Dezvoltarea agilă de software

Nicolae Sfetcu

16.08.2020

Sfetcu, Nicolae, "Dezvoltarea agilă de software", URL = https://www.telework.ro/ro/e-books/dezvoltarea-agila-de-software/

Email: nicolae@sfetcu.com

Text online: https://www.telework.ro /ro/dezvoltarea-web-agila/, https://www.telework.ro /ro/filozofia-dezvoltarii-web-agile/, https://www.telework.ro /ro/metode-de-dezvoltare-web-agile/
Extras din:
Sfetcu, Nicolae, "*Proiectarea, dezvoltarea şi întreţinerea siturilor web*", pg. 5-7, MultiMedia Publishing (2014), ISBN 978-606-033-075-2, URL = https://www.telework.ro /ro/e-books/proiectarea-dezvoltarea-si-intretinerea-siturilor-web/

Dezvoltarea software agilă este un grup de metode de dezvoltare software în care cerinţele şi soluţiile evoluează printr-o colaborare între echipe auto-organizate, inter-funcţionale. Aceasta promovează planificarea adaptivă, dezvoltare evolutivă, livrarea rapidă, îmbunătăţirea continuă, şi încurajează răspunsul rapid şi flexibil la schimbări. Este un cadru conceptual care se concentrează pe furnizarea de software funcţional cu un minimum de muncă.

Agile Manifesto, manifestul care a enunţat pentru prima dată conceptele care stau la baza dezvoltării agile, a introdus termenul în 2001.

Principiile agile

Agile Manifesto se bazează pe douăsprezece principii:

* Satisfacţia clientului prin livrarea rapidă de software util
* Accent pe cerinţele de schimbare, chiar şi într-o etapă târzie a dezvoltării
* Software funcţional este livrat frecvent (săptămâni, mai degrabă decât luni)
* Cooperarea strânsă, zilnică, între oamenii de afaceri şi dezvoltatori
* Proiectele sunt construite în jurul persoanelor motivate, care ar trebui să fie de încredere
* Conversaţia faţă în faţă este cea mai bună formă de comunicare (co-locaţie)
* Software funcţional este măsura principală a progresului
* Dezvoltarea durabilă, în măsură să menţină un ritm constant
* O atenţie continuă pentru excelenţă tehnică şi design bun
* Simplitatea - arta de a maximiza volumul de muncă în lucru - este esenţială
* Echipe auto-organizate
* Adaptarea periodică a circumstanţelor în schimbare

Prezentarea generală

Există mai multe metode specifice de dezvoltare agilă. Cele mai multe promovează dezvoltarea, munca în echipă, colaborarea, şi adaptabilitatea procesului pe tot parcursul ciclului de viaţă al proiectului.

Iterativ, incremental şi evolutiv

Cele mai multe metode agile descompun sarcinile în etape mici, cu planificare minimă şi care nu implică în mod direct o planificare pe termen lung. Iteraţiile sunt termene scurte care durează de obicei între una şi patru săptămâni. Fiecare iteraţie implică o echipă inter-funcţională de lucru în toate funcţiile: planificare, analiza cerinţelor, proiectare, codificare, unitate de testare, şi testarea de acceptare. La sfârşitul iteraţiei are loc o demonstraţie a produsului în faţa beneficiarului. Acest lucru minimizează riscul global şi permite proiectului să se adapteze la schimbările repede. O iteraţie s-ar putea să nu adauge suficientă funcţionalitate pentru a justifica o lansare pe piaţă, dar scopul este de a avea o lansare disponibilă (cu defecte minime) la sfârşitul fiecărei iteraţii. Iteraţii multiple ar putea fi necesare pentru a lansa un produs sau noi caracteristici.

Comunicare eficientă şi faţă-în-faţă

Indiferent de disciplinele de dezvoltare necesare, fiecare echipă agilă va conţine un reprezentant al clientului. Această persoană este numită de către părţile interesate să acţioneze în numele lor şi face un angajament personal de a fi disponibilă pentru dezvoltatori pentru a răspunde la întrebări în timpul iteraţiei. La sfârşitul fiecărei iteraţii, părţile interesate analizează progresele şi re-evaluează prioritatile, pentru a optimiza rentabilitatea investiţiei şi asigurarea alinierii cu nevoile clientului şi obiectivele companiei.

În dezvoltarea de software agil, un *radiator de informaţii* este un afişaj fizic vizibil (în mod normal mare) situat vizibil într-un birou. Acesta prezintă un rezumat actualizat al stării proiectului

software sau a altui produs aflat în lucru. Numele a fost inventat de către Alistair Cockburn, şi este descris în cartea sa din 2002, *Agile Software Development*. Alternativ, poate fi utilizat un indicator luminos de construcţii pentru a informa o echipă despre starea actuală a proiectului lor.

Buclă de feedback şi ciclu de adaptare foarte scurte

O caracteristică comună în dezvoltarea agilă sunt întâlnirile zilnice privind evoluţia. Într-o scurtă şedinţă, membrii echipei raportează între ei ce au făcut în ziua precedentă, ce intenţionează să facă în ziua respectivă, şi ce obstacole întâmpină.

Focalizare pe calitate

Instrumente şi tehnici specifice, cum ar fi integrarea continuă, unitate de testare automatizată, programare pereche, dezvoltare condusă prin teste, modele de design, design în funcţie de domeniu, reingineria codului, şi alte tehnici, sunt adesea folosite pentru a îmbunătăţi calitatea şi a spori agilitatea proiectului.

Filozofia dezvoltării agile de software

Comparativ cu ingineria software tradițională, dezvoltarea agilă se adresează în principal la sistemelor complexe și proiectelor cu caracteristici dinamice, nedeterministe și non-lineare, unde estimările exacte, planurile stabile și previziunile sunt de multe ori greu de obţinut în stadii incipiente, si marile proiecte şi aranjamente prestabilite pot provoca, probabil, o mulțime de pierderi, nefiind economice. Aceste argumente de bază și experiențele prețioase din industrie învățate în ani de succese și eșecuri au ajutat la favorizarea modelului agil de adaptare, iterativ și dezvoltare evolutivă.

Adaptiv vs. predictiv

Metodele de dezvoltare se găsesc într-un continuum de la *adaptiv* la *predictiv*. Metodele agile se află pe partea *adaptivă* a acestui continuum. O idee de metodă de dezvoltare adaptivă este abordarea de tip "Rolling Wave" în planificarea programului, care identifică repere, dar lasă flexibilitatea căilor de a le atinge, și, de asemenea, permite chiar schimbarea etapelor. Metodele adaptive se concentrează pe adaptarea rapidă la realitățile în schimbare. În cazul în care este nevoie de modificarea proiectului, o echipă de adaptare îl schimbă. O echipă de adaptare va avea dificultati în a descrie exact ce se va întâmpla în viitor. Cu cât este mai îndepărtat termenul, cu atât mai vagă va fi metoda de adaptare va fi cu privire la ce se va întâmpla la acea dată. O echipă de adaptare nu se poate raporta exact la ce sarcini se vor îndeplini săptămâna viitoare, ci doar ceea ce se intenţionează pentru luna viitoare. Când e întrebată despre o lansare de peste șase luni de acum încolo, o echipă de adaptare ar putea să vorbească doar despre angajamentul de lansare, sau privind valoarea așteptată față de cost.

Metoda *predictivă*, în schimb, se concentrează pe analiza și planificarea viitorului în detaliu și luarea în considerare a riscurilor cunoscute. La extreme, o echipă de predicție poate raporta exact ce opțiuni și sarcini sunt planificate pentru întreaga durată a procesului de dezvoltare. Metodele predictive se bazează pe o analiză eficientă a fazei incipiente și, dacă acest lucru merge foarte rău, proiectul ar putea avea dificultăți în schimbarea direcției. Echipele de predicție vor institui de multe ori un panou de control cu schimbările pentru a se asigura că numai modificările cele mai valoroase sunt luate în considerare.

Analiza de risc pot fi folosită pentru a alege între metodele adaptive (*agile* sau *bazate pe valori*) și predictive (*bazate pe planificare*). Barry Boehm și Richard Turner sugerează că fiecare parte a continuumului are propriul teren, după cum urmează:

Zone adecvate pentru diferite metode de dezvoltare:

Metode agile	**Metode planificate**	**Metode formale**
Criticitate redusă	Criticitate mare	Criticitate extremă
Dezvoltatori seniori	Dezvoltatori juniori (?)	Dezvoltatori seniori
Cerințele se schimbă des	Cerintele nu se schimbă des	Cerințe limitate, opţiuni limitate
Număr mic de dezvoltatori	Număr mare de dezvoltatori	Cerințe care pot fi modelate
Cultură care răspunde la schimbare	Cultură care necesită ordine	Calitate extremă

Iterativ vs cascadă

Una dintre diferențele dintre agil și cascadă este că testarea software este realizată la diferite stadii ale ciclului de viață de dezvoltare a software. În modelul cascadă, există întotdeauna o *fază de testare* separată, aproape de finalizarea unei *faze de implementare*. Cu toate acestea, în metoda agilă și mai ales în programarea extremă, testarea se face de obicei în același timp cu codarea, sau, cel puțin activitatea de testare începe în primele etape ale iteraţiei.

Pentru că faza de testare se face la fiecare mică iterație - care dezvoltă o mică bucată de software - utilizatorii pot folosi frecvent aceste piese noi de software și a valida valoarea.

După ce utilizatorii cunosc valoarea reală a piesei actualizate de software, ei pot lua decizii mai bune cu privire la viitorul software. Cu o retrospectivă a valorii și sesiunii de re-planificare software la fiecare iterație - modelul Scrum are maximum o lună ca durată de iteraţie, - va ajuta astfel echipa să adapteze continuu planurile sale astfel încât să maximizeze valoarea pe care o oferă.

Această practică iterativă introduce, de asemenea, o "mentalitatea de produs", mai mult decât "mentalitatea de proiect" de tip cascadă. Software poate fi văzut ca un organism viu, care se

schimbă în mod activ ca urmare a schimbărilor de mediu. Atâta timp cât software este utilizat, în special atunci când are concurenţi, iterațiile în dezvoltarea de software agilă va conduce schimbarea.

Din cauza stilului scurt de iteraţie în dezvoltarea de software agilă, există de asemenea, legături puternice cu conceptul de "lean startup".

Cod vs. documentație

Într-o scrisoare către *IEEE Computer*, Steven Rakitin și-a exprimat cinismul cu privire la dezvoltarea agilă, numind un articol care sprijină dezvoltarea software agilă "încă o încercare de a submina disciplina de inginerie software" și a tradus: "software de lucru înaintea documentației cuprinzătoare" ca "Vrem să petrem tot timpul codificând. Ţineţi minte, programatorii reali nu scrie documentație."

Acest lucru este contestat de către susținătorii dezvoltării de software agilă, care afirmă că dezvoltatorii ar trebui să scrie documentație în cazul în care acesta este cel mai bun mod de a îndeplini obiectivele în cauză, dar că există modalități de multe ori mai bune de a atinge aceste obiective decât scrierea documentației statice. Scott Ambler afirmă că documentația ar trebui să fie "doar destul de bună", că documentație prea multă sau completă ar provoca, de obicei, pierderi, iar dezvoltatorii rareori au încredere în documentația detaliată, pentru că este, de obicei, desincronizată cu codurile, în timp ce documentația prea puțină poate provoca, de asemenea, probleme de întreținere , comunicare, învățare și schimb de cunoștințe. Alistair Cockburn a scris despre metoda *Crystal Clear*:

> Cristal consideră dezvoltarea a fi o serie de jocuri cooperatiste, iar furnizarea de documentare este destinată să fie suficientă pentru a ajuta la următorul câștig din jocul următor. Produsele de lucru pentru Crystal includ cazuri de utilizare, lista cu riscurile, planul de iteraţie, modele de domeniu de bază, și note de proiectare pentru informare cu privire la alegeri ... cu toate acestea, nu există template-uri pentru aceste documente, și descrierile sunt în mod necesar vagi, dar obiectivul este clar,

doar suficientă documentație pentru urmatorul joc. Întotdeauna tind să descriu aceasta echipei mele ca: ceea ce ai vrea să știi dacă ai intra în echipă mâine.
- Alistair Cockburn

Metode de dezvoltare

Metodele agile sunt axate pe diferite aspecte ale ciclului de viață de dezvoltare software. Unele se concentrează pe practici (de exemplu, XP (Programarea extremă), Programarea pragmatică, Modelarea agilă), în timp ce altele se concentrează pe gestionarea proiectelor software (de exemplu, Scrum). Cu toate acestea, există abordări care oferă o acoperire completă pe durata ciclului de viață de dezvoltare (de exemplu DSDM (Metoda dinamică a dezvoltării sistemelor), IBM RUP (Procesul unificat raţional)), în timp ce cele mai multe dintre ele sunt adecvate din faza de specificare a cerințele (FDD (Dezvoltarea în funcţie de caracteristici), de exemplu). Astfel, există o diferență clară între diferitele metode agile în acest sens.

Dezvoltarea agilă este susținută de un pachet de practici concrete propuse de metodele agile, care vizează domenii cum ar fi cerințele, proiectarea, modelarea, codarea, testarea, managementul de proiect, procesul, calitatea, etc

Alianța Agile a oferit o colecție online cuprinzătoare, cu un ghid al practicilor agile aplicabile.

Croiala metodei

În literatura de specialitate, termeni diferiți se referă la noțiunea de adaptare metodei, inclusiv "croiala metodei", "adaptarea fragmentelor metodei" și "ingineria metodei situaționale". Croiala metodei este definită ca:

Un proces sau capacitate în care agenții umani determină o abordare a dezvoltării unui sistem pentru o situație specifică a unui proiect prin schimbari sensibile în, precum și interferențe dinamice între contexte, intenții, și fragmentele metodei.

Teoretic, aproape toate metodele agile sunt potrivite pentru croiala metodei. Chiar metoda DSDM este folosită în acest scop și a fost adaptată cu succes într-un context CMM (Metoda maturităţiii capabilităţii). Adaptarea la situaţie poate fi considerată ca o caracteristică distinctivă între metodele agile și metodele tradiționale de dezvoltare software, acestea din urmă fiind relativ mult mai rigide și prescriptive. Implicația practică este că metodele agile permite echipelor de proiect să adapteze practicile de lucru în funcție de necesitățile proiectelor individuale. Practicile sunt activități și produse care sunt parte a unui cadru de metode concrete. La un nivel mai extrem, filosofia din spatele metodei, constând dintr-un număr de principii, ar putea fi adaptată (Aydin, 2004).

Programarea extremă (XP) face explicită nevoia de adaptare a metodei. Una dintre ideile fundamentale ale XP este că niciun proces nu se potrivește fiecărui proiect, ci mai degrabă că practicile ar trebui adaptate la nevoile proiectelor individuale. Adoptarea parțială a practicilor XP, după cum a sugerat Beck, a fost raportată în mai multe rânduri. Mehdi Mirakhorli propune o practică a croielii care oferă o suficientă planificare și orientări pentru adaptarea tuturor practicilor. Practica de cercetare-dezvoltare este concepută pentru personalizarea XP. Această practică, propusă la început ca o lucrare de cercetare de amploare în workshop-ul ASPO la conferinţa ICSE 2008, este în prezent singura metodă propusă și aplicabilă pentru personalizarea XP. Deși este în mod special o soluție XP, această practică are capacitatea de extindere la alte metodologii. La prima vedere, această practică pare să fie în categoria adaptării metodei statică, dar experiențele cu practice de cercetare-dezvoltare spune că poate fi tratată ca o adaptare a metodei dinamică. Distincția dintre adaptare metodei statică și adaptarea metodei dinamică este subtilă.

Comparație cu alte metode

RAD

Metodele Agile au multe în comun cu tehnicile RAD (Rapid Application Development - dezvoltare rapidă de aplicații) din anii 1980/'90 aşa cum au fost îmbrățișate de James Martin și alții. În plus față de metodele focalizate pe tehnologie, metodele concentare pe client-şi-proiectare, cum ar fi Visualization- Driven Rapid Prototyping dezvoltat de Brian Willison, funcţionează prin angajarea clienților și utilizatorilor finali, pentru a facilita dezvoltarea de software agil.

CMMI

În 2008, Institutul de Inginerie Software (SEI) a publicat raportul tehnic "*CMMI sau Agile: De ce să nu se folosească ambele*" pentru a face clar că CMMI (Capability Maturity Model Integration - Integrarea modelului de maturitate a capabilităţii) şi Agile pot coexista. Procesele de dezvoltare compatibile CMMI moderne sunt, de asemenea, iterative. Versiunea 1.3 CMMI include sfaturi pentru punerea în aplicare şi îmbunătăţirea proceselor Agile și CMMI împreună.