minute)

Formați echipe de 2-3 persoane. O echipă pornește cu o poveste formată din 3-4 fraze. Celelalte echipe vor construi **simultan** a 2-a, a 3-a, a 4-a parte a poveștii. Vedeți la sfârșit ce a ieșit.

Exercitiu integrare (40 de minute)

- Identificați componentele proiectului. Ce abordare de integrare considerați ca este potrivită pentru proiectul vostru? Justificați alegerea.
- Alegeti abordarea de integration testing potrivita pentru proiectul vostru. Justificați alegerea.
- Prezentați alegerile făcute în fața colegilor. Aduceți argumente pro pentru alegerile voastre / contra alegerilor colegilor.

Lucru la proiect (30 de minute)

Continuați lucrul la proiect.

Bibliografie

- <http://www.enterpriseintegrationpatterns.com/Introduction.html>
- <http://msdn.microsoft.com/en-us/library/ms978729.aspx>

Termeni

Bug - O problemă, eroare, defect, posibilitate nedocumentată, greșeală într-un sistem software care împiedică sistemul să funcționeze așa cum este de așteptat este un bug.

Bug report - Un raport care detaliază un bug într-un program, adică cum se manifestă, când, care parte a codului este responsabilă, eventual și o soluție posibilă.

Bug tracking - Bug tracking-ul este un mod organizat de a ține evidența bugurilor și a stărilor acestora (deschis, rezolvat, testat, închis). Acesta poate varia de la forme neorganizate (întâlniri în echipă) la forme organizate (liste de discuții, mail-uri, soluții specializate).

Sistem de bug tracking - O aplicație software care are ca rol ușurarea bug tracking-ului (este o metodă eficientă centralizată și ușor utilizabilă.) Există soluții gratuite, dar și soluții enterprise cu prețuri foarte ridicate. Un sistem de bug tracking este un subtip de sistem de issue tracking.

Sistem de issue tracking - O aplicație software care are ca rol centralizarea și adresarea tuturor cererilor legate de unul sau mai multe produse, de la probleme la schimbări de design și aplicări de patch-uri. Cele mai multe sunt în același timp și bug trackere. În general, sunt mai complexe decât bug trackerele.

Patch - O mică bucată de software creată spre a rezolva problemele și a îmbunătăți performanțele unui program. Aceasta include, dar nu este limitată la: repararea de bug-uri, creșterea vitezei, îmbunătățirea graficii. Unele patch-uri creează și probleme noi.

Patch management - Procesul de a stabili o strategie și a plănuți ce patch-uri să fie aplicate, la ce sisteme și când este momentul potrivit pentru aplicarea lor.

Aplicații

GitHub

Așa cum a fost descris în cadrul laboratorului 4, [GitHub](#) permite și un management al bug-urilor.

Redmine

[Redmine](#) este un tool complex, care oferă și suport pentru bug tracking. Prin secțiunea "New issue" se pot publica bug-uri, iar acestea pot fi urmărite.

Bugzilla

[Bugzilla](#) este unul dintre cele mai populare sisteme de urmărire a bug-urilor având o multitudine de [funcționalități](#).

Trac

[Trac](#) nu este atât de sofisticat ca Bugzilla în ce privește urmărirea bugurilor (adică bug tracking), dar oferă mult mai multe alte facilități precum un navigator de Subversion și un wiki, deci o platformă completă de colaborare între utilizatori și dezvoltatori. De asemenea Trac poate fi extins ușor prin intermediul [modulelor](#), unul dintre cele mai populare fiind cel pentru code review.

Debian bug tracking software

Fiecare bug are o adresă de mail distinctă. Comentariile și controlul bug-urilor se face cu emailuri tipizate.

Launchpad

[Launchpad](#) este o aplicație web și un site pentru publicarea proiectelor software (în special cele open source). Printre multe alte facilități interesante dispune și de [bug-tracking](#).

Launchpad poate centraliza rapoartele bugurilor din mai multe surse independente și oferă posibilitatea dezvoltatorilor de a discuta despre acel bug într-un singur loc. Launchpad se integrează bine cu trackere externe: Bugzilla, Trac, Sourceforge, Roundup, Mantis, RT și Debian BTS. Launchpad încearcă să elimine patchurile scrise în comentarii prin oferirea posibilității de publicare a unei ramure de cod (branches). Bugurile pot fi urmărite prin email și atom feeds, fiecare bug având asociată o adresa de email (ex: 249177@bugs.launchpad.net).

Patch-uri

Un fișier patch e un fișier text care descrie diferențele dintre două versiuni ale unui fișier sau dintre două fișiere distincte. Există două formate standard pentru astfel de fișiere: tipul normal și context copiat. Mai jos este prezentat un format îmbunătățit, **Context unificat**, care este cel mai întâlnit și mai ușor de vizualizat format.

Structura unui fișier patch cu context unificat:

```
--- original_file_name      comentarii (ștampilă de timp, versiunea fișierului
în svn/git/etc., etc.)

+++ modified_file_name      comentarii (idem)

@@                          -start_line__original_file,nr_lines__original_file
+start_line__modified_file,nr_lines__modified_file @@

context_line_before_1

context_line_before_2

...

context_line_before_n

- original_line_1

- original_line_2

...

- original_line_m

+ modified_line_1

+ modified_line_2

...

+ modified_line_p

context_line_after_1

context_line_after_2

...
```

Exemplu de fișier patch cu context unificat:

```

--- v1.c      2008-11-18 02:34:38.000000000 +0200
+++ v2.c      2008-11-18 02:38:14.000000000 +0200
@@ -261,8 +261,18 @@
     if (build_key(argv[1]))
         return 2;

-    display("original message: [%s]\n", msg, blocks);
-
+    //display("original message: [%s]\n", msg, blocks);
+    if (blocks)
+    {
+        char c;
+        char * fmt = "original message: [%s]\n";
+        int len = blocks;
+        c = msg[len-1];
+        if (c == 'X')
+            msg[len-1] = '\\0';
+        printf(fmt, msg);
+        msg[len-1] = c;
+    }

    apply(msg, blocks, normalize);

    encrypt((unsigned short*) msg, blocks/2);

```

Creare patch***diff***

'diff' e un program cu care se pot crea fișiere de tip patch având la dispoziție fișierul original și fișierul modificat. `diff` poate crea fișiere patch în oricare din cele trei formate standard:

- **normal**

```
diff original_filename modified_filename
```

- **context unificat**

pentru a crea un fișier patch cu un număr implicit de linii de context unificat:

```
diff -u original_filename modified_filename
```

pentru a crea un fișier patch cu x linii de context unificat:

```
diff -Ux original_filename modified_filename
```

git format-patch

`git` poate genera fișiere de patch care descriu diferențele între două commit-uri. Implicit `git format-patch` crează un fișier diff în format unificat (cel mai răspândit format în lumea open-source) și îl scrie la stdout. Pentru a-l scrie într-un fișier pe disc, trebuie redirectată ieșirea către un fișier:

```
git format-patch [optional parameters] > file.patch
```

Utilizare patch

Un fișier `patch` nu este folosit doar pentru a vizualiza diferențele între două versiuni ale unor fișiere, ci și pentru a transmite un anumit set de modificări de la un utilizator la altul pentru a fi aplicate.

Utilitarul standard cu care se aplică patch-uri se numește `patch`.

Modul cel mai întâlnit de utilizare este:

```
patch -pNUM < filename.patch
```

`patch` își ia fișierul din stdin, de aceea trebuie redirectat fișierul de intrare.

În fișierul `filename.patch`, `diff` sau `git format-patch` va scrie și numele fișierelor pe care le-a comparat și din conținutul cărora a extras diferențele. E posibil ca `diff` sau `git format-patch` să fie rulate într-un alt director față de cel în care se va rula `patch`. Cu opțiunea `-pNUM` se specifică numărul de nivele din calea specificată în fișierul `filename.patch` care vor fi ignorate când se va încerca să se determine.

De asemenea, un patch poate fi aplicat cu comanda:

```
git am filename.patch
```

Test Cases

Pentru o testare de calitate, este important să existe o planificare riguroasă a testării încă din faza de proiectare sau development. Pe măsură ce se conturează definițiile modulelor, entităților de date, obiectelor, claselor, funcțiilor, etc. este recomandabil să se scrie și "scenarii" de testare ale acestora, fie *top-down*, fie *bottom-up*.

În industria software, "scenariile" de test se numesc **test cases**.

Un exemplu de test case:

trebuie verificată funcționarea unei pagini de login care conține un input de user name și unul de parolă

- click pe "Login" fără a completa user/pass → trebuie să rămân în pagină și să se afișeze un mesaj de eroare
- completat user corect, parola incorectă sau nulă, click pe "Login" → trebuie să rămân în pagină și să se afișeze un mesaj de eroare
- completat user corect, parola corectă, click pe "Login" → trebuie să fiu corect autentificat
- completat user corect, parola corectă, tastat "Enter" → trebuie să fiu corect autentificat

Există "testare pozitivă" și "testare negativă", concretizată în **positive test cases** și **negative test cases**. Testarea pozitivă înseamnă *verificarea faptului că sistemul face ceea ce trebuie să facă*. Testarea negativă înseamnă verificarea faptului că sistemul *nu face ceea ce nu trebuie să facă*.

Pe cazul anterior:

- testare pozitivă: se verifică faptul că la user/pass corecte se face login iar la user/pass incorecte se afișează un mesaj de eroare
- testare negativă: se verifică faptul că la user/pass corecte NU se afișează un mesaj de eroare iar la user/pass incorecte NU se face login și NU se "sparge" sistemul

În principiu, cele două approachuri sunt echivalente, însă în practică testarea pozitivă se referă la funcționarea "normală" a sistemului, iar testarea negativă la "corner cases". De exemplu, pentru testarea unui feature critic ca time to market dar non-critic ca și calitate (ex. twitter), se va prefera testarea pozitivă, care asigură că sistemul funcționează corect pentru cei mai mulți utilizatori. Pentru testarea unui feature critic ca și calitate (ex. online banking) se va insista pe teste negative, ex. se va încerca "spargerea" sistemului prin combinații incorecte.

Există, ca în orice alt domeniu, tool-uri open source pentru managementul test cases:

- Bugzilla Testopia: <http://www.mozilla.org/projects/testopia/>
- Incremental Scenario Testing: <https://sourceforge.net/projects/istt/>
- QA Traq: <http://www.testmanagement.com/>

Black Box Testing

Se aplică în cazul în care pentru programul ce trebuie testat sursele nu sunt disponibile, ci doar interfața de acces (binarul, o interfața implementată de clasa testată, etc). Cum se testează: Se aplică un set de intrări, iar ieșirile sunt comparate cu un set de ieșiri corecte.

- Intrări - un set de intrări ce vor fi trimise programului pentru execuție. Intrările nu trebuie neapărat să fie niște fișiere ci pot fi niște apeluri de funcții sau un alt program ce simulează alte module ce interacționează cu programul testat.
- Ieșiri - pentru fiecare intrare se definește un set de ieșiri posibile cu care se testează ieșirea programului. În multe din cazuri ieșirea e deterministică și trebuie doar comparate ieșirea programului cu ieșirea așteptă, iar în alte cazuri este necesară scrierea unui program.

Exemplu Testarea unui program care la intrare primește un string pe care vrea să îl parseze în

- număr întreg
 - limitele inferioare și superioare
 - Unordered List Item câteva constante mici (0, -1, 1)
 - câteva constante mari
 - întregi negativi

- caractere invalide (de exemplu, pentru baza 10 nu putem avea A)
- numere care au 0 în față (dar excesiv de lungi gen: 000000000000000000000000000000666)
- același lucru pentru diferite baze (dacă sunt suportate)
- float
 - ca mai sus, dar în principal trebuie ținut cont de valorile speciale definite în standardul [IEEE 754](#)
 - $-0.0 \neq 0.0$
 - Nan, -Nan
 - inf, -inf
- un șir de caractere
 - limbajul acceptat (LFA style)
- un vector
 - se aplică cazurile de la numere întregi (pentru 1 element)
 - 0 elemente, 1 element, 2 elemente, 5 elemente, număr maxim de elemente
 - caractere de despărțire / elemente invalide

Boundary testing sau boundary value analysis este o metodă de proiectare a suitelor de teste pentru cazurile în care se folosesc **valori la limită** acceptate de program. În general, accentuează testarea "corner case"-urilor. Unele teste care fac parte din suita boundary testing sunt "stress tests".

Primul pas înseamnă, de obicei, partiționarea valorilor posibile în clase valide și invalide. *Exemplu* Un program care primește valori pozitive până în 99, va avea trei clase:

Al doilea pas înseamnă proiectarea unei suite de teste care vor selecta anumite valori care să verifice reacția programului la valori valide sau invalide. *Exemplu* Dacă un program primește valori în domeniul $[-999,999]$, atunci o suită posibilă de test ar fi:

În general, testele trebuie să includă prima și ultima valoare posibilă. De asemenea, se recomandă testarea de condiții extreme de intrare sau ieșire (valori foarte mici, foarte mari, invalide etc.)

- folosirea incorectă a operatorilor relaționali (mai mic, mai mare etc.)
- folosirea de valori constante inițializate incorect
- erori de calcul care ar putea realiza overflow sau wrap-around în cazul conversiei între diverse tipuri de date

- Boundary testing isn't guessing at numbers
- boundary values software testing

- [Boundary Value Testing](#)
- [Boundary Value Analysis](#)

Unit testing

Unit testing este o metodă folosită pentru a testa fiecare componentă a unui proiect. O *unitate* este cea mai mică componentă a unei aplicații. În mod ideal modulele de test sunt independente unele de celelalte. Pentru fiecare *unitate* se fac teste separate.

Există și o abordare **Test Driven Development - TDD** în care se scrie testul pentru unitate înainte de scrierea codului.

Stubs

O problemă des întâlnită în testarea proiectelor este testarea unei părți a proiectului înainte ca alte părți să fie gata. Se pot folosi pentru asta interfețe, numite **Stubs**, care simulează funcțiile de bază ale obiectului respectiv fără să efectueze și teste de integritate a datelor sau ale fluxului logic al problemei. Ele sunt des folosite în cursul dezvoltării unităților proiectului care depind de obiectul simulat.

Mockup

Mockups sunt tot implementarea unor interfețe care testează mai aprofundat funcțiile necesare. Ele simulează spre exemplu funcționarea unui server pentru a putea testa facilitățile clientului și testează de asemenea autentificarea clientului înainte ca acesta să poată efectua anumite tranzacții. Pentru o utilizare mai facilă se recomandă folosirea interfețelor și utilizarea lor în funcția de testare. O implementare pentru testare este o implementare care conține numai cod de test și imită cât mai bine funcționarea viitorului obiect. Mockup-urile sunt utile în multe situații precum: * cazul când obiectul în sine nu există * obiectul real/funcția reală ia foarte mult timp să ruleze * obiectul real este prea dificil de pus în funcțiune * funcția reală returnează valori nedeterminate și se dorește testarea comportării cu toate valorile limită * funcția reală necesită interacțiunea cu utilizatorul și nu se poate folosi în teste automate

Important este ca atunci când se folosesc obiecte pentru simulare, trebuie să se țină cont de faptul că obiectul trebuie să simuleze cât mai bine realitatea. Există și facilități implementate pentru folosirea mockup-urilor în .NET precum NMock, POCMock, .NET Mock Object.

Regression testing

"Also as a consequence of the introduction of new bugs, program maintenance requires far more system testing per statement written than any other programming. Theoretically, after each fix one must run the entire batch of test cases previously run against the system, to ensure that it has not been damaged in an obscure way. In practice, such regression testing must indeed approximate this theoretical idea, and it is very costly." – Fred Brooks, The Mythical Man Month (p 122)

Regression testing implică verificarea ca odată cu avansarea în proiect să nu se piardă funcționalități deja implementate, sau să se genereze erori noi.

Cea mai simplă și eficientă metodă de regression testing este să se păstreze toate testele într-un batch care să se ruleze periodic, astfel orice bug nou va fi remarcat imediat și poate fi remediat. Desigur, asta implică ca testele respective să poată fi rulate automat.

Fault injection

'Fault injection' este o metodă de testare software care implică generarea de input-uri care să ducă programul pe căi (în general de error handling) care altfel ar fi parcurse foarte rar în decursul unei testări normale, îmbunătățind astfel foarte mult code coverage-ul.

Există atât software cât și hardware fault injection.

HWIFI(Hardware Implemented Fault Injection)

Există încă din 1970, și implică crearea de scurtcircuite pe placă, generând astfel erori.

SWIFI(Software Implemented Fault Injection)

Se împarte în două mari categorii # Compile time injection # Run time injection

Compile time injection

Modificarea de linii de cod la compilare pentru a genera comportamente eronate. Ex: a++ poate fi modificat în a-;

Run time injection

- Coruperea spațiului de memorie al procesului
- Interceptarea syscall-urilor și introducerea de erori în ele
- Reordonarea, coruperea și distrugerea pachetelor de pe rețea.

Platforme de testare

- Java: una din cele mai cunoscute platforme de testare este [JUnit](#) pentru care găsiți [aici](#) un tutorial.
- C#: pentru cam toate platformele .NET există [NUnit](#). Un tutorial găsiți la adresa <http://www.nunit.org/index.php?p=quickStart&r=2.4>
- [Python unittest](#) pentru Python.

Testarea interfețelor grafice

Există mai multe utilitare pentru testarea automată a programelor cu interfețe grafice (o listă mai detaliată aveți [aici](#)).

AutoIt

[AutoIt](#) este un limbaj de programare asemănător Visual Basic cu un compilator ce rulează pe Windows și care permite (printre altele):

- apelul unor funcții din DLL-uri Win32
- execuția de aplicații (consolă/GUI)
- creare de interfețe GUI (ferestre de mesaje, atenționare, de introducere de date, etc.)
- manipulare sunete
- simulare mișcări de mouse și apăsare taste și combinații de taste
- manipulare ferestre și procese
- manipulare elemente în cadrul unei ferestre

Scripurile pot fi compilate sub forma unor executabile Win32.

Două tutoriale de AutoIt: interacțiune cu [notepad](#) și instalare [winzip](#)

Abbot

[Abbot](#) este o platformă de testare automată a aplicațiilor GUI scrise în Java. Testele sunt scrise sub forma unor unit-test-uri. Mai multe detalii pe site-ul proiectului.

Deși folosite în special pentru optimizări și pentru identificarea bottleneck-urilor din sistem, utilitarele de tip code-coverage și code-profiling pot fi folosite pentru detectarea anumitor tipuri de probleme precum bucle infinite, sincronizare inefficientă etc.

Code coverage

Utilitarele de tipul code coverage sunt folosite în procesul de testare a programelor pentru inspectarea unei părți cât mai mari a programului. Diversele tipuri de mecanisme de tip code coverage sunt folosite pentru a determina ce funcții sunt acoperite la o rulare, ce instrucțiuni sunt apelate, ce fluxuri de execuție sunt parcurse.

Programele folosesc opțiuni speciale de code-coverage. Cu ajutorul acestor opțiuni se pot determina funcțiile sau instrucțiunile des (sau rar) folosite și oferă o imagine a nivelului de testare a anumitor părți dintr-un program.

În general, utilitarele de code coverage sunt privite ca utilitare pentru depanare automată și sunt folosite, de obicei, de inginerii de testare. Depanarea efectivă, cu utilitare de debugging specializate, este realizată, în general de dezvoltatorii care au cunoștință de codul inspectat.

Code profiling

Profilerele sunt utilitare care prezintă informații referitoare la execuția unui program. Sunt utilitare care intră în categoria "dynamic analysis" spre deosebire de alte programe care intră în categoria "static analysis".

Profilerele folosesc diverse tehnici pentru colectarea de informații legate de un program. De obicei se obțin informații de timp petrecut în cadrul unei funcții (nivel ridicat) sau numărul de cache miss-uri, TLB miss-uri (nivel scăzut).

În general, un program care este "profiled" este instrumentat astfel încât, în momentul rulării, să ofere la ieșire informațiile utile dorite. Spre exemplu, pentru a folosi opțiunile gprof, se folosește opțiunea `-pg` transmisă gcc.

Software Quality Assurance (SQA)

- ciclu de asigurare a calității produsului software
- se desfășoară de-a lungul întregului ciclu de dezvoltare a produsului software, pe care **îl îndrumă, monitorizează, auditează, evaluează și coordonează**
- completează procesul de dezvoltare a proiectului
- (în mediile enterprise) este realizat de către un grup de resurse umane dedicate asigurării calității.

Domeniul QA

- metodologia QA a fost introdusă inițial în cadrul fabricilor, unde și-a demonstrat aportul la succesul producției de calitate
- ulterior, s-a dorit adoptarea QA la mediul dezvoltator de aplicații software
- *Problema:* produsul software nu este palpabil, deci măsurarea funcțiilor, a beneficiilor și a costurilor sunt mai dificil de determinat
- *Soluția:* s-au adăugat concepte precum quality attributes (metrici software, cerințe functionale și non-functionale) pentru a putea măsura calitatea produsului software.

Cel mai cunoscut *model de atribut* ale produselor software de calitate este **FURPS+ model**, care cuprinde următoarele atribute:

1. **funcționalitate** (*functionality*)

- cuprinde toate cerințele funcționale ale sistemului așa cum au fost stabilite în SRS

2. **utilizabilitate** (*usability*)

- cuprinde cerințe legate de ușurința de folosire a interfeței sistemului de către utilizator (interfața trebuie să fie estetică, intuitivă, ergonomică, consistentă, accesibilă și celor cu disabilități)

3. **încredere** (*reliability*)

- cuprinde cerințe legate de precizia răspunsurilor, de disponibilitatea și toleranța la defecte a sistemului

4. **performanță** (*performance*)

- cuprinde cerințe legate de timpii de lucru, precum timp de răspuns (*response time*), timp de revenire (*recovery time*), timp de lansare (*startup time*)

5. **avantaje suplimentare** (*supportability*)

- cuprinde cerințe legate de ușurința testării (*testability*), a mentenanței (*maintainability*), a configurării (*configurability*), de gradul de scalare (*scalability*) și compatibilitate (*compatibility*) și de oferire de multiple traduceri ale conținutului interfeței cu utilizatorul.
- “+” se referă la faptul că la acest model se pot adăuga și alte cerințe de calitate personalizate.

Au apărut **standarde** dedicate produselor software, de tipurile:

- **standarde de managementul calității** (*Quality management*)
 - *ISO 9000 – 3 Quality Management and Quality Assurance Standards – Part 3: Guidelines for the application of 9001 to the development, supply, installation and maintenance of computer software*
- **standarde de documentare** (*Documentation Standards*) specifică formatul și conținutul documentelor de planificare și control și a documentației produsului
 - **standarde de planificare** (*Producing plans*): *IEEE Std1059-1993 Guide for Software Verification and Validation Plans*
- **standarde de Project Management**
 - *General project management: IEE Std 1058.1-1987 Standard for Software Project Management Plans*
- **standarde de Software Engineering**
 - *Software Project Lifecycle: ISO/IEC WD 15288 System Life Cycle Processes*
 - *Software Project Requirements: IEEE Std 1233-1996 Guide for Developing System Requirements Specifications*
 - **standarde de proiectare** (*Design Standards*) specifică reguli și metode de proiectare a soluției pornind de la specificații și formatul și conținutul documentelor de proiectare
 - **standarde de codare** (*Code Standards*) specifică limbajele de programare alese, convențiile de stil, reguli de construire a interfețelor și modul de comentare a surselor.

Ulterior, s-au definit **proceduri**:

- sunt succesiuni de pași ce trebuie urmați pentru a împlini un proces (ex: testare, configuration management).

Activități SQA

Activitățile principale SQA sunt:

- asigură că standardele și procedurile de calitate alese pentru a fi urmate în proiect sunt *adecvate* la specificul proiectului (stabilirea lor este critică pentru că standardele

oferă criteriile de evaluare a produsului și procedurile criteriile de comparare pentru procesele de dezvoltare și control)

- asigură că standardele și procedurile alese sunt *bine documentate* pentru că activitățile de monitorizare, audit și evaluare se bazează pe ele
- monitorizarea proceselor de dezvoltare și control *în raport cu procedurile de calitate* (pașii urmați coincid cu cei ai procedurilor?)
- evaluarea proiectului *în raport cu standardele de calitate*.

Activitățile de evaluare și monitorizare au loc în cadrul **auditurilor**. *Auditul este tehnica SQA de bază folosită pentru verificarea calității produsului*

- întocmește *rapoarte de audit* care sunt predate management-ului, ce afirmă gradul de implementare a calității și oferă *recomandări* pentru implementarea / sporirea calității în faza următoare a proiectului.

Există **activități SQA specifice fazelor** ciclului de viață al proiectului

1.faza de inițiere

- SQA intervine în redactarea și revizuirea planului de management
- asigură că standardele și procedurile alese sunt potrivite, clare și pot servi ca bază de auditare

2.faza cerințelor software

- SQA intervine în revizuirea specificațiilor
- asigură că specificațiile sunt clar exprimate, sunt categorisite corect în cerințe funcționale și non-funcționale (de interfață, de performanță, etc.), acoperă toate cerințele utilizatorului, pot fi măsurate

3.faza de proiectare

- SQA intervine în revizuirea documentelor de proiectare
- asigură că toate cerințele software au fost traduse corect în componente software în raport cu standardele și procedurile de proiectare

4.faza de dezvoltare

- SQA asigură că soluția este dezvoltată în concordanță cu documentul de proiectare și cu standardele și procedurile de codare

5.faza de testare

- SQA asigură respectarea standardelor și a procedurilor de testare
- anunță sfârșitul procedurii de testare a cerințelor funcționale

6.faza de livrare

- SQA verifică și declară respectarea cerințelor non-funcționale
- anunță că produsul final este gata de livrare

7.faza de mentenanță

- SQA intervine pentru a asigura că subciclurile de dezvoltare respectă normele de calitate

SQAP (Software Quality Assurance Plan)

În general, scopul unei organizații este crearea unui **SQAP** care să asigure nivelul dorit de calitate a produsului.

Din standardul [IEEE 730-1998](#), structura unui SQAP conține următoarele secțiuni:

- 1.Scopul documentului (Purpose)
- 2.Documente referite
- 3.Management
- 4.Documentație
- 5.Standarde, practici, convenții, metrice
- 6.Revizii și audituri

7. Managementul riscului
8. Raportarea problemelor și acțiuni de corecție
9. Utilitare, tehnici și metodologii
10. Controlul furnizorului
11. Training
12. Colectarea înregistrărilor, mentenanța

Un exemplu de document SQAP puteți găsi [aici](#).

Terminarea unui proiect

Contexte de terminare

Un proiect se poate finaliza în următoarele contexte:

- **s-a terminat** (cazul cel mai fericit – proiect de *succes*)
- **a fost oprit** din motive (proiect *eșec*):
 - **de afaceri** (apar oportunități noi de business care ar genera un profit semnificativ mai mare decât proiectul actual)
 - **de cost** (bugetul alocat proiectului a fost epuizat și nu mai există surse de finanțare)
 - **de calitate** (calitatea proiectului este foarte redusă, deci investiția nu mai merită)
 - **tehnice** (între timp, au fost lansate soluții mai viabile decât cea a proiectului actual, tehnologiile utilizate de proiect au devenit demodate)
 - **de timp** (timpul alocat proiectului s-a încheiat)

Feluri de terminare a unui proiect:

- *Extincție* (proiectul se termină prin succes sau eșec și, pe viitor, nu se va mai lucra la proiect)
- *Terminare prin adiție* (proiectul se termină cu succes și, pe viitor, echipa care a dezvoltat proiectul se va ocupa de mentenanța produsului)
- *Terminare prin integrare* (proiectul se termină cu succes și resursele sunt reintegrate în alte proiecte ale companiei – modul cel mai frecvent)

Checklist-ul terminării proiectului

- cuprinde *activitățile rămase* din cadrul proiectului
- cuprinde *activitățile colaterale* ce trebuie realizate pentru a încheia proiectul în conformitate cu procedurile actuale:
 - completarea și distribuirea rapoartelor de performanță
 - completarea și distribuirea documentației sistemului
 - completarea auditurilor de calitate
 - completarea auditurilor de vânzare
 - revizia sistemului împreună cu help desk
 - întâlnirea cu staff-ul operațional care urmează să preia software-ul (în vederea mentenanței)
 - împlinirea cerințelor de training ale staff-ului operațional
 - primirea sign-off-ului de la staff-ul operațional
 - oferirea detaliilor despre performanța membrilor proiectului (evaluarea echipei și evaluările individuale ale membrilor și a PM-ului)
 - primirea sign-off-ului de acceptare de la client și acceptarea formală a tuturor livrabilelor proiectului.

În cazul în care se folosește un sistem de bug/issue tracking, trebuie ca *toate issue-urile să fie rezolvate* înainte de finalizarea proiectului chiar dacă ele nu au fost incluse în specificații.

În cazul în care nu se poate acest lucru, cei care predau proiectul trebuie să se asigure că problemele rămase deschise sunt de prioritate scăzută și nu au un impact major asupra funcționalității.

Procesul de terminare

- se decide terminarea proiectului
 - se obțin aprobările necesare (de la clienți, sponsori)
 - se comunică decizia părților interesate
 - se realizează activitățile rămase din proiect și cele colaterale
 - se face post-performance analysis
 - se publică raportul final
 - se sărbătorește terminarea proiectului
 - se reasignează resursele (persoane și echipamente)
 - se efectuează închiderea financiară și administrativă
- Post-Performance Analysis (PPA)

Încheierea unui proiect nu presupune numai livrarea produsului, ci și **oportunitatea de a învăța** din această experiență pentru îmbunătățirea contribuțiilor la proiectele viitoare.

Recomandări:

- analiza finală a proiectului să se bazeze pe *metrici* (așa se pot cuantifica acurat rezultatele);
- exemplu de metrici:
 1. **calitatea produsului:**
 - a. *Defects per KLOC* (Kilo Line Of Code): numărul de buguri per KLOC, unde **KLOC** este unitatea de măsură pentru codul produsului
 - b. *MTTC* (Mean Time To Change): cât timp este necesar pentru a localiza eroarea, a proiecta repararea, a o implementa și testa
 - c. *threat probability*: probabilitatea atacurilor ce exploatează vulnerabilitățile produsului pe o perioadă de timp
 - d. *security probability*: probabilitatea eliminării vulnerabilităților pe o perioadă de timp
 2. **calitatea activităților QA:**
 - a. *Test Coverage*: numărul de unități testate (KLOC/FP) / dimensiunea sistemului, unde **FP** (function point) este unitate de măsură pentru codul produsului din perspectiva funcționalității de business oferite
 - b. *Number of tests per unit size*: numărul de teste per KLOC/FP
 - c. *Cost to locate defect*: costul de testare / numărul de defecte detectate
 3. **Quality of Testing:** $E / (E + D)$, unde E este numărul de erori identificate înainte de livrarea produsului, D numărul de erori identificate după livrare
 - a. *System complaints*: numărul de plângeri din partea clienților / numărul de plângeri rezolvate
 - b. *Test Planning Productivity*: numărul de teste proiectate / efortul pentru proiectarea și documentarea testelor
 - c. *Test Execution Productivity*: numărul de cicluri de testare executate / efortul de testare
 - d. *Test efficiency*: numărul de teste solicitate / numărul erorilor sistemului
- să se determine *cauzele* ce au condus la valorile finale ale metricilor

- să se rețină și să se înțeleagă aceste cauze, să se determine ce *posibilități* ar fi existat/ar exista pentru a evita producerea acestor cauze, ce posibilități sunt de stimulare a producerii a acestor cauze
- să se colecteze *valorile reutilizabile* care au fost produse în cadrul proiectului (proceduri, checklisturi, guidelines) și să se facă publice

PPA:

- este *analiza situației finale* a proiectului în vederea concluzionării asupra factorilor de succes și a modalităților de stimulare a lor, precum și asupra factorilor de insucces și a modalităților de prevenire, rezolvare a efectelor lor
- este *analiza ce stimulează procesul de învățare* din experiența proiectului
- este recomandat să se realizeze periodic (imediat după milestone-uri) pentru a putea folosi concluziile analizei chiar în faza următoare

Procesul PPA are următorii pași:

1. invitarea echipei la analiză (proponând ca fiecare membru să reflecteze asupra factorilor de succes/insucces și să vină cu propuneri de îmbunătățire – se trimite un chestionar de analiză)
2. colectarea individuală a feedback-ului fiecărei persoane implicate în proiect (individual pentru a înțelege întreaga panoramă a proiectului)
3. realizarea unei întâlniri a echipei în vederea concluzionării asupra: ce și cum s-a întâmplat, în ce moduri ar fi putut fi prevenite/soluționate problemele?
4. publicarea și arhivarea sumarului PPA.

În general, se folosește **un chestionar** pentru colectarea informațiilor. De obicei, acesta are conținut diferit între team leaderi și team membri.

Câteva întrebări de chestionar sunt cele de mai jos:

1. Identificați lucrurile care au mers bine în cadrul acestui proiect.
2. Identificați lucrurile care au mers prost în cadrul acestui proiect.
3. Ce evenimente neprevăzute au afectat pozitiv sau negativ desfășurarea proiectului?
4. În cadrul acestui proiect, ce lucruri ați face diferit dacă ar fi repornit?
5. Descrie un lucru pe care tu l-ai fi putut realiza personal pentru a îmbunătăți calitatea produsului obținut din cadrul acestui proiect.
6. Ai fost informat despre ceea ce se așteaptă de la tine în cadrul acestui proiect?
7. Au fost rolurile membrilor echipei definite clar?
8. Echipa a făcut tot ce se putea face pentru a duce la bun sfârșit acest proiect?
9. Cum s-a comportat echipa în ansamblul său?
10. Care a fost componenta cea mai satisfăcătoare din cadrul proiectului?

Un exemplu complet de chestionar găsiți [aici](#)

Exerciții (50 min)

Raspundeti la urmatoarele intrebari:

1. Se da urmatorul patch: [patch](#). Precizati:
 - cate modificari s-au facut?
 - cate fisiere au fost modificate?
 - ce bug ar putea rezolva acest patch?
 - cum ati testa acest fix?
2. Ganditi-va la platforma vmchecker.
 - ce fel de testare face: Boundary testing / Black Box Testing?
 - argumentati si exemplificati
 - Dar dintre urmatoarele: Stubs / Mockups / Regression testing?
 - argumentati si exemplificati

3. Ce fel de testare este ilustrată [aici](#) ?

4. **[Bonus]** De unde vine denumirea de bug?

Nota: fiecare subpunct valorează un punct, se acordă un punct pentru prezență. Punctajul total aferent laboratorului curent este de **11p**.

1. Descrieți sumar conținutul unui SQAP pentru proiectul vostru.

- Parcurgeți secțiunea [Software Quality Assurance Plan](#)

2. Pregătiți un checklist pentru terminarea proiectului vostru. Argumentați!

- Parcurgeți secțiunea Terminarea unui proiect

3. Răspundeți la întrebările de la procesul PPA pentru proiectul vostru.

- Parcurgeți secțiunea Post-Performance Analysis.

Bibliografie

- <http://www.sqatester.com/methodology/PositiveandNegativeTesting.htm>
- <http://www.opensourcetesting.org/testmgt.php>
- <http://www.sqatester.com/documentation/testcasesmpl.htm>
- <http://www.computerhistory.org/tdih/September/9/>

Lucru la proiect (50 min)

Stabiliți ce mai aveți de făcut pentru finalizarea proiectelor.

Laborator 10 - Conducere, motivare, inteligență emoțională

Ce înseamnă "conducere"?

Actul conducerii ("leadership" sau "management") poate fi definit cel mai bine prin scopul său. Două definiții edificatoare:

"Leadershipul înseamnă crearea unui context prin care un grup de oameni pot contribui la înfăptuirea de lucruri extraordinare." - *Alan Keith*

"Leadershipul este capacitatea de a agrega resursele disponibile cu mediul intern și extern în scopul de a atinge obiectivele organizației - sau ale societății." - *Ken Ogbonnia*

Contrar părerii generale, actul conducerii NU înseamnă putere, cât mai ales RĂSPUNDERE. Primul lucru pe care trebuie să îl înțeleagă un lider este ca el poartă răspunderea întregii organizații subordonate, respectiv că eșecul unui membru al organizației este deopotrivă un eșec al conducerii organizației. *Demonstrație: întrucât leadershipul presupune crearea contextului în care poate fi atins un obiectiv, rezultă că succesul sau eșecul în atingerea obiectivului nu pot fi niciodată desprinse de actul conducerii.*

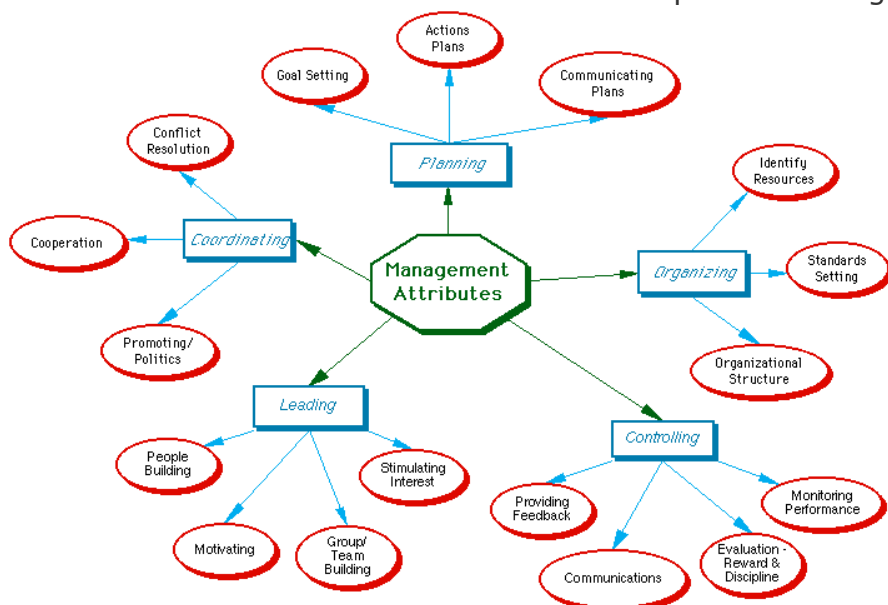
În același timp, responsabilitatea NU trebuie confundată cu implicarea directă, respectiv cu imixtiunea nemijlocită! *Actul conducerii nu presupune schimbarea unui bec sau spălarea cu mopul, ci definirea organizației, a proceselor și a resurselor astfel încât schimbarea unui bec sau spălarea cu mopul să se întâmple în mod firesc.* 😊

Al doilea punct esențial este complementar primului: anume, actul conducerii trebuie să fie ADAPTIV. Nu există rețete universale sau șabloane, ci doar principii și direcții. *Actul conducerii înseamnă construcție inteligentă și nu aplicare.*

Fundamentele actului de conducere

La baza actului de conducere trebuie să stea în permanență scopul final, respectiv crearea aceluia context "câștigător". Crearea și menținerea unui asemenea context se desfășoară simultan pe mai multe planuri. Ele nu trebuie înțelese în sens strict, ci interdependente, cu ponderi flexibile și aplicate în mod adaptiv:

- **Planificare:** definirea obiectivelor și calendarului, ce trebuie să se întâmple și când
- **Organizare:** definirea organizației, repartizarea și gestiunea resurselor disponibile
- **Coordonare/Directionare:** direcționarea organizației către atingerea obiectivelor
- **Control:** monitorizarea proiectului în concordanță cu planurile
- **Motivare:** mobilizarea membrilor echipei către atingerea obiectivelor



- Sursa: <http://www.cognitive-technologies.com/>

Se spune că un leadership (sau management) bun este invizibil deoarece creează iluzia că lucrurile merg spontan, firesc, "de la sine". Consecințele unui leadership de calitate:

- succese ale organizației
- personal motivat
- procese eficiente
- mediu de lucru plăcut
- obiective clare pentru fiecare și în armonie cu ale celorlalți membri

Semnele unui leadership slab:

- nesiguranța sau amânarea deciziilor dincolo de o limită rezonabilă
- nemulțumiri și/sau frustrări în rândul personalului
- efort și stres percepute ca prea mari în raport cu realizările
- lipsa de eficiență a grupului în ansamblu
- divergența de obiective între persoane/echipe/departamente diferite
- eșec repetat în atingerea obiectivelor

"Management" vs "Leadership"

Ambii termeni desemnează actul de conducere, însă termenul "leadership" s-a desprins odată cu lucrarea lui J. M. Burns ("Leadership", 1978) ca un stil de conducere axat pe interacțiunea umană (comunicare-direcționare-motivare) și mai puțin pe "tranzacții" impersonale (control-stimulare-penalizare). Folosind o metaforă, managementul este "știința" care creează organizații funcționale și eficiente, iar leadershipul este un adaos de interacțiune umană care poate aduce în plus entuziasm. 😊

Obiectivul acestui laborator va fi leadershipul.

Leadership: cum?

Leadershipul înseamnă direcționare și motivare. Nu vom discuta aici partea "tehnică" de planificare, construcție și control ale unei organizații, ci doar funcționarea ei optimă în condițiile unei structuri fixe.

Gândiți membrii unei organizații ca număr de vectori (forțe) cu origini fixe (posturile în cadrul organizației) însă cu direcții și module variabile în timp (oamenii). Pentru ca rezultanta să fie maximă în aceste condiții, este nevoie de două lucruri: * vectorii să aibă direcții și sensuri cât mai apropiate * variația vectorilor să fie coerentă (respectiv maximele să se suprapună cât mai mult iar minimele cât mai puțin)

Direcția și sensul înseamnă că membrii organizației împart un set comun de obiective. Variația vectorilor reprezintă evoluția în timp a motivării, a direcției, a efortului depus. În aceste condiții, leadershipul presupune următorul *checklist*:

Definirea unor **objective** "SMART" ale grupului:

- **Specific** – obiectivele trebuie să fie clare
- **Measurable** – trebuie să existe o metrică de măsurare a gradului de atingere a obiectivelor
- **Achievable** – sunt obiectivele tangibile în mod absolut?
- **Realistic** – sunt obiectivele tangibile cu resursele disponibile?
- **Time-based** – în ce interval de timp trebuie atinse?
- comunicarea acestor obiective membrilor grupului

- definirea unui context comun (sistem valoric, terminologie, etc) împărtășit de toți membrii grupului
- transmiterea informației adecvate și într-un mod neechivoc
- **translatarea** obiectivelor organizației în obiective individuale ale membrilor grupului
- înțelegerea aptitudinilor și slăbiciunilor membrilor grupului în mod individual
- identificarea rolurilor potrivite/nepotrivite pentru fiecare
- derivarea unor obiective individuale (de asemenea SMART) în concordanță cu obiectivele organizației
- definirea unor metrici pentru gradul de atingere a acestor obiective
- translatarea obiectivelor individuale în **beneficii** individuale
- cunoașterea membrilor grupului, înțelegerea nevoilor/dorințelor/problemelor fiecăruia
- definirea beneficiului adecvat fiecăruia, respectiv a resortului care îl determină să acționeze pentru îndeplinirea obiectivelor sale individuale
- definirea regulii de corelare a atingerii obiectivului cu obținerea beneficiului individual
- monitorizare
- detectarea și măsurarea abaterilor de la obiectivul global
- translatarea acestora în abateri de la obiectivele individuale
- înțelegerea cauzelor care au condus la acestea
- corectarea cauzelor:
- medierea conflictelor
- clarificarea neînțelegerilor
- dacă este cazul, redefinirea obiectivelor și beneficiilor individuale

În mod simplist, obiectivul organizației poate fi "livrarea proiectului, fără buguri, la termen", un obiectiv individual poate fi "layerul de acces la baza de date până la milestone-ul X" iar beneficiul individual poate fi "nota 10" sau "salariul Y". Această abordare asimilează însă (naiv) oamenii cu automate deterministe, neglijând faptul că indivizii sunt de fapt animați de resorturi interioare mult mai profunde, complexe și adeseori iraționale.

Motivare

Motivația este un sinonim pentru energia mentală pe care este dispus un individ să o investească în atingerea unui scop. **Motivarea** înseamnă "energizarea" unui individ în vederea unui obiectiv (*observați faptul că definiția subînțelege un factor extern individului*).

Orice acțiune din partea unui individ presupune o anumită energie mentală: ex. *deplasarea către un anumit loc*.

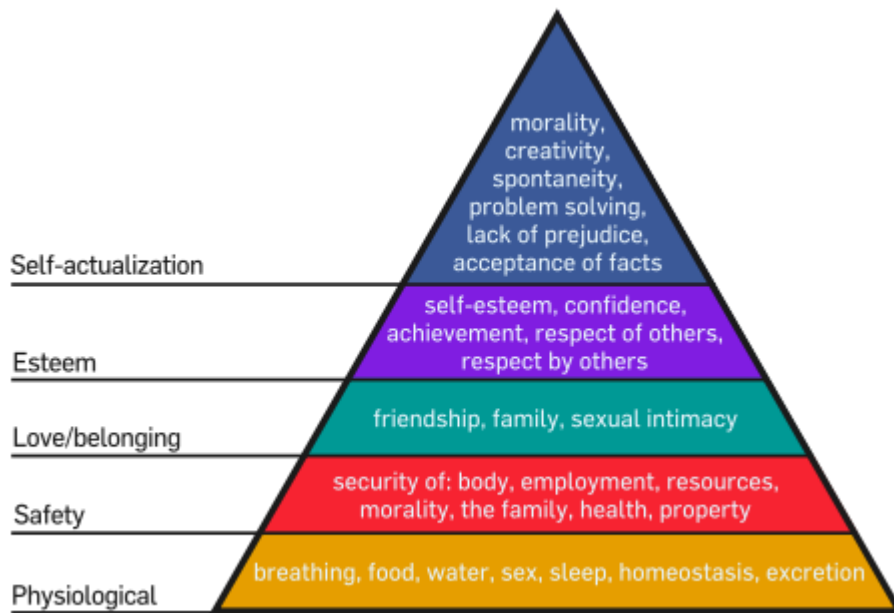
Uneori, această energie apare în mod spontan, **intrinsec**: ex. *atunci când locul este camera prietenului/prietenei*. 😊

Alteori, energia nu este disponibilă în mod spontan, ci doar **extrinsec**, respectiv are nevoie de un aport extern: ex. *atunci când trebuie să vă deplasați exact aceeași distanță, cu același efort, dar pentru a veni la facultate*.

Într-un mediu profesional: * indivizii motivați caută întotdeauna metode tot mai bune de a-și îndeplini obiectivele * indivizii motivați sunt mai orientați către calitate * indivizii motivați sunt mai productivi

Conform cu teoria clasică a lui Maslow (*A Theory of Human Motivation*), energia investită de un individ este alocată conform unei piramide de nevoi ale individului. La baza acestei piramide stau nevoile primare, fiziologice: hrana, respirația, etc. Pe nivelele superioare stau securitatea, apoi relaționarea cu apropiații (familia, etc), s.a.m.d. Individul nu va investi energie pentru un nivel superior al piramidei dacă nevoile de pe nivelele inferioare nu sunt satisfăcute. Lipsa de împlinire a nevoilor de pe nivelul inferior dă manifestări fiziologice. Lipsa de împlinire a nevoilor de pe celelalte nivele dă manifestări psihologice: nervozitate, oboseală psihică, anxietate, etc. Însă, odată nevoile de pe un nivel satisfăcute, individul își va concentra în mod

constant energia pe nivelul superior, fără a mai "reinvesti" în nivelele inferioare altfel decât în mod temporar, atunci când este nevoie.



Motivarea poate fi gândită ca un joc de echilibristică între *amenințarea cu neîmplinirea* și *promisiunea împlinirii* acestor nevoi.

Din start, motivarea prin amenințarea cu neîmplinirea este limitată în timp: individul se adaptează (caz în care ea nu mai are efect) sau renunță (efect contrar celui dorit). În plus, ea crește gradul de stres, ceea ce e contraproductiv. Din contra, motivarea prin promisiunea împlinirii este eficientă, are grad mai mic de stres și aplicabilitate de durată.

Banii

Banii impactează nivelele inferioare ale piramidei lui Maslow, deci sunt un factor puternic de motivare - însă nu unul de durată. Nivelele superioare (*respectul de sine* și *atingerea potențialului maxim*) dau o motivație mai puternică și pe termen mai lung.

Respectul de sine

Nivelul respectului de sine reunește generic nevoile de **posesie** în două subniveleuri:

- subnivelul inferior:

- statut
- recunoaștere/faimă/prestigiu
- atenție

- subnivelul superior:

- putere
- competență
- încredere în sine
- independență
- libertate

Exercițiu (colectiv): *enumerați elemente care pot crește motivarea individului la nivelurile de mai sus.*

Multe elemente simple pot crește motivarea individului la aceste niveluri:

- nomenclatura postului 😊
- *administration associate* sau *administrator adjunct*?

- *documentation specialist* sau *inginer documentare*?
- *security expert* sau *inginer însărcinat cu securitatea*?
- poziția de conducere ("*a fi șef*")
- recunoașterea publică a meritelor sale în cadrul organizației
- recunoașterea importanței și/sau dificultății sarcinilor care îi sunt atribuite
- libertatea de a lua decizii în cadrul atribuțiilor sale
- delegarea unor responsabilități importante
- arătarea respectului și atenției față de el
- *deschiderea de a asculta problemele sale*
- *posibilitatea de a discuta direct, de la egal la egal, cu managementul de pe nivele superioare*
- *tonul amabil al comunicației*
- *sarcinile exprimate în mod prietenos și nu imperativ*

Alte elemente motivaționale de pe acest nivel pot apela la psihologia inversa. Anume, deși individul are o performanță bună, poate primi un feedback ușor negativ care îi poate afecta imaginea de sine – ceea ce îl va împinge să încerce să compenseze acest lucru prin eforturi suplimentare de a se afirma prin ceea ce face. Astfel, individului i se poate spune că:

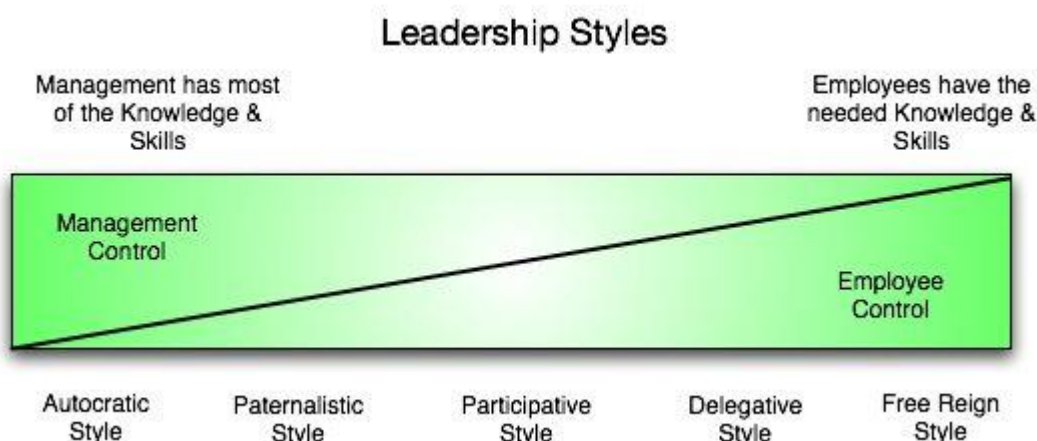
- nu face destule eforturi
- nu e destul de creativ
- colegii nu au atâtea nevoi și plangeri ca el

O altă teorie despre motivare este așa-numita "teorie a așteptărilor" (Vroom, 1964). Conform acesteia, indivizii sunt motivați atunci când sunt convinși că:

- efortul suplimentar va duce la o performanță mai bună
- o performanță mai bună va duce la beneficii atribuite din partea organizației
- beneficiile respective sunt previzibile și valoroase pentru angajat

Stiluri de conducere

Am vorbit mai devreme despre *checklistul* unui leader, dar nu și despre felul în care el "implementează" acest checklist. Din acest punct de vedere, există câteva stiluri de conducere rezumate în imaginea următoare:



După cum se vede, diferențele constau în gradul de delegare a responsabilităților și de participare al subalternilor la decizii. Este important de reamintit că **în leadership nu există rețete, ci construcție**. Un leader experimentat din punct de vedere tehnic care lucrează cu o echipă neexperimentată poate avea rezultate bune pe termen scurt și mediu printr-un management autoritativ sau paternalist, dar acest lucru poate frâna dezvoltarea membrilor echipei, ceea ce dăunează pe termen lung. Invers, un leader care lucrează cu o echipă experimentată are probabil cel mai mult succes printr-o delegare a responsabilităților și

deciziilor, însă aici apare pericolul ca echipa să diveargă de la obiectivele organizației către obiectivele proprii.

Inteligența emoțională

Inteligența emoțională este un concept vehiculat din ce în ce mai mult în ultimii ani, atât în ceea ce privește actul de conducere cât și în ceea ce privește dezvoltarea personală.

Inteligența emoțională se definește drept "capacitatea de a conștientiza, evalua și gestiona emoțiile proprii sau ale altora". Este ușor de văzut de ce este importantă pentru un leader: capacitatea sa de a relaționa cu membrii organizației la un nivel emoțional îi va atrage sprijinul și atașamentul grupului și va echilibra în același timp grupul.

Cei mai buni lideri sunt aceia care nu doar conduc, ci și "inspiră" grupul. Jerald Greenberg ("Organizational Behavior: The State of the Sciences", 1994) identifică mecanismele acestei "inspirații":

- **viziune** coerentă și în concordanță cu valorile de bază ale grupului
- **pasiune** și credință fermă în această viziune
- **încredere în sine**, hotărâre, **consecvență**
- **image personală** construită în mod conștient
- **modelare** – grupul se va identifica cu valorile pozitive ale liderului, lăsându-se astfel modelat de acesta
- **reprezentare externă** – liderul își asumă rolul de a reprezenta grupul în fața unui mediu extern
- **responsabilitate și încredere acordate membrilor grupului** – liderul comunică așteptările sale, ca și încrederea că grupul le va putea împlini
- **comunicare inspirațională** – comunicarea liderului nu este seacă, ci însuflețită, colorată, puternică

Psihologia grupului

Poate părea surprinzător, dar **grupurile se comportă foarte diferit față de indivizii care le compun** - sau, cu alte cuvinte, **indivizii se comportă diferit în grup versus singuri**.

Vom menționa pe scurt câteva caracteristici:

- **"Facilitarea socială"**: Individul singur este mai relaxat și mai indiferent la mediu. În prezența grupului, nivelul său de atenție (alertă) crește. Ca o consecință, individul va executa mai bine sarcinile ușoare (sau automatismele) dar mai prost sarcinile complexe ("Social Facilitation" - Guerin, 1993).
- **Polarizare**: Un grup ai cărui membri împărtășesc opinii similare tinde să treacă prea ușor peste contraargumente atunci când ia o decizie. Membrii grupului concentrează energia pe sublinierea punctelor cu care sunt de acord și evită concentrarea de energie pe cele asupra cărora există semne de întrebare. Deciziile au de aceea tendința de a fi impulsive, unilaterale și dezechilibrate, neglijând aspectele asupra cărora grupul nu este întru totul de acord și exacerbandu-le pe cele cu care este de acord. Simultan, are loc o amplificare reciprocă a energiei, care poate conduce la reacții supradimensionate (*este mecanismul prin care manifestațiile pașnice se pot transforma în lupte de stradă, dar și mecanismul prin care membrii unei echipe se motivează/demotivează reciproc*).
- **Diluarea responsabilității și efectul bystander**: Cu cât un grup este mai mare, cu atât influența individuală asupra grupului este mai mică. Aceasta conduce la o *diluare a responsabilității* resimțită de fiecare individ din grup. Efectul bystander (= martor) este un fenomen observat în justiție, respectiv agresiuni petrecute în văzul a zeci de

martori *care nu au intervenit*. Explicația este o combinație a polarizării și a diluției de responsabilitate: pe de o parte, fiecare membru al grupului așteaptă ca ceilalți să intervină (diluția responsabilității) și pe de altă parte când observă că ceilalți nu intervin are tendința să acționeze în concordanță cu comportamentul grupului (polarizare). *În cadrul unei echipe, efectul bystander poate însemna că un email prin care soliciți ceva are șanse mai mari de succes dacă este trimis unui individ față de cazul în care este trimis pe o listă.*

Cultura organizațională

Ansamblul de valori, termeni de specialitate, reguli, procedee, obiceiuri - formale dar mai ales informale - din cadrul unei organizații poartă numele de **cultura organizațională** și este **contextul comun în care comunică membrii organizației**. Noii veniți într-o organizație "învăță" de la membrii vechi această cultură organizațională și ajung să și-o însușească.

Lipsa ori fragilitatea unei culturi organizaționale înseamnă că membrii grupului sunt permanent expuși neînțelegerilor reciproce (fiecare presupune altceva) și de aici înșelarea așteptărilor, frustrări și lipsa de randament.

O organizație este atât de bună pe cât este cultura sa organizațională.

Relația cu managementul superior

Dacă ierarhia unei organizații este un arbore, atunci actul conducerii se exercită în toate nodurile acestui arbore, la toate nivelurile. Pentru a completa tabloul actului de conducere, trebuie să vorbim și despre relația unui manager cu superiorii săi.

Tipic, un *project manager* sau un *development manager* fac parte din ceea ce se numește *middle management*. Superiorii către care aceștia raportează și cu care relaționează pot fi atât din *middle management*, cât și din *executive management*:



- **CEO** - Chief Executing Officer, conduce întreaga organizație și raportează unui *board* de directori sau investitori
- **CTO** sau **CIO** - Chief Technical (sau Information) Officer, responsabil de tehnologie și de organizația de engineering

Sunt câteva lucruri fundamentale care trebuie înțelese despre superiorul ierarhic, fie el senior sau middle management:

- **vede "mai de sus"**. Aceasta înseamnă că el are context mai larg și o viziune mai de ansamblu, dar în același timp că nu vede atât de multe detalii ca și tine. Corolar: nu îi vorbi mai tehnic decât îți vorbește el.
- **este o resursă shared**. Capacitatea lui de prelucrare a informației este aceeași cu a ta, însă el are mai multe proiecte similare cu al tău. Ca atare, timpul alocat de el proiectului tău este mai mic decât al tău, iar "cuantele" lui informaționale sunt mai mari decât ale tale. El nu poate - și nu trebuie! - să fie la curent cu toate detaliile.
- **este un client**. Tu trebuie să îl informezi, să îi preiei feedbackul - dar să îi și "vinzi", respectiv ori de câte ori îi ceri ceva, să îi comunici clar beneficiile din punctul **lui** de vedere.

Exemple

1. Ai nevoie de niște servere pentru deploymentul proiectului tau, iar bugetul de hardware al firmei este limitat.

- **"cerere"**: Poți cere CTO-ului serverele așa: *"am nevoie de trei servere noi, îmi trebuie pentru proiectul X"*. Va răspunde foarte probabil "nu putem acum".
- **"vânzare"**: Sau, le poți cere așa, "vânzând" beneficiile: *"ne apropiem de finalul proiectului X. Așa cum știți, proiectul X ne va dubla profitul și numărul de utilizatori. Pentru un user experience cât mai bun, recomandăm trei servere noi"*

2. Ai nevoie de un upgrade al versiunii de PHP, care presupune rescrierea unor părți de cod fără a aduce un profit vizibil. Trebuie să obții acordul CEO-ului. El nu este o persoană tehnică dar este deschis și îți poate acorda cinci minute din timpul lui ca să îl convingi de ce upgrade-ul de PHP este necesar.

- **"așa nu"**: *"vrem un upgrade al versiunii de PHP de la 4.0.4 la 5.2.12... am putea și la 5.2.12 RC 3 dar încă investigăm dacă merge cu Zend Server 5.0... upgrade-ul e necesar pentru că 5.2.12 fixează bugul de popen care produce un segfault atunci când pasezi întregi octali care nu sunt moduri de acces"*
- **"așa da"**: *"în momentul de față site-ul este vulnerabil și potențial instabil din cauza unui bug de PHP. Versiunea nouă fixează acest bug. Va trebui să rescriem și porțiuni de cod, dar în final site-ul va fi mai sigur și mai stabil pentru useri."*

3. Echipa X este foarte competentă. Rezolvă cele mai dificile probleme tehnice ale proiectelor fără ca development managerul să știe măcar că au aparut probleme. Echipa Y este mai puțin competentă. Atunci când apar probleme, ei încearcă să le rezolve, nu reușesc, își informează superiorul ierarhic despre dificultăți și cer ajutor sau uneori apelează direct la membrii echipei X pentru consultanță. După un timp, echipei X i se crește salariul cu 5% iar echipei Y cu 20%. *Explicația: din punctul de vedere al superiorului ierarhic, activitatea echipei X este plată, ei par să își facă treaba fără să se fi lovit de probleme. Echipa Y a avut însă probleme, a încercat să le rezolve, a informat despre ele, a cerut ajutor, a primit în final ajutor de la membrii echipei X. Concluzia superiorului: echipa X nu e foarte încărcată (de aceea a avut timp și să ajute Y) iar echipa Y a primit cele mai grele proiecte (de aceea au avut atâtea probleme) și s-a străduit exemplar (așa cum se vede și din comunicările făcute).*

Bibliografie

- http://books.google.com/books?id=kchznb5VKKoC&printsec=frontcover&source=gbs_v2_summary_r&cad=0#v=onepage&q=&f=false
- <http://en.wikipedia.org/wiki/Motivation>
- <http://en.wikipedia.org/wiki/Leadership>
- http://en.wikipedia.org/wiki/Outstanding_leadership_theory
- <http://www.nwlink.com/~donclark/leader/leadstl.html>

- http://www.mindtools.com/pages/article/newLDR_84.htm
- http://en.wikipedia.org/wiki/Emotional_intelligence
- <http://allpsych.com/psychology101/groups.html>
- [http://en.wikipedia.org/wiki/Change_management_\(people\)](http://en.wikipedia.org/wiki/Change_management_(people))

Exercitii (50 de minute)

Exercitiul 1 (10 minute)

Raspundeti la intrebarile din testul urmator https://memorado.com/emotional_quotient pentru a va afla scorul EQ si descrierea inteligentei voastre emotionale. Conform descrierii obtinute, cu ce credeti ca va ajuta inteligenta voastra emotionala in meseria de zi cu zi?

Exercitiul 2 (10 de minute)

Parcurge [acest site](#) și alege cele mai relevante 3-5 elemente care te motivează.

- Există elemente care te-ar motiva foarte puțin, deloc, sau negativ?

Exercitiul 3 (30 de minute)

Grupati-va pe echipe (echipele de proiect). In 10 minute obtineti un achievement cat mai mare aici: <https://clickclickclick.click> In urmatoarele 10 minute avansati cat mai multe niveluri (dungeons) aici: <https://codecombat.com/play/dungeon> Calculati scorul fiecarei echipe. Cine a castigat? Calculul se face in felul urmator: Adunati procentul de achievement de la primul task cu numarul de niveluri depasite la al doilea task inmultit cu 5 si adunati experienta obtinuta (tot la al doilea task).

Lucru la proiect (50 de minute)

Proiectul trebuie prezentat in laboratorul 12.

- Din perspectiva project managerului, ce strategie de motivare ar trebui sa folositi pentru a ajuta echipa sa duca la bun sfarsit task-urile?
- Din perspectiva membrilor echipei, identificati o problema intampinata in timpul proiectului (rezolvata sau inca prezenta) pe care nu ati prezentat-o cum trebuie "conducerii". Prezentați-o sub alta forma si analizati daca reactia conducerii este diferita.

Lucrați la proiect în cadrul echipei.

Laborator 11 - Tehnici de prezentare

Prezentare PPT (10 minute)

View presentations from [Daniel Rosner](#)

Broșură / materiale promoționale

În timpul unei prezentări către client, *materialele de prezentare* costumizate relativ la produs și la client pot aduce un plus prezentării. Acestea denotă interes și atenție față de client și pot fi folosite pentru a transmite mai bine anumite informații și pentru a atrage atenția asupra avantajelor produsului.

Împachetare, deployment (5 min)

Cum un proiect nu poate fi livrat către client sub formă de proiect NetBeans / Visual / etc, clientului îi va fi oferit, în general, proiectul sub forma unui installer.

Există diferite formate / metode de generare de installere, în funcție de sistemul de operare și de platforma software.

În general, mediile vizuale de dezvoltare oferă posibilitatea de generare automată a unui installer pentru proiect, prin intermediul unui proiect de tip installer. (Exemplu: <http://support.microsoft.com/kb/307353>).

Exemplu de aplicație dedicată: <http://sourceforge.net/projects/installgen/>

Scenariu de demo (5 min)

În cadrul unei prezentări, este foarte important pentru client să vadă rezultatul concret. Prezentările teoretice și promisiunile sunt foarte bune în stadiul incipient al proiectului, dar la finalizarea proiectului important este rezultatul.

Pentru a putea demonstra clientului funcționalitatea produsului, se recomandă construirea unui scenariu de demo, care trece utilizarea programului printr-o serie de pași, accentuând funcționalitățile produsului.

De asemenea, un scenariu de demo poate fi folositor și ca help guide. Există aplicații specializate în care pot fi realizate filme de prezentare pas cu pas a modului de funcționare a unui produs software și a modului de utilizare.

O astfel de aplicație este <http://www.adobe.com/products/captivate/>

Aplicații open source:

- <http://camstudio.org/>
- <http://recordmydesktop.sourceforge.net/about.php>
- <http://xvidcap.sourceforge.net/>

Un exemplu de help file realizat folosind captivate (dummy-proof) : <http://elf.cs.pub.ro/~mps/labs/lab-12/navigation.html>

Ofertă de preț (5 min)

În cele mai multe cazuri, scopul final al unui proiect este unul comercial. Astfel, proiectul trebuie să fie oferit la vânzare contra unui cost, care trebuie să:

- acopere costurile de realizare, care pot include:
 - cheltuieli cu personalul

- cheltuieli cu aparatura
- cheltuieli administrative
- licențe
- acopere costurile de întreținere, upgrade, bug fix-ing pe o perioada de timp pentru care o să fie oferit suport către client
- evident, trebuie să existe și un profit în final

Există mai multe modele de vânzare, care depind și de natura proiectului:

- dacă proiectul este realizat la comandă, cel mai probabil va fi vândut unui singur client și din banii obținuți de la acesta trebuie acoperite toate cheltuielile
 - contractul de vânzare poate include clauze de asigurare a mentenanței, training, bug-fixing, upgrading. Banii pentru aceste activități pot fi incluși, fie în suma inițială, fie într-o sumă periodică stabilită prin contract.
- dacă proiectul este unul general, care va fi vândut mai multor clienți:
 - se poate alege varianta cea mai des folosită - clientul cumpără inițial o licență, și primește softul, un anumit grad de suport bine definit, acces la bug-fixuri și upgrade-uri.
 - în ultimul timp, a devenit din ce în ce mai populară varianta în care compania vinde servicii reprezentate de anumite acțiuni posibile folosind soft-ul dezvoltat (**Software as a Service**). O astfel de abordare se pretează foarte bine serviciilor online.
 - o altă variantă este cea în care compania distribuie soft-ul gratuit, iar clientul plătește servicii de suport. Serviciile de suport pot reprezenta servicii de modificare a aplicației pentru nevoile clientului, pregătirea unui installer pentru mediul clientului, suport pentru instalare și pentru folosirea produsului, și alte variante în care clientul deși poate folosi soft-ul gratuit, trebuie să plătească.

Estimare costuri (5 min)

Un mod simplist de a realiza o estimare a costurilor ar fi următorul:

Ce	Unitate	Cantitate	Total
Dezvoltare	40RON/oră*om	500 ore*om	20000 RON
Suport	20RON/oră*om	200 ore*om	4000 RON
Bug fixing	20RON/oră*om	200 ore*om	4000 RON
Costuri promovare	-	-	4000 RON
Chirie	800RON/lună	2 luni	1600 RON
Costuri administrative	800RON/lună	2 luni	1600 RON
Rate (amorsare) echipamente	800RON/lună	2 luni	1600 RON
Rate (amorsare) licențe	800RON/lună	2 luni	1600 RON

Exerciții

Exercitiul 1: Prezentare (20 de minute)

Parcurgeti una dintre aceste prezentari [1]. Ce parti bune/rele observati? Sustineti prezentarea aleasa.

[1] <https://drive.google.com/drive/folders/0B7WjsFvWhaIPRV9VSWRmNEM2UG8?usp=sharing>

Exercitiul 2: Realizarea unei prezentari (40 de minute)

Imaginați-vă că ar trebui să realizați acum o prezentare pentru unul din produsele următoare dezvoltate de echipa voastră: # **Program pentru afișarea online a mersului trenurilor și achiziția online de bilete pentru o companie de cai ferate**. Programul va include și interfețe pentru gestionarea de la ghișeele clasice de bilete. # **Sistem de home-banking** ce permite clienților să folosească servicii online cum ar fi: transfer bancar, schimb valutar, tipărirea de bilanțuri, autorizarea tranzacțiilor online. # **Sistem de supraveghere a traficului dintr-o rețea** destinat corporațiilor. Sistemul va permite prin intermediul unei interfețe grafice închiderea accesului la anumite servicii (yahoo, youtube) a căror utilizare în timpul programului de lucru de către angajați nu aduce un beneficiu companiei. De asemenea, va putea afișa și rapoarte despre activitatea web a fiecărui angajat.

Scopul este ca aceasta prezentare să convingă clientul să achiziționeze produsul vostru. Care este structura acestei prezentări? Ce elemente esențiale ar trebui să conțină? Imaginați-vă că ceilalți colegi ai voștri sunt reprezentanții companiei și prezentați-le cât mai atractiv produsul.

Exercițiul 3: Logo-uri (30 de minute)

Impartiti-va in doua echipe (sau in 3, conform impartirii proiectului 2).

Din fiecare echipa, pe rand, se desemneaza cate un membru care se duce la tabla si deseneaza pentru echipa lui cate un logo din cele puse la dispozitie de catre asistent. Scopul este sa il recunoasca ceilalti membri (ca la mima, sau ca la jocul Activity).

Prezentarea proiectelor

Completare formular de feedback MPS

Pentru a îmbunătăți cursul de MPS, componentele sale și modul de desfășurare, ne sunt foarte utile opiniile voastre. Pentru aceasta, vă rugăm, să accesați și completați [formularul de feedback de pe site-ul cs.curs.pub.ro](#). Trebuie să fiți autentificați și înrolați în cadrul cursului. Formularul este anonim și este activ începând cu 06.01.2020. Rezultatele vor fi vizibile în cadrul echipei cursului doar după încheierea sesiunii. Găsiți [formularul](#) în partea de sus a paginii principale de MPS de pe [cs.curs.pub.ro](#) în secțiunea "Informații utile", accesând link-ul "Formular feedback".

Vă invităm să evaluați activitatea echipei de MPS și să precizați punctele tari și punctele slabe și sugestiile voastre de îmbunătățire a disciplinei. Feedback-ul vostru este foarte important pentru noi să creștem calitatea materiei în anii următori și să îmbunătățim disciplinele pe care le veți face în continuare.

Ne interesează în special:

- Ce nu v-a plăcut și ce credeți că nu a mers bine?
- De ce nu v-a plăcut și de ce credeți că nu a mers bine?
- Ce ar trebuie să facem ca lucrurile să fie plăcute și să meargă bine?

Vă mulțumim!