Universitatea Politehnica Bucureşti
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

AGILE MODELING

Stundeți:
Bălănescu Diana
Vasile Ioana Iuliana
Grupa 442 A

Anul universitar 2011 – 2012

Cuprins

1. Introducere

de Bălănescu Diana

Modelarea Agile reprezintă un set de valori, pricipii şi metode practice pentru documentarea şi modelarea efectivă a sistemelor software [1]. Dezvoltarea unui proiect software constă într-un proces laborios, pentru fiecare etapă fiind necesară o pregătire minuţioasă şi o organizare deosebită. Din acest motiv existenţa unei astfel de metodologii este imperios necesară. Croită împreună cu una din metodologiile mai complexe, deplin dezvoltate precum **XP** sau **RUP**, modelarea agile permite dezvoltarea de procese software care satisfac pe deplin nevoile şi ceirinţele utilizatorilor.

Valorile pe care Modelarea Agile le exercită sunt: comunicare, simplitate, feedback, curaj şi atitudine umilă.

Comunicarea constituie baza modelării cu succes, aceasta trebuie să existe în permanenţă atât între dezvoltatorii produsului software, între dezvoltator şi managerul de proiect cât şi între dezvoltator şi cumpărător.

Simplitatea se referă la concentrarea tutoror resurselor în găsirea unei soluţii cât mai simple posibile care să satisfacă pe deplin nevoile.

A obţine feedback bine pus la punct referitoar la produsul în dezvoltare este fară îndoială un lucru deosebit de important. Dezvoltatorul trebuie să facă toate eforturile necesare pentru ca să obţină informaţiile cât mai corecte şi mai complete.

Curajul de a lua o decizie şi de a o duce până la bun sfârşit face diferenţa între un proiect bun şi un proiect excelent. Curajul de a inova a dus şi va duce întotdeauna spre evoluţie.

Umilinţa reprezintă a cunoaşte faptul că nu le poţi şti pe toate şi a admite că într-o astfel de meserie eşecul nu este un lucru rar întâlnit.

Agile Modeling se bazează pe o colecţie de principii cum ar fi importanţa asumării simplităţii şi versatilitatea. A fi deschis către schimbare este important în acest domeniu deoarece cerinţele se pot schimba pe parcursul dezvoltării proiectului. Utilizatorul trebuie să fie conştient de faptul că modificarea incrementală pe parcurs a sistemului permite dezvoltarea agilităţii şi că trebuie depus mult efortu pentru a obţine feedback rapid şi a se asigura că acesta reflectă în mod corect novoile părţilor implicate în proiect. Modelarea trebuie să aibă un scop, dezvoltatorul trebuie să fie sigur de motivul pentru care lucrează şi să cunoască foarte bine cerinţele beneficiarului. Mai mult, sunt necesare mai multe modele în setul de instrumente intelectuale pentru eficacitate. Un concept critic este faptul că modele considerate nu reprezintă neapărat documente, ceea ce permite un grad de lejeritate în lucru şi de asemenea permite ştergerea modelelor odată ce acestea şi-au atins scopul. Conţinutul este mai important decât reprezentarea, la rezultatul correct se poate ajunge prin mai multe căi. Ceea ce distinge un modelator eficient este comunicarea deschisă şi onestă care de cele mai multe ori constă în cea mai buna politică de urmat pentru a asigura o muncă in echipă eficientă. Calitatea muncii este un pricipiu de la sine înţeles deoarece niciun producător de software nu îşi propunde muncă de mântuială.

Practicile fundamentale pentru Agile Modeling sunt: crearea mai multor modele în paralel, aplicarea artefactului corect pentru situaţia dată şi iterarea la un alt artefact pentru a

progresa într-un mod constant, modelarea în incrementari mici, reprezentarea prin cod a faptului că ideile ce au fost puse în practică funcţioneză, participarea activă a părţilor, asumarea simplicităţii, folosirea unor mijloace simple, utilizarea unor modele anterioare, afişarea în mod public a modelelor, eliminarea modelelor temporare şi actualizarea modelelor numai atunci când este necesar.

2. Valorile AM

de Bălănescu Diana

Valorile Agile Modeling (comunicare, simplitate, feedback , curaj şi atitudine umilă) sunt factorii fundamentali care conduc către succesul în dezvoltarea de produse software.

Comunicarea

Comunicarea este un process prin care informaţia este interschimbată între indivizi prin intermediul unui sistem comun de simboluri, semene şi tipuri de comportament. Comunicarea trebuie să fie bidirectională, informaţiile trebuie să fie atât dobândite cât şi furnizate. Experineţa arată că eficienţa în comunicaţie – între toate persoanale participante la proiect (dezvotatori, acţionari, beneficiari) – reprezintă un criteriu important pentru succesul în dezvoltare. De exemplu, unul din dezvoltatori nu îşi anunţă colegii că partea lui de cod nu funcţionează corespunzător, ceea ce ar presupune ca un alt coleg să muncească în plus pentru a descoperi problema şi a o rezolva. Un alt exemplu este atunci când utilizatorul nu explică importanţa anumitor cerinţe iar dezvoltatorii se concentrează pe anumite aspectede impact scăzut, ignorând pe acelea care pot fi critice necesităţilor organizaţiei. Unul din prcincipalele motive pentru care avem nevoie de modelare este pentru a îmbunătăţi comunicarea însăşi. De exemplu printr-o reprezentare grafică se pot expunde o multitudine de idei şi descrie procese care în cuvinte ar fi fost foarte complex de explicat. Modelarea ajută la comunicarea ideilor proprii şi la înţelegerea ideilor celorlalţi.

Simplitatea

În domeniul dezvoltării de software există o frază celebră: *Keep it simple!*[2], adică *Menţine totul simplu!*. Simplitatea este o valoarea de bază în Industria IT însă din varii motive aceasta nu este întodeauna adoptată. Pot apărea diverse porbleme care complică lucrurile. Cele mai comune complicaţii sunt:

- Aplicarea unor modele complexe prea devreme din diverse motive fie că sunt impuse, fie că sunt necesare sau fie pentru simplul fapt că sunt interesante. Aceste modele au nevoie însă de mai mult timp pentru a le construi, testa şi implementa.
- Supraîncărcarea arhitecturii sistemului cu scopul de a prevede cerinţe viitoare potenţiale. Prevederea de suport pentru cerinţe viitoare este un lucru important însă acest lucru trebuie facut cu măsură, deoarece exista riscul de a complica foarte mult produsul şi de a îngreuna funcţionarea lui actuală.
- Dezvoltarea unei infrastructuri complexe. O greşeală comună pe care echipele de proiect o fac este aceea de a investi o primă porţiune din timpul de dezvoltare infrastructurii ce implică dezvoltarea componentelor tipice, a framework-urilor, librăriilor pe care intenţioneză să le folosescă drept blocuri de construcţie a sistemului lor. Aceştia pleacă de la premisa că în timp cheltuielile se vor amortiza. Decizia luată implică o serie de neajunsuri. În primul rând se cheltuieşte o parte din buget fără a avea rezultate

concrete. În al doilea rând se asumă un risc ridicat faţă de acţionari dacă nu se furnizează funcţionalitatea proiectului rapid. O abordare mai bună este dezvoltarea infrastructurii de-alungul dezvoltării produsului.

Scopul fundamental este de a păstra modelele cât mai simple posibil. "*Modelează astăzi pentru a prevedea nevoile de azi şi fă-ţi griji mâine pentru nevoile de mâine!*" [3]

Feedback

Singura modalitate prin care se poate determina dacă munca depusă în dezvoltarea unui produs software este corectă este prin obţinerea de feedback asupra modelelor. Există o diversitate de metode prin care se poate obţine feedback privind un model:

- Dezvoltarea modelului în echipă
- Analizarea modelului împreună cu utilizatorii implicaţi.
- Implemenatrea modelului. Cel mai sigur mod de obţinere a feedbackului este prin implementarea produsului.
- Testarea produsului. Această testare este facută de acţionari pentru a vedea daca produsul indeplineşte cerinţele acestora.

Un alt lucru de care trebuie ţinut cont este timpul necesar obţinerii feedbackului. Dacă se lucrează în echipă feedbackul este obţiunt aproape imediat. Dacă au loc analize informale, procesul poate dura ore. Dacă are loc o analiză formală procesul poate dura zile sau chiar săptpmâni. Cu cât se obţine feedback mai repede cu atât modelele sunt corespunzatoare necesităţilor şi cerinţelor.

Curaj

Metodologiile Agile cer ca dezvoltatorii de produse software să lucreze cu alte persoane, sa aibă încredere în ele şi de asemenea saibă încredere în propriile forţe. Acest lucru necesită curaj. Tot acestea cer să ai încredere că problemele de mâine se vor rezolva mâine, pentru acest lucru este nevoie de curaj. AM cere să creezi documentaţie numai atunci când aceasta este imperios necesară şi să laşi persoanele care se ocupă de afaceri să ia anumite decizii cum ar fi prioritizarea cerinţelor. Aceste lucruri necesită din nou curaj. Curajul reprzintă o cerinţă de bază în dezvoltarea de produse software. În primul rând curajul este important deoarece dezvoltatorul trebuie să aleagă o abordare agile şi de a îşi menţine decizia până la finele proiectului. În al doilea rând în timpul dezvoltării curajul este necesar pentru a lua decizii importante. Şi, în al treilea rând este nevoie de curaj pentru a recunoaşte greşelile.

Atitudine umilă

Cei mai buni dintre dezvotatori au umilitaţa de a recunoaşte faptul că nu cunosc totul. Modelatorii Agile înţelg că managerii de proiect şi colegii lor de breazlă sunt pregătiţi în anumite arii de expertiză şi reprezintă persoane de valoare pentru succesul proiectului. De asemenea şi utilizatorii pot înleţelege mai bine anumite aspecte legate de proiect decât însuşi dezvoltatorul. Modelatorii Agile au umilinţa de a admite că au nevoie de ajutor pentru a îşi îndplini cu success jobul. Aceştia ştiu să lucreze în echipă, respectă munca altora şi realizează că alte persoane poat avea priorităţi şi aşteptări diferite. Aroganţa duce la probleme de comunicaţie, care la rândul lor pun proiectul în pericol. Modelatorii Agile sunt umili şi drept rezultat sunt eficienţi.

3. Principii Agile Modeling

de Bălănescu Diana

3.a. Principii de bază

În acest capitol vor fi prezentate principiile de bază pentru AM. Aceste principii trebuie înţelese pe deplin pentru a putea implementa corespunzător tehnica AM. Acestea sunt:

- Dezvoltarea software-ului
- Next effort
- Lejeritate în modelare
- Asumarea simplitatăţii
- Acceptarea schimbărilor
- Modificarea incrementală
- Modelarea cu un scop
- Modele multiple
- Calitea muncii.

Dezvoltarea software-ului

Scopul principal in dezvoltarea de software constă în producerea de software de bună calitate, ce poate mulţumi toate nevoile unui proiect. Acest principiu nu trebuie să aibă ca şi rezultat producerea de documentaţie neesenţială. Creerea unei documentaţii complexe nu are nici o importanţă propiru-zisă. Important este axarea pe creerea unui sistem bazat pe cerinţele utilizatorilor şi concentrarea tuturor resurselor pentru a atinge scopul final.

Enabling the Next Effort

Un lucru foarte important intr-un proiect este acela de a permite ca produsul software să fie destul de robust încât să poată fi scalat cu uşurinţă în timp. Următorul pas poate fi acela de a creea o versiune nouă pentru un produs deja existent, sau de a upgrada versiunea curentă . Pentru acest lucru trebuie creeată o documentaţie ce poate permite pe viitor dezvoltatorilor să creeze noi versiuni cu uşurinta. Aspectul important ce trebuie ţinut mereu în minte este acela că în crearea unui produs este întotdeauna necesară o perspectiva de viitor.

Lejeritate in modelare

Aceasta lejeritate constă în faptul că nu trebuie create decât modelele şi documentaţia ce sunt absolut necesare. Acest lucru este important deoarece fiecare model pe care un dezvoltator îl crează, trebuie menţinut în timp; astfel creerea artefactelor inutile va umple volumul de muncă şi posibil va deplasa solutia intr-un punct mort. Cu cât numarul de modele este mai mic, cu atât schimbările la nivelul lor sunt mai puţine.

Similar, cu cât modelele sunt mai complexe (detaliate), cu atât schimbările vor putea fi implementate mai greu [4]. De fiecare dată când se acceptă un model se pierde din agilitate, însă acest lucru este uneori convenabil deoarece dezvoltatorul va avea mai multe perspective asupra problemei si va putea alege pe cea mai bună dintre ele . Aşadar, trebuie puse în balanţă efectele pe care le are alegerea unui numar de modele şi trebuie aleasă varianta corespunzătoare.

Un alt aspect ce trebuie avut în minte este documentarea modelelor. Este de bun augur ca un dezvoltator sa decidă ca numai acele modele cu adevarat necesare sa fie documentate periodic.

Comunicarea în rândul echipei de programatori este o virtute ce trebuie dobândită dacă se doreşte lejeritate în modelare, deoarece numai aşa problemele pot fi rezolvate în timp util.

Lejeritatea în modelare permite dezvolatatorilor o abordare simplă, deoarece timpul de întreţinere a modelelor scade dramatic.

Asumarea simplităţii

Pentru un dezvolator cel mai important aspect de care trebuie să ţină cont este acela că soluţia cea mai simplă este întotdeauna şi cea mai bună. Avantajul constă in faptul că timpul nu este pierdut în implementarea soluţiilor dificile. De asemenea, chiar dacă soluţia simplă nu se dovedeşte a fi cea bună, măcar timpul cheltuit în implementarea ei a fost mult mai mic. Modelele simple au o rată de succes mult mai mare.

Acceptarea schimbărilor

Trebuie cunoscut şi acceptat faptul că de-a lungul produceri unui software pot avea loc unele schimbări.

Aceste schimbări apar datorită cerinţelor care evoluează in timp. Un modelator agil acceptă cu căldură schimbarea. Aceştia inteleg că intr-un proiect schimbările sunt lucruri obişnuite. Modelatorii trebuie să comunice cu acţionarii de proiect pentru a putea observa implicaţiile unor schimbări şi de a estima costurile introduse de acestea.

Modificarea incrementală

Pentru a putea accepta schimbarile, dezvoltatorii trebuie sa aibă o abordare incrementală, pentru a schimba sistemul încet, câte o părticica pe rand, în defavoarea unei abordări ce ar implica schimbarea sistemului intru-totul.

Un concept important in modelarea agile constă în faptul că nu trebuie ca software-ul să funcţioneze perfect din prima, ci mai degrabă să existe un punct de plecare.

Modelarea cu un scop

Dezvolatatorii nu trebuie să creeze un produs din simpla dorinţă de a modela. Fie acel produs le este cerut de catre managerii de proiect, fie doresc să dezvolte un produs pentru ei pentru a îşi satisface o nevoie sau fie din simpla dorinţă de aduce inovări în acest domeniu, trebuie avut întotdeauna un scop. Cele mai bune motive pentru care un dezvoltator modelează ar fi: modelarea pentru a întelege mai bine un proiect, modelarea pentru o mai bună comunicare. Un motiv invalid pentru crearea unui model ar fi urmatorul: cineva cere acel model, însă nu este în stare să explice nesitatea acestui lucru.

Primul pas atunci cand se pune problema modelării constă in găsirea unui scop valid pentru a face un model şi a utilităţilor acelui model.

Modele multiple

Dezvoltatorul are la dispoziţie o serie largă de artifacte pentru modelare. Aceste artifacte includ: diagramele UML (Unified Modeling Language), atrifacte pentru dezvolltare structurată cum ar fi modelele de date, artifacte folosite în dezvoltarea interfeţelor cu utilizatorul. Fiecare model are punctul lui forte, insă are şi slabiciuni. Fiecare model este convenabil pentru anumite situaţii, însă pentru altele poate fi complet nefolositor. Ideea este că, pentru a fi eficace în descrierea sistemelor software trebuie să se folosească modele multiple, deoarece fiecare model existent în prezent poate descrie doar un singur aspect al softwareului în dezvoltare.

Calitea muncii

Dezvoltarea sau dorinţa de a dezvolta un produs de calitate nu este un lucru rar întâlnit în cazul programatorilor agili. Aceştia înţeleg nevoia de a investi efort în ceea ce fac pentru a ajunge la un rezultat de calitate de care se pot mândri, care satisface nevoile clientului şi cerinţele managerului.

3.b. Pricipii suplimentare

Peincipiile suplimentare Agile Modeling defineste o serie de concepte care cresc eficienţa ca modelator Agile [5].

Aceste principii sunt:

- Conţinutul este mult mai important decât reprezentarea
- Fiecare are ceva de învăţat de la toţi ceilalţi
- Cunoaşte-ţi modelele
- Adaptarea locală
- Comunicarea deschisă şi onestă
- Instinctele modelatorilor

Conţinutul este mult mai important decât reprezentarea.

Orice model dat poate fi reprezentat în mai multe moduri. Acesta poate avea o documentaţie diversificată, sa fie frumos colorat, sa aibe alături imagini, grafice, diagrame. Aceste lucruri nu sunt necesare şi nu aduc nici un rezultat şi duc la nici o finalitate. Importantă este asigurarea unei funcţionalităţi produsului şi ducerea lui la bun sfârşit.

Fiecare are ceva de învăţat de la toţi ceilalţi

Modelatorii Agile recunosc faptul că nu pot cunoşte perfect un anumit domeniu. Există întodeauna oportunităţi de a învăţa lucruri noi şi de a extinde cunoştinţele. Tehnologiile sunt într-o continua schimbare, cele existente evoluează iar npe lângă ele apar tehnologii noi. Tehnicile de dezvolare existente evoluează, ce-i drept într+un ritm mai lent, însă fară îndoială evoluează. Ideea este că, într-o astfel de industrie, trebuie ca să se ia în considerare fiecare oportunitate de a învăţa ceva nou, de a citi, de a practica, de a lucre în echipă şi de a împărţi şi cu ceilalţi cunosştinţlele dobândite.

Cunoaşte-ţi modelele

Pentru că există o multitudine de modele ce pot fi aplicate, un modelator agile, trebuie să le cunoască punctele forte şi defectele pentru a putea alege modelul potrivit şi pentru a îl implementa.

Adaptare locală

Faptul ca AM va putea fi aplicat ca la carte prezintă urme de îndoială, in schimb vor trebui făcute cîteva schimbări care sa reflecte mediul si natura organizaţiei. Atunci cînd se doreşte adaptarea AM in funcţie de nevoi, va trebui luată in vedere alegerea facută in cazul tehnicii de modelare.

Comunicarea deschisă şi onestă

Trebuie sprijinită ideea libertăţii de exprimare. Comunicarea liberă şi onestă le permite oamenilor să ia decizii bazate pe informaţii precise.

Comunicarea liberă şi onestă necesită devotament din partea tuturor si o inţelegere a faptului că numai comunicarea poate duce la proiecte de succes.

Instinctele modelatorilor

Pe măsură ce un modelator cîştigă experienţă in domeniu, cu atît instinctul său devine mai ascuţit. Modul în care instinctele ghidează un modelator poate avea o importanţă hotărâtoare în modelarea pe care acesta o are de făcut.

4. Procese software agile

de Vasile Ioana Iuliana

Există diferite versiuni ale metodelor de dezvoltare softare agile, fiecare păastrează valorile agile, dar aduc îmbunătățiri pentru anumite caracterisitci alte produselor ce urmează a fi dezvoltate. Printre cele mai folosite metode agile se numără programarea extremă (*eXteme Programming*) și Scrum.

a. Programarea extremă (XP)

Programarea extremă este folosită în echipe de lucru mici sau medii pentru o dezvoltare rapidă și eficientă, cu o durată și un cost predictibil.

„*Xp este o metodologie ușoară pentru echipe mici spre medii care dezvoltă produse softwere care au cerințete vagi sau schimbătoare*” este definiția dată de Kent Beck, creatorul noțiunii de programare extremă [6].

Cele patru valori ale programării extreme sunt: comunicarea, simplicitatea, feedback și curaj.

Programarea extremă pune preț pe comunicarea față în față, in detrimentul celei prin documentație, deci comunicarea personală este importantă. Pentru facilitarea comunicării, echipele trebuie să folosească un vocabular comun, ușor înteles de toată lumea, să lucreze îndeaproape cu echipa, într-un spațiu deschis, dar să păstreze același tip de legătură și cu colegii din sectorul de business. De asemenea, echipa trebuie să integreze continuu codul, să programeze în subechipe. Cât despre relația cu clientul, echipa trebuie să îi prezinte acestuia frecvent rapoarte despre starea proiectului.

Regulile pentru proiectarea simplă date de Kent Beck sunt:

1. Executută toate testele
2. Să nu exite cod duplicat
3. Exprimă toate ideile pe care autorul vrea sa le exprime
4. Minimizează clasele și metodele

Clientul privește progresul echipei prin intermediul unui soft funcțional primit la un interval de timp prestabilit. Acest feedback continuu îi perimite acestuia să orienteze proiectul, să îl ghideze către nevoile, dorințele lui. De aceea, programatori trebuie să dezvolte în versiuni mici, să fragmenteze aceste versiuni în iterații mai mici, să apoi să fragmenteze caracteristicile pentru fiecare iterație, apoi să fragmenteze și mai mult caracteristicile in unele mai mici. În final, trebuiesc conduse teste pentru integritatea produsului.

Curajul în programarea extreme se poate traduce în încredere, încredearea în coechipieri, în practicile lor, în sine și în clienții lor. Aceasta încredere înseamnă să: lase deciziile de afaceri clientului, ceară ajutorul coechipierilor sau clientului cându au nevoie, să nu amâne treaba, să îmbunătățească codul, să schimbe cod care nu a fost scris de ei, să schimbe procesul când acesta nu funcționează.

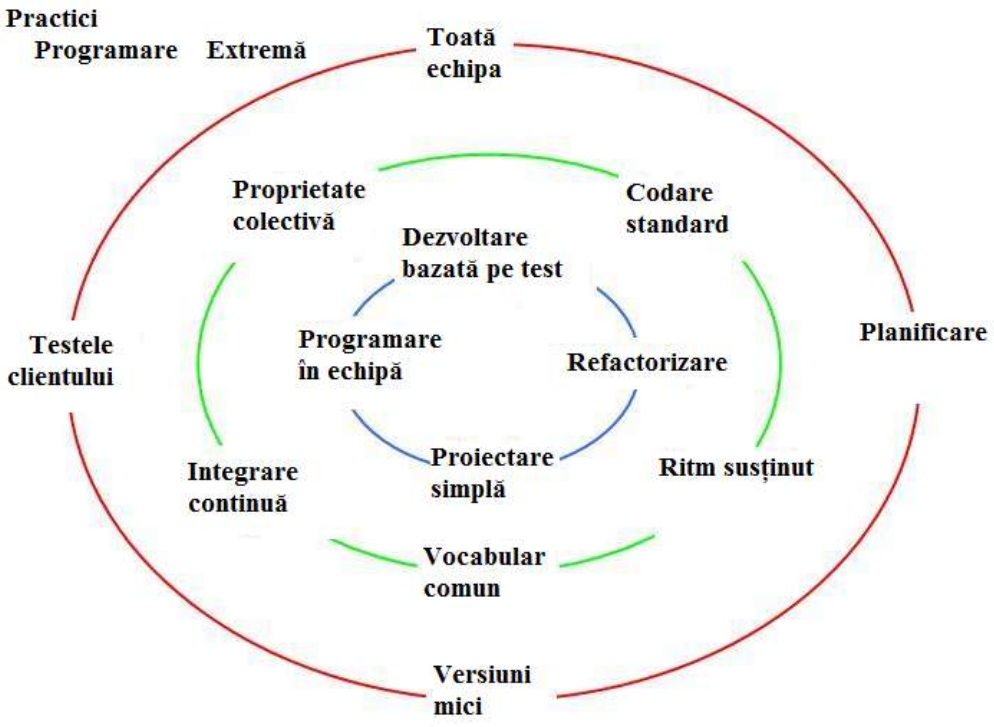


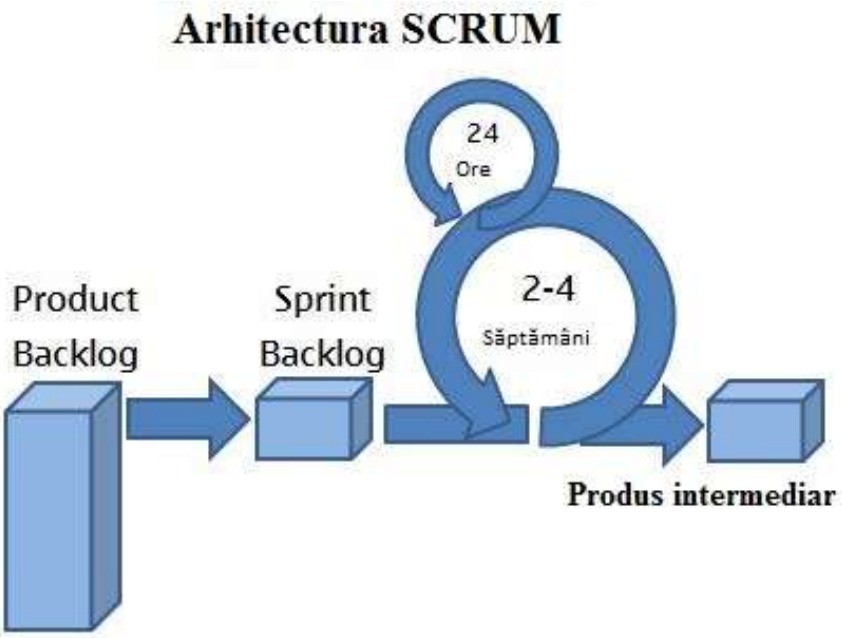
Figura 1 Practicile Programare Extremă [7]

b. SCRUM

Scrum este un tip de management de proiect de tip agil, folosit pentru a furniza iterativ stadii intermediare ale produsului pentru client. Scrum se bazează pe auto-organizare, și pe echipe abile care să livreze produsul treptat. De asemenea, se bazează pe un client, sau proprietar de produse, să îi ofere echipei o listă de caracteristici dorite folosind ca mecanism de prioritate valoarea de business.

Cu ajutorul Scrum, proiectele progresează prin o serie de iterații lunare numite sprinturi (***sprints***).

Scrum este ideal pentru proiectele care are cerințe care se schimbă rapid, sau care apar urgent în timpul proiectării. Munca ce trebuie făcută într-un proiect Scrum este listată într-un Jurnal de Comenzi Nerezolvate ale Produsului (*Product Backlog*), care reprezintă o listă a schimbărilor care se doresc a fi făcute pentru produsul respectiv. La începutul fiecărui sprint este ținută o Întâlnire de Planificare a Sprintului (*Sprint Planning* Meeting) în timpul căruia Proprietarul Produsului (*Product Owner*) prioritizează Jurnalul de Comenzi Nerezolvate și Echipa Scrum selectează sarcinile pe care le poate completa în timpul sprintului care urmează a fi făcut. Aceste sarcini sunt mutate din Jurnalul Produsului în Jurnalul de Comenzi Nerezolvate ale Sprintului (*Sprint Backlog*).



Figură 2 Arihtectura Scrum[8]

În fiecare zi în timpul sprintului se ține o întâlnire scurtă denumite Scrum-ul Zilnic (*Daily Scrum*), care ajută echipa să păstreze traiectoria bună a proiectării. La sfârșitul fiecărui sprint, echipa demonstrează funcționalitățile complete la Întâlnirea de Retrospectivă a Sprintului (*Sprint Review Meeting*).

Se desemnează și un stăpân al Scrum-ului (*ScrumMaster*) care nu este conducătorul echipei, ci rolul lui principal este de a înlătura problemele ce împiedică echipa să atingă țelul sprintului. Astfel, el este un mediator între echipă și influențele care ar putea distrage echipa de la atingerea țelului.

Scrum încurajează mutarea membrilor prin auto-organizare, astfel încât să fie acestia să se alfe în același spațiu fizic și comunicarea verbală între membrii echipei să fie de preferat altor metode.

Metoda Scrum este admite ca provocările fundamentale nu pot fi adresate tradițional – predictivă. În felul acesta, Scrum adoptă o abordare empirică, deci se acceptă faptul că problema nu se poate înțelege sau defini complet, așadar, se concentrează pe maximizarea abilității echipei de a livra rapid și a răspunde eficient la cerințe noi.

Figură 3 Echipa Scrum [9]

Scrum nu este numai un set concret de practici, mai degrabă, şi mai important, este un cadru care asigură transparenţa, şi un mecanism care permite "inspectarea şi adaptare". Scrum lucrează prin a face vizibile disfuncţiile şi obstacolele care au un impact asupra Proprietarului de Produse şi eficacităţii echipei, astfel încât acestea să poată fi abordate. Scrum nu rezolvă problemele, doar le face vizibile și asigură o metodă de explorare a metodelor scurte de rezolvare.

5. Agile Model & Plan Driven Methods

de Vasile Ioana Iuliana

Cu toate că mulţi dintre susţinătorii lor consideră metodele de dezvoltare software de tipul agil şi plan-driven ca fiind opuse, sintetizarea acestor două poate oferi dezvoltătorilor o gamă largă de unelte şi opţiuni.

Ambele abordări, atât cea agilă cât şi cea plan-driven, se bazează pe repsonsabilitate şi pe o interpretare a marginilor radicale. Deşi fiecare abordare are propriul set de caracteristici ale proiectelor, pentru care are performanţe foarte bune, şi mult mai eficiente decât cealaltă, în afara acestor seturi specifice fiecări abordaări, o combinaţie între ele este preferată şi fezabilă.

Putem plasa diversele variante de abordări agile și plan-driven în spectrul metodele care pun preț pe plan. În acest context, termenul „plan" include documentarea evoluării proceselor care implică cerințe scurte și repere de plan, dar și strategii de dezoltare care implică cerințele generale, proiectarea și planurile arhitecturale. În ansamblu, metoda agilă plasează mai multă valoare asupra procesului de planificare decât asupra documentaţiei rezultate, prin urmare aceste abordari adesea par mai putin orientate către planificare decât sunt într-adevăr. Metodele plan-driven se axează mai de graba pe reperele majore de timp și a conținutului lor global, decât pe micro – reperele blocate într-un acord inflexibil.

Atât meoda agilă, cât și cea plan-driven au puncte forte și puncte slabe, iar cea mai bună comparație între cele două se poate face din următoarele presecive cheie: dezvoltatori, clienți, cerințe, arhitectură, remodelare, dimensiune și obiective primare.

Dezvoltatorii

În metodele agile factorii umani critici pentru o dezvoltarea eficientă sunt: amiciţia, talentul, abilitatea şi comunicarea. Lucrul în echipă este foarte important, mai mulţi oameni competenți care lucrează împreuna, negreșit vor dezvolta produse mai bune.. Însă fără oameni competenți, e foarte posibilă obţinerea unei arhitecturi încurcate a produsului. O semnificaţie importantă o are în acest caz inevitabilă statistică ce arată că aproape jumătate dintre dezvoltatorii de software din lume sunt sub medie.

Aceasta nu înseamnă în mod precis că metodele agile necesită omeni cu acelaşi nivel înalt de pricepere. Multe proiecte agile, cât şi cele plan-driven au fost reuşite combinând angajaţi experimentaţi cu juniori. Diferenţa principală este aceea că metodele agile extrag mult din agilitatea lor bazându-se pe cunoaşterea şi înţelegerea tacită existentă în echipă, decât pe scrierea cunoaştinţelor în planuri.

Când cunoaşterea tacită a echipei este suficientă pentru necesitățile ciclului de viaţă al aplicaţiei, lucrurile decurg bine. Dar există de asemenea un risc ca echipa să facă greşeli arhitecturale iremediabile din cauza unor scurte neînţelegeri nerecunoscute în cunoaşterea lor tacită. Metodele plan-driven reduc acest risc învestind în arhitecturi şi planuri a întregului ciclu de viață, şi folosind acestea pentru a uşura revizuirile expertului extern. Făcând aceasta, acceptă totuşi riscul ca o schimbare rapidă va face planurile învechie sau foarte scumpe pentru a fi ţinute la zi.

Clienţii

S-a descoperit că dacă clienţii nu sunt dedicați, cu cunoştinţe, cooperanți, reprezentativi şi împuterniciţi, în general produsele dezvoltate nu realizează tranziţia către o utilizare de succes, chiar dacă acestea satisfac clientul. Metodele agile funcţionează cel mai bine când clienţii sunt dedicați în lucrul cu echipa de dezvoltare, şi când cunoaşterea lor tacită este suficientă pentru a acoperi întreaga aplicaţie. Deşi aceste metode riscă scurte neînţelegeri în cunoaşterea tacită, metodele plan-driven le-ar reduce prin documentaţie şi folosirea revizuirilor arhitecturale ale comitetelor şi revizuirea proiectului de către un expert independent pentru a compensa micile nereguli din partea clientului.

Cerinţe

Abordările agile sunt mai bine aplicate pe mediile turbulente, cu un grad ridicat de schimbare. Conform cu viziunea agilă, organizaţiile sunt sisteme adaptive complexe în care cerinţele sunt mai degrabă emergente, decât pre – specificate. Cu otate aceste, în timp ce principiile din manifestul metodei agile – cum ar fi cel de-al doilea care susţine că schimbarea cerinţelor este binevenită, chiar dacă apare târziu în cadrul dezvoltării – oferă un potenţial ridicat pentru succes, dezvoltatorii pot aplica greşit aceaste reguli, soldându-se cu rezultate dezastruoase.

Metodele plan-driven funcţionează perfect când dezvoltatorii pot determina cerinţele din timp – de exemplu prin realizarea de prototipuri – şi când acesete cerinţele rămân relativ stabile, cu o probabilitate de schimbare în jurul unui mic procent pe lună. În tot mai desele situaţii în care cerinţele se schimbă cu o rată mult mai mare decât cea preconizată, accentul tradiţional plasat pe importanţa definirii unor cerinţe complete, consistente, precise, testabile şi uşor urmărită va prezenta dificultăți, întâlnindu-se probleme majore, dacă nu chiar imposibil de rezolvat, legate de actualizarea cerinţelor. Totuși aceast accent este vital pentru un software stabil şi sigur.

Arhitectura

Valorile metodei agile preferă în primul rând un software funcționabil, decât o documentație completă, și susține simplicitatea: maximizarea cantității de lucru nefinalizat. Acest principiu poate fi interpretat în mai multe moduri. Majoritatea sunt în regulă, însă unele interpretări pot cauza probleme. De exemplu, bazat pe noțiunea YAGNI: „*Nu vei avea nevoie de lucrul acesta*.”(***Y**ou **A**ren't **G**oing to **N**eed **I**t*) , programarea extremă pledează pentru munca în plus pentru a scăpa de trăsăturile arhitecturale ce nu sunt suportate de versiunea curentă a sistemului. Aceasta abordare funcţionează bine când cerinţele viitoare sunt imprevizibile la scară largă. Pe de altă parte, în situațiile în care cerințele viitoare sunt previzibile, această practică nu numai că aruncă un suport arhitectural valoros pentru ele, dar de asemenea crează probleme cu clienții care doresc ca dezvoltatorii să considere că prioritățile lor şi evoluția cerințelor merită această ajustare.

În ceea ce privește metodele plan-driven care îmbrățișează arhitectura și documentația de proiectare complexe și dificile va întâlni probleme grele, și chiar iremediabile în încercarea de a ține pasul cu schimbarea rapidă a cerințelor. Pe de altă parte, dacă arhitectura anticipează și se

adaptează schimbării cerințelor, aceste metode bazate plan-driven pot păstra chiar aplicații valorând milioane în linia bugetului și a planului de dezvoltare.

Remodelarea

Cu dezvoltatori competenți și sisteme mici, supoziția că remodelarea este în esență gratuită este considerată validă. În aceste circumstanțe abordarea YAGNI („*Nu o să ai nevoie de lucrul acesta*") are un risc scăzut de eșec. ToușI, dovezi empirice arată că lucrând cu dezvoltatori mai puțin de succes, efortul de remodelare crește cu numărul de cerințe și versiuni. Pentru sistemele foarte mari, o analiză indică faptul că 20% din problemele ce determină 80% din reoperare provin intens din „căderile arhitecturale"[10], precum discontinuitățile arhitecturale la ajustarea performanței, toleranța greșelilor sau probleme de securizare, în care remodelarea n-ar putea în niciun fel să îndrepte situația.

Dimensiunea

Metodele agile nu sunt recomandate echipelor de dimensiune mare, deși există și cazuri ocazionale de succes în care echipa proiectului era formată din aproape 250 de persoane. Metodele agile funcționează cel mai bine pentru proiecte mici, deoarce pentru echipe de peste 20 de persoane, efortul pentru o coordonare de succes muncii în echipă este mult prea mare.

Metodele plan-driven se pliază mai bine pe proiectele mai mari. Iar acest tip de model, cu birocrația sa care necsită foarte mult timp doar pentru autorizații, nu va fi deloc eficient pentru proiectele mici.

Obiective principale

Primul principul al manifestului agil spune că „*Prioirtatea principală este satisfacerea clientului printr-o livrea timpurie și continuă de software de valoare*"[11]. Livrarea timpurie și continuă este compatibilă cu țelurile unui sistem mic dezvoltat de persoane competente. Dar supra-concentrarea pe rezultate timpurii în sisteme mari poate duce la reprelucrări majore atunci când arhitectura nu este proporționată corespunzător. În acest caz, este nevoie de o bună planificare.

Metodele plan-driven sunt necesare în proiecte software de siguranță ridicată, aplicarea metodelor agile pe astefel de proiecte este o provocare. Un alt set de obiective majore pentru metode tradiționale plan-driven a fost predictibilitatea, repetabilitatea, și optimizarea. Dar într-o lume care implică schimbări frecvente, chiar radicale, acestea nu pot fi un obiectiv prea bun. Din fericire, s-au dezvoltat modalități de migrare spre o adaptabilitate mai mare.

	Metodele Agile	Metodele plan-driven
Dezvoltatori	Agili, cu cunoștințe avansate, co-locați și cooperanți	Orientați pe plan, cu abilități adecvate, au acces la cunoștințe externe
Clienți	Dedicați, cu cunoștințe, co-locați, cooperanți, reprezentativi și împuterniciți	Cu acces la cunoștințe, cooperanți, reprezentativi și împuterniciți
Cerințe	Foarte emergente, cu schimbări rapide	Cunoscute din timp, foarte stabile
Arhitectură	Proiectare pentru cerințele curente	Proiectare pentru cerințele curente și previzibile
Remodelare	Necostisitoare	Costisitoare
Dimensiune	Echipe și proiecte mici	Echipe și proiecte mari
Obiective primare	Valoare rapidă	Siguranță ridicată

Tabel 1 Comparație între metodele agile și cele plan-driven [12]

Echilibru între agilitate și disciplină

În particular, pentru sector de servicii electronice, comaniile cu o bază de clienți foarte mare, nu are nevoie doar de schimbări valoroase sau doar siguranță ridicata, ci au nevoie de amândouă. De aceea, este necesar o amestecare a celor două tipuri de dezvoltare software.

Sunt câteva metode de îmbinare a celor doua metode. O altă alternativă ar fi managementul riscului care poate să echilibreze agilitate și disciplina prin răspunsul la întrebări ca: „*Cât de mult este suficient*?”, „*De câtă planificare este nevoie?*. Pentru acest concept este necesară determinarea expunerii la risc a unei acțiuni.

Atât metoda agilă, cât și cea plan-driven fac parte din spectrul planificărilor. În ciuda unor anumite terminologii extreme, fiecare este parte dintr-un centru responsabile, decât din margini radicale. De asemenea, metodele agile efectuează un servicu valoros când îndreaptă programatorii rebeli spre o zonă de responsabilitate.

Amândouă au în comun faptul că pentru anumite tipuri de caracterisitci ale proiectului acesta lucrează la nivelul cel mai bun, dar și unele în care are dificultăți. Abordările hibride care combină cele două metode sunt fezabile și necesare pentru proiecte care combină caracteristicilor tipului de proiect agile, și tipului plan-driven. O analiză poate să determine echilibrul între caracteristici, obținându-se întreteserea corectă între metoda agilă și cea plan-driven.

Cu toate că tehnologia informației tinde spre caracteristici specific agile, precum cerințe emergente și rapide, dar fiabilitatea ridicată cere să se aplice și măsuri din spectrul metodelor plan-driven. În felul acesta, se așteaptă ca metodele agile să fie folosite și în proiecte cum ar fi: serviciile financiare, comerțul electronic, controlul de trafic aerian, sisteme medicale, etc.

6. Agile Model Driven Development (AMDD)

de Bălănescu Diana

Intr-o traducere mai liberă a titlul de mai sus am obţine Modelul Agil îndreptat către Dezvoltare. Prin urmare aşa cum sugerează şi numele, AMDD este o versiunea agilă a Modelului Driven Development (MDD) (Model bazat/îndreptat către dezvoltare). MDD propune o abordare asupra dezvoltării den produse software complxe în care modelele sunt create înainte de scrierea codului sursă. Aşdar acest model îngăduie mai întâi o şansă de a trece prin şi de a înţelege mai bine problemele mai complexe înainte de a dezvolta codul. MDD nu reprezintă o metodologie completă ci mai degrabă o abordare generică ce poate fi aplicată procesului de dezvoltare software pentru a beneficia de avantaje precum: reutilizabilitate şi portabilitate. Un exemplu [13] al MDD este standardul Model Driven Arhitecture (MDA), aparţinând de Object Management Group (OMG). Prin MDD se adopta o abordare seriala de dezvoltare însă este posibilă şi o abordare iterativă cu MDD.

Aşa cum este de aşteptat AMDD reprezintă o combinare a principiilor şi valorilor agile cu modelul MDD. Aceste două tehnologii sunt însă destul de diferite. În timp ce MDD este un model centrat ce pune dezvoltarea înainte de cod, dezvoltarea agilă apreciază codul executabil şi ezită atunci când vine vorba de modele complexe care necesită o documentare extensivă. AMDD poate fi descris totuşi ca un compromis inteligent creat în aşa fel încât să se bucure de beneficiile ambelor tehnologii. Este posibilă fie integrarea principiilor MDD şi a uneltelor de dezvoltare în procese software agile existente, fie introducerea agilităţii în metodologiile curente bazate pe MDD [14].

Prin urmare AMDD propune ca înainte de a scrie codul sursă să se creeze modelul. Insă în locul creerii unor modele extinse să se creeze modele agile suficient de bune pentru a conduce eforturile generale de dezvoltare. AMDD devine astfel o strategie esentiala în dezvoltarea de software agil.

În figura de mai jos este descris un ciclu de viaţă la nivel inalt al AMDD pentru eliberarea unui sistem. Diagrama reprezentată are următoarea semnificaţie:

Fiecare dreptunghi reprezintă o activitate de dezvoltare. Imaginea include două activităţi principale, prezentarea cerintelor iniţiale şi prezentarea arhitecturii iniţiale. Cele două activităţi au loc în timpul primei itaraţii notată Iteraţia 0 sau Itteration 0. Înainte de a incepe propriu-zis partea de dezvoltare AMDD implementează aceste două activităţi ce corespund momentului iniţial. Urmatoarele activitati prezente în ciclu sunt - iteratii de modelare, model storming, recenzii şi implementare. Acestea pot aparea în timpul oricărei iteraţii din procesul de dezvoltare..

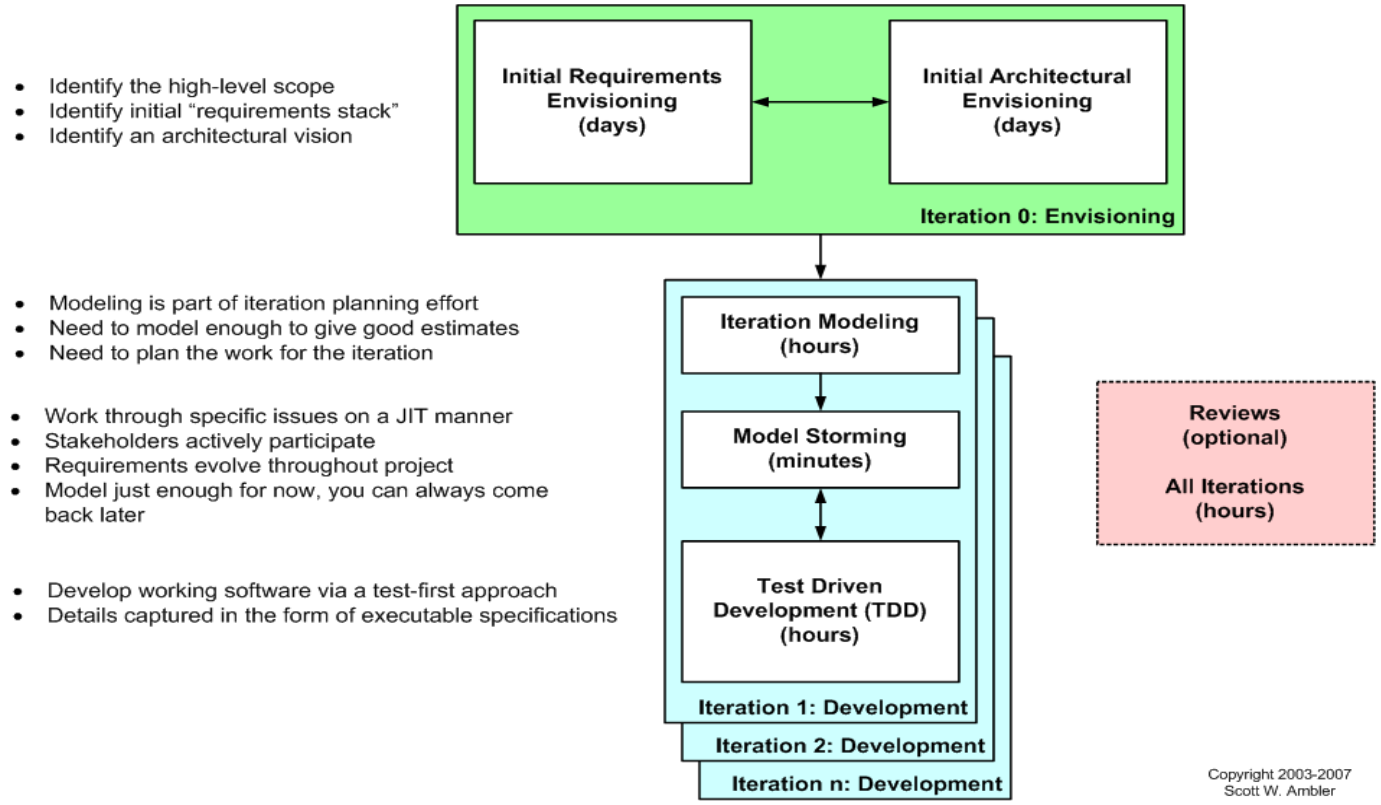


Figura 4: Agile Model Driven Development (AMDD) [15]

Din punct de vedere al designului abordarea AMDD din figura 4 este foarte diferită de abordările traditionale de dezvoltare (MMD) în care se crea mai întai un model de design şi apoi plecând de la acesta codul sursă. Cu AMDD modelarea este mai simplă şi mai puţin utilizată, codificare însă se execută destul de frecvent. Există posibilitatea iteraţiei înapoi atunci când este nevoie. Eforturile de design sunt impărtite între activităţile de modelare şi cele de codare iar proiectarea face parte din eforturile de implementare. Acest lucru a rămas valabil pentru multe proiecte traditionale.

Un alt mod de a privi AMMD este ca o combinaţie între cele mai bune practici, valori şi pricipii ale modelelor MDD şi Agile: Simplitate, Generalitate, Coerenţă, Flexibilitate, Rapiditate, Învăţare, Eficacitatea Costului, Managementul datelor, Analiza problemelor, Teste Automate, Verificare/Validare, Unelte pentru suport.

In functie de aceste principii AMDD este abordat din mai multe puncte de vedere. Acestea ar fi: Sage, Hybrid MDD, MDD-SLAP, High Level Life Cycle. Detalii despre aceste procese pot fi găsite aici [16]. În tabelul de mai jos este prezentat modul în care cele 4 îmbină principiile Agile Modeling, MDD şi AMDD. În acest tabel se urmăreşte evaluarea proceselor AMMD, afişarea rezultatelor şi posibilitatea de a le compara una cu cealaltă.

AMDD Process / Criterion		Sage	Hybrid MDD	MDD-SLAP	High-Level Life Cycle
Agility Evaluation Criteria	Flexibility	☑	☑	☑	☑
	Rapidity	☑	☒	☑	☑
	Learning	☒	☑	☑	☒
	Responsiveness	☑	☒	☑	☑
	Cost Effectiveness	☒	☑	☒	☒
MDD Evaluation Criteria	Metadata Management	☑	☑	☒	☒
	Automatic Test	☒	☑	☑	☑
	Tool Support	☑	☑	☑	☒
	Problem Domain Analysis	☑	☑	☑	☒
	Verification / Validation	☒	☒	☒	☒
AMDD Evaluation Criteria	Smoothness	☒	☒	☑	☑
	Coherency	☒	☑	☑	☑
	Simplicity	☑	☒	☑	☑
	Generality	☒	☑	☑	☑

Figura 5: Evaluarea Abordărilor AMDD [17]

AMDD reprezintă o tehnologie promiţătoare şi un concept practic deosebit, însă în momentul de faţă îi lipseşte maturitatea. Combinarea celor două tehnologii de modelare a adus, cum am menţionat mai sus, o serie de variante modelului AMDD. Acesta nu este un lucru rău deoarece dezvoltatorul îşi poate alege varianta care i se potriveşte cel mai bine.

7. Când folosim Agile Modeling

de Vasile Ioana Iuliana

În ultima perioadă, interestul pentru metodele agile a crescut simțitor, dar odată cu acestea au apărut și îndoieli asupra eficienței acestor metode. Din nefericire, nu există încă date statistice concrete despre acest subiect, dar au început să se țină seminarii în care să se cumuleze experiențele persoanelor care au lucrat cu metode agile și să se evidențieze problemle cu care s-au confruntat. Astfel, s-au pus cap la cap niște orientări despre utilizarea eficientă a metodelor agile.

Când o organizație se decide să adopte pentru dezvoltarea proiectelor agile, ar trebui să țină seama că cerințele unor proiecte contrazice principiile metodelor agile, de ușurința cu care se poate adopta aceasta, dar și de cum se va face managementul proiectelor agile.

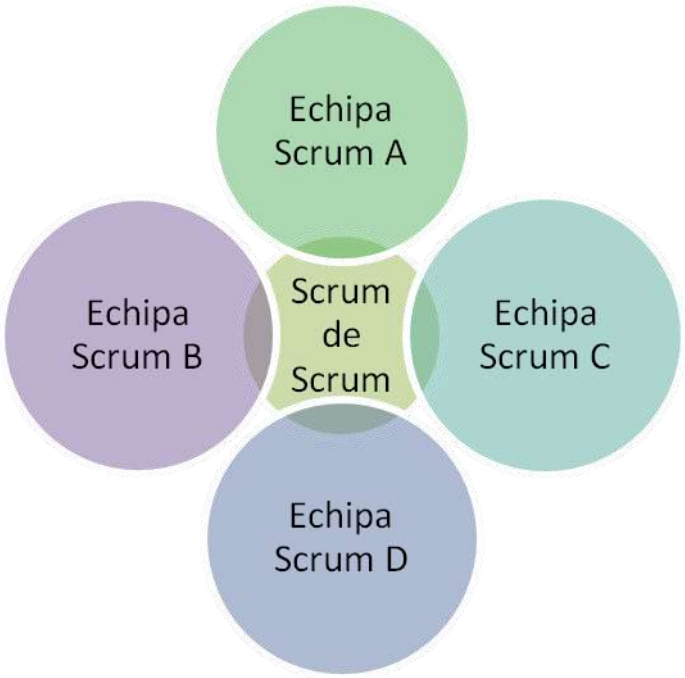
a) Selectarea proiectelor adecvate pentru metodele agile

Dimensiunea Proiectului este cel mai important factor care determină dacă o metodă agilă se poate aplica. S-au colectat date din experiențele anterioare de folosire a metodelele agile pentru a se determina aplicabilitatea lor. Astfel, există multe date despre folosirea metodelor agile în echipe de până la 12 persoane, câteva date despre echipe de în jur de 25 de persoane, foarte puține date despre echipe de până la 100 de persoane și informații izolate de echipe mai mari de 100 de persoane.

Cu cât dimensiunea echipei crește, coordonarea interfețelor devine o problemă dominantă. Ideea de comunicare față – în – față a metodelor agile se împarte, devenind foarte dificilă și complexă dacă echipa depășește 20 – 40 de persoane. De aceea, este important să se găsească și să se aplice strategii speciale pentru o comunicare eficientă între mai multe persoane.

O strategie de scalare este organizarea unui proiect mare in echipe de echipe. S-a folosit ideea de „Scrum de Scrum”. Aceasta prevedea că fiecare echipă să fie completată cu membri din mai multe linii de produse, în scopul de a crea o înţelegere pe scară largă a proiectului în ansamblul său. Reuniuni scurte, dar regulate de sub – echipa proiectului – încrucișat (persoane de rang superior sau zone tehnice comune) trebuiesc a fi ținute regulat pentru coordonarea proiectului și echipele de echipe existente pentru dezvoltarea proiectului. Este important să se țină o echipă – nucelu responsabilp pentru arhitectură și standarde sunt necesare pentru această configurație de lucru. Acestă echipă va lucra activ cu sub – echipele existente și le va coordona lucrul.

Metode de coordonarea echipelor multiple include: conferințe anuale pentru alinierea interfețelor, rotația angajaților între echipe în stagii de trei luni.



Figură 6 Echipa Scrum de Scrum[18]

Un subiect dezbătut cu privire la eficacitatea metodelor agile este **personalul echipei**. Acesta trebuie să fie experimentat şi competent în următoarele sensuri:

- să aibă experineță reală în domeniul thenologic
- să aibă experiență cu sisteme agile
- să fie sociabil şi abilități de comunicare excelente

Totuşi, experienţa în dezvoltarea de sisteme software de orice tip e mai importantă decât experienţa cu metodele agile. De aceea, este foarte folositoare rotirea personalului între sub-echipe pentru a învăţa unul de la altul.

Folosirea metodelor agile în proiecte care necesită **cerinţe critice, fiabilitate şi securitate ridicată** nu este întodeauna indicată. Doar dacă acestea au fost cerute explicit la începutul proiectului şi se pot derula testări planificate. În general, metodele agile se pliază mai bine pe aplicațiile la care poat fi construite doar „schelet-ul" foarte rapid, ca cele care, de-a lungul vieții, stau mai mult timp în mentenanță.

Pentru proiectele critice metodele agile care pun preț pe testare, în particular practica XP de dezvoltare bazată pe testare (*test – driven developemnt XP practice*), sunt cheia succesului. Astfel, datorită acestor multiple teste care trebuiesc trecute înainte de lansare, proiectele dezvoltate se pot conforma unor cerințe strice sau de siguranță. Clienții pot concepe teste de aceptanță care să măsoare cerințele non – funcționale, dar pot fi mai dificile şi pot avea cerințe mai sofisticate legate de mediul de dezvoltare.

Cu toate acestea, este mai ușor să adresezi probleme critice dacă acestea sunt furnizate de client explicit și timpuriu. Practica „răspundere responsabilă la schimbare” specific agilă sugerează că este nevoie de investigare a sursei schimbării și ajustare confromă a soluției, nu numai răspuns și continuare. Dacă se aplică în mod corect, strategia „testare prima dată” satisface cerințele critice.

b) Adoptarea metodelor agile

Metodele agile sunt foarte ușor de adoptat în cadrul unei companii, deoarce nu este nevoie de foarte multă instruire formală înainte ca o echipă să le poată folosi, spre deosebire de alte metode tradiționale. În metodele agile se pune mai mult accentul pe abilități, decât pe învățarea metodelor agile, care de foarte multe ori poate fi autodidactă.

c) Managementul proiectului

Cea mai bună metodă de învățare din experiențele anterioare este analiza proiectelor deja completate din perspectiva **factorilor de succes**. Cei mai imporanți trei facotri identificați sunt cultura, oamenii și comunicarea.

Cultura este importantă în control și adaptare a membrilor echipelor. Dacă aceasta nu există, o organizație nu poate fi agilă. Cultura trebuie să fie în definitiv un suport pentru buna negociere, care este o parte importantă a metodelor agile.

Cum s-a precizat și mai sus, este important ca membrii echipelor agile să fie competenți. În felul acesta, sunt necesari mai puțin oameni. Acestia trebuie să fie de încredere, pentru că dacă organizația se îndoiește de capacitățile lor de a face decizii, acesti cu desăvârșire nu vor fi motivați, ci din contra, munca lor se va degrada.

Comunicarea bună se face față – în – față după cum s-a menționat mai sus, de accea se recomandă ca echipele să fie locate în spațiu apropriat și să se programeze în perechi. Asta nu doar îmbunătățește comunicarea, și vor fi mai puține interpretări eronate, dar în același timp comunicarea va fi mai rapidă.

Acesti factori de succes trebuiesc implementati cu grijă pentru ca ei să funcționeze. Dar, pe lângă cei trei menționați mai sus, în cazul metodelor agile, feedback-ul provenit de la client este și el un factor de succes. Dacă se menține o interacțiune bună cu acesta, proiectul se va mișca în direcția bună, nu vor exista întoarceri datorită interpretărilor greșite, deci, proiectul se va finaliza mai rapid.

O altă parte critică în managementul proiectului este să se identifice din timp **semenele de avertizare** care pot indica anumite direcții greșite. Pentru aceasta, întâlnirile zilnice sunt foarte utile în rezolvarea problemelor. Datorită strategiei de comunicare deschisă, personalul poate să aducă la cunoștință mai repede problemele eventual apărute. Astfel, se poate observa rapid în astfel de întâlniri un moral scăzut al echipei sau producerea unei documentații inutilă ce pot indica probleme în decursul proiectului. Un al semn de averitzare este atunci când echipa rămâne în urmă cu planificarea interațiilor. De aceea, este important să existe iterații mai mici și mai frecvente, deoarce se poate monotoriza mult mai bine aceste avertizări.

Un principiu – cheie al metodologiilor agile (în special în XP) este **refactorizarea**. Aceasta înseamnă îmbunătăţirea proiectării codului existent, fără a modifica funcţionalitatea sistemului. Diferite forme de refactorizare implică: simplificarea situaţiilor complexe, abstractizare unor soluţii comune în cod reutilizabil, precum şi eliminarea de cod duplicat.

Refactorizarea arhitecturii sistemului nu este foarte folositoare atunci când aceasta afectează toate părțile interne și externe interesate. În schimb, abordarea ar trebui să fie refactorizare frecventă de cod cu dimensiuni rezonabile, păstrând în domeniul de aplicare stabilit, astfel încât modificările să fie mai mult locale.

Documentaţia de produs şi de proiect este un subiect care a atras multă atenţie în discuţiile despre metodele agile, în special din privința necesității și cantității acesteia. Documentaţia poate fi privită ca o forma slaba de comunicare, dar uneori este necesară pentru a păstra informaţiile critice de-a lungul timpului. Cel mai frecvent, politica organizaţiilor cerere documentaţie mai mult decât este necesară. Scopul, însă, ar trebui să fie comunicarea eficientă şi documentaţia ar trebui să fie una dintre ultimele opţiuni în îndeplinirea acestui obiectiv. Totuși, un proiect documentat bine face diagnosticarea probleme mai uşoră pentru un expert extern. Cel mai util pentru metodele agile este ca documentaţiei să I se atribuie un cost şi amploarea ei să fie determinată de către client (cu excepţia documentaţiei interne).

8. Concluzii

de Vasile Ioana Iuliana

Mai nou, produsele software, și în general toată industria tehnologiei informației, a luat o direcție în care reînnoirea acestora se realizează la intervale foarte scurte de timp. Aceasta se datorează nevoii de satisfacere a utilizatorilor care așteaptă mereu ceva inedit, a dezvoltărilor echipamentelor harware, dar și a concurenței în domeniu, care dă un dinamism aparte industriei software. Un bun exemplu pentru acest dinamism est piața browser-elor web: apariția browser-ului Chrome a aduc o concurență mare browser-ului Firefox; Chrome are o politică de reînnoire a versiunii anterioare într-un timp scurt, iar Firefox, pentru a ține pasul a introdus o politică asemănatoare, iar de atunci numărul versiunilor Firfox a crescut foarte mult față de perioadele precedente. Metodele tradiționale de management ale produselor software nu sunt foarte eficiente, în aceast secol al vitezei, în care cerințele sunt emergente, schimbătoare. Aici intervin metodele de dezvoltare software agile.

Metodele agile, prin principile și practicile lor, reușesc să țină pasul cu cerințe de sistem ale produselor schimbătoare, emergente, neprevizibile. Astfel, se pune accentul pe satisfacerea clientului prin livări timpuri și continue de cod valoros, nu sunt împotriva schimbărilor, ci din contră, le imbrățișează indiferent de momentul apariției lor. O mare importanță cade și pe buna comunicare, atât între client și dezvoltatori, dar și în intermediul echipei, care trebuie să fie constituită din oameni competenți, sociabili, care lucrează bine în echipă și comunică bine, motivați și cooperanți, iar în cadrul echipei auto-organizarea este esențială. Așadar echipa este un factor important în reușita unui proiect dezvoltat prin procese agile. Se punctează astfel necesitatea dezvoltării angajaților nu numai din punct de vedere tehnic, al cunoștințelor pe care le posedă, dar și pe dezvoltarea inteliegenței lor emoționale, care face integrarea lor mai bună în echipa, un factor important în succesul proiectelor. Dezvoltarea metodelor agile, pune în

evidență o deficiență a metodelor tradiționale de învățământ: nu se pune accent și pe dezvoltarea emoțională corectă a elevilor. De accea, mulți dintre acești elevi ajung să fie angajați care nu pot servi corect în dezvoltarea metodelor agile: nu reușesc să comunice eficient, nu reușesc să programeze în grup cu altă persoană, nu reușesc să observe când un co-echipier are nevoie de ajutor, nu reușesc să înteleagă ceea ce clientul încearcă să spună, etc. Importanța inteligenței emoționale în rândul membrilor echipei este poate chiar mai importantă decât cunoștințele și experiența lor. Simplicitatea este o altă valoare a metodelor agile, pentru că este mai usor să adaugi caracteristici noi și simple, decât să dezvolți de la început ceva complicat. Posibilitatea de a greși este mult mai mică în acest caz.

Cu toate acestea, chiar dacă ritmul de dezvoltare este mai accelerat, securitatea nu trebuie niciodată dată la o parte. Dar, metodele agile nu se orientează spre o creștere a securității și a fiabilității, de aceea au început să se dezvolte mdetode agile hibride, care combină rapiditatea și emergența metodelor agile cu securitatea și arhitectura bine stabilită a metodelor plan-driven. Printre acestea se numără și metoda agilă orientată pe model. Chiar unul din creatori manifestului agile, Scott Ambler, afirma acest deficit al metodelor agile simple în proiectele care necesită asigurări de securitate și fiabilitate: „*Ar fi foarte «șmecher» să se aplice modelare agile pe sistemele cu viață critică*”[19].

Așadar, metodele agile sunt foarte eficiente, dar doar pe anumite tipuri de proiecte cu diverse caracteristici. În alte împrejurări, este posibil ca aceste metode să nu fie foarte profiabile. Deci, atunci când se dorește începerea unui proiect software este de dorit analiza cerințelor și posibilităților financiare și umane înainte de algerea metodei de dezvoltare a acelui proiect.

Bibliografie:

Capitolele 1,2,3,6

- Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process, Scott Ambler
- Agile Model Driven Development: An Intelligent Compromise, Reza Matinnejad Information and Communication Technology Institute Isfahan University of Technology Isfahan, Iran
- http://www.agilemodeling.com/essays/introductionToAM.htm
- http://en.wikipedia.org/wiki/Agile_Modeling

Capitolele 4,5,7,8

- *„Get Ready for Agile Methods, with Care,Barry Bohem*”, *IEEE Ianuarie 2002, tabel 1, pagina 68*
- „*Empirical Findings in Agile Methods*”, Mikael Lindvall, Vic Basili, Barry Boehm, Patricia Costa, Kathleen Dangle, Forrest Shull, Roseanne Tesoriero, Laurie Williams, and Marvin Zelkowitz
- http://epf.eclipse.org/wikis/xp/, accesat 12.12.2011
- http://epf.eclipse.org/wikis/scrum/, accesat 12.12.2011
- http://en.wikipedia.org/wiki/Scrum_%28development%29, accesat 21.12.2011

Referințe

[1] http://www.agilemodeling.com/essays/introductionToAM.htm

[2] *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, Scott Ambler

[3] *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, Scott Ambler

[4] *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, Scott Ambler

[5] *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, Scott Ambler

[6] Kent Beck 2000. *Extreme Programming Explained* Addison-Wesley Publishing Co

[7] www.Xprogramming.com , accesat 12.12.2011

[8] http://www.expertprogrammanagement.com/wp-content/uploads/2010/08/scrum-framework-diagram.jpg, *accesat 12.12.2011*

[9] http://www.projectperfect.com.au/images/info/agile.JPG, *accesat 12.12.2011*

[10] *Empirical Findings in Agile Methods*, Mikael Lindvall, Vic Basili, Barry Boehm, Patricia Costa, Kathleen Dangle, Forrest Shull, Roseanne Tesoriero, Laurie Williams, Marvin Zelkowitz

[11] http://agilemanifesto.org/principles.html, acesat 12.12.2011

[12] *Get Ready for Agile Methods, with Care,Barry Bohem, IEEE Ianuarie 2002, tabel 1,pagina 68*

[13] K. Beck, et al., „*Principles behind the Agile Manifesto*”, Agile Alliance, 2001

[14] J. Ralyté, R. Deneckère, and C. Rolland, "*Towards a Generic Model for Situational Method Engineering*", 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003), Klagenfurt/Velden, Austria, pp. 95-110, 2003.

[15] http://www.agilemodeling.com/essays/introductionToAM.htm

[16] Agile Model Driven Development: An Intelligent Compromise, Reza Matinnejad Information and Communication Technology Institute Isfahan University of Technology Isfahan, Iran

[17] Agile Model Driven Development: An Intelligent Compromise, Reza Matinnejad Information and Communication Technology Institute Isfahan University of Technology Isfahan, Iran

[18] http://toolsforagile.com/blog/wp-content/uploads/2010/08/scrum_of_scrums.png, accesat 21.12.2011

[19] Scott Ambler, „*When Does(n't) Agile Modeling Makes Sense*?”