Metodologie dezvoltare software AFIR

Versiunea 1.1

FRAMEWORK AFIR

AFIR adopta o metodologie agila de lucru in ceea ce priveste dezvoltarea de software. Prin aceasta se doreste alinierea la urmatoarele principii Agile:

1. Livrarea continua si cat mai devreme a software-ului (cu o frecventa de cateva saptamani, nu luni)
2. Imbratisarea nevoilor de schimbare a cerintelor, chiar si in etapele de final ale proiectului
3. Cooperare intensa si zilnica intre business si echipa de dezvoltare
4. Conversatia este cea mai buna forma de comunicare
5. Software-ul functional este principala masura a progresului
6. Dezvoltare durabila, capabila sa mentina un ritm constant
7. Atentie continua pentru excelenta tehnica si un design bun, simplitatea fiind esentiala
8. Cele mai bune arhitecturi, cerinte si design-uri software provin din echipe care se auto-organizeaza
9. În mod regulat, echipa reflectă cum să devină mai eficientă și ajustează în consecință
10. Proiectele sunt construite in jurul unor indivizi motivati, carora trebuie sa li se acorde incredere

Pornind de la aceaste principii Agile, AFIR si-a elaborat propriul framework de lucru pe care intentioneaza sa il aplice atat in proiectele desfasurate intern, cat si in cele realizate de sau in colaborare cu contractori externi. Frameworkul are la baza Scrum, cel mai popular framework Agile, pentru a gestiona procesul, la care sunt adaugate in acelasi timp practici si unelte moderne de dezvoltare software.

La o privire de ansamblu, framework-ul AFIR arata astfel:

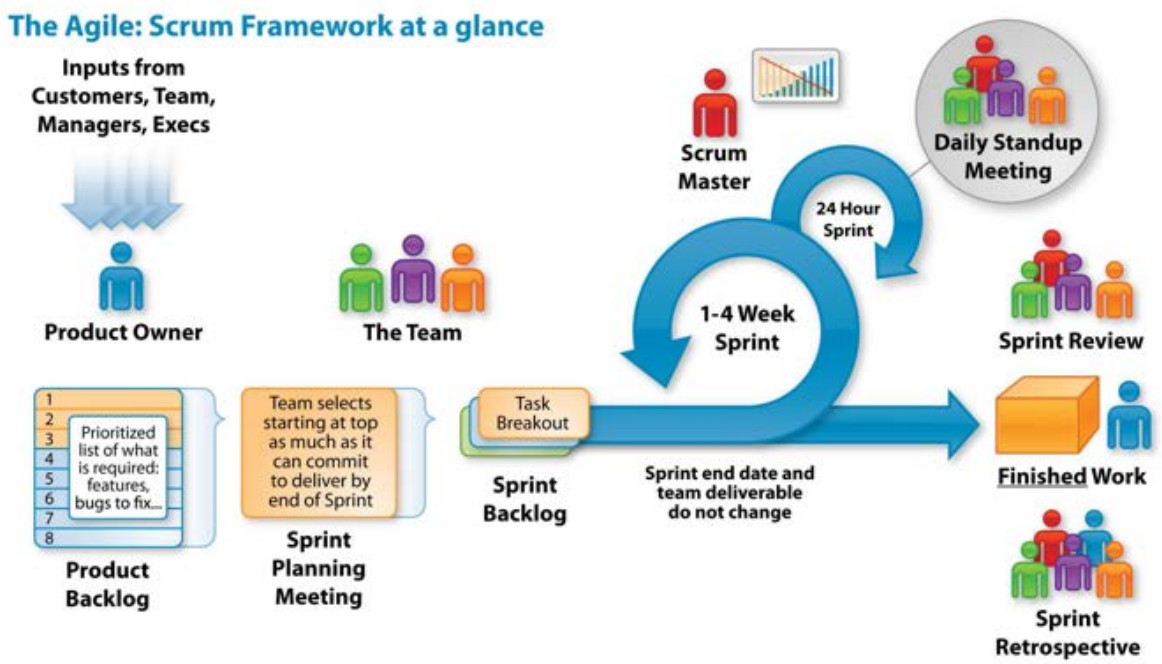
- Scrum, ca si proces de lucru iterativ si incremental, preferandu-se iteratii de dezvoltare de 2 saptamani

- Procesul e sustinut prin AzureDevOps, reprezentand suita de unelte prin care sunt gestionate echipe, Product backlog-uri, iteratiile si Sprint Backlog-urile aferente, precum si partea de raportare/metrici: burndown chart, velocity, cummulative flows
- Mangement-ul eficient al proiectului si echipelor se face folosind Delivery Plans (tot parte din AzureDevOps)
- Sunt incorporate tehnici de branching (in source control) care sa favorizeze livrarea unor incremente de calitate, cu potential mare de a fi puse in productie (production-ready). Pentru aceasta va fi folosit GiT ca si sistem de source control, integrat in AzureDevOps
- Calea spre productie a fiecarui proiect este bine punctata, prin definirea mediilor de dezvoltare, testare acceptanta si productie
- Procesul de deployment este automat, folosind tehnici de Continuous Integration si Continuous Deployment, automatizare asigurata de asemenea prin AzureDevOps
- Inglobarea de strategii de testare care ridica gradul de incredere in ceea ce priveste calitatea fiecarui increment, printr-o dimensionare buna a tipurilor de teste, pornind de la cele unitare, de integrare si pana la cele de interfata. Accentul se va pune cat mai mult pe testare automata.
- Calitatea sistemelor dezvoltate este monitorizata continuu, pe intreg parcursul procesului de dezvoltare. Acest lucru se realizeaza in mod automat, folosind SonarQube, integrat cu AzureDevOps.

Acest document isi propune sa descrie framework-ul AFIR, acordand atentie procesului de dezvoltare neesar in cadrul unei echipe, raportarea la scara intregii organizatii si derularea mai multor proiecte in paralel, enumerarea practicilor de dezvoltare ce trebuie sa fie incorporate.

PROCESUL DE DEZVOLTARE

Modelul de livrare AFIR este construit pe baza metodologiei Agile, respectiv a framework-ului Scrum, un cadru iterativ, incremental și adaptabil pentru construirea unui software complex, care propune livrarea de software functional în iterații fixe (de 1-4 săptămâni), în funcție de ceea ce are nevoie cel mai mult componenta de business și de promovare a unui proces de îmbunătățire continuă. Durata unei iteratii va fi stabilita de comun acord intre Prestatorul extern si DIT-AFIR in cadrul Raportului Initial de Activitate. Este un proces iterativ, deoarece dezvoltarea se realizează în intervale fixe de timp numite Sprinturi și incremental deoarece fiecare Sprint are ca rezultat un



increment care se bazează pe cel precedent până la livrarea finală.

În afară de echipa în sine, sunt identificate două roluri specializate: Product Owner (PO) și Scrum Master (SM). Product Owner-ul este una sau mai multe persoane desemnate de business-ul AFIR, care deține controlul asupra cerintelor pe care echipa trebuie sa le implementeze, descrise sub forma de User Story-uri. User Story-urile sunt colectate intr-un artifact numit Product Backlog, unde sunt prioritizate de catre Product Owner. Scrum Master-ul este unul dintre membrii

echipei, care asigură că procesul Scrum este urmărit, identifică și elimină impedimentele, ceea ce duce la îmbunătățirea continuă. Scrum Master-ul va fi desemnat de comun acord de Prestator cu DIT-AFIR.

User Story-urile sunt discutate între PO și echipă în sesiuni Refinement. Odată ce Story-urile sunt suficient de clare, ele sunt estimate de către Echipă. În același timp, PO este responsabil pentru atribuirea valorii de business pentru fiecare dintre Story-uri. Împreună, valoarea Story-urilor și estimările determină de obicei ordinea în care are loc dezvoltarea. Ca atare, Product Backlog-ul devine o listă ordonata de User Story-uri care vor fi preluate de Echipă în fiecare Sprint de dezvoltare.

Înainte de a începe o nou Sprint, echipa se angajează sa implementeze o serie de Story-uri din partea de sus a Product Backlog-ului pe care consideră că pot să le facă în viitoarea iterație. Aceast moment are loc în cadrul ceremoniei de planificare a Sprint-ului (Sprint Planning) și, la sfârșitul unui Sprint, echipa prezintă PO-ului (și opțional altor părți interesate), în ceremonia de revizuire Sprint (Sprint Review) funcționalitatea care a fost realizată. Ciclurile de feedback scurte sunt realizate prin organizarea unei întâlniri zilnice în care echipa se sincronizează (Daily Meeting): aici echipa analizează progresul, decide cu privire la sarcinile de lucru următoare și vede dacă există impedimente.

Fiecare iteratie este încheiată de o retrospectivă (Sprint Retrospective), unde echipa evaluează ceea ce s-a întâmplat bine, ceea ce nu a mers conform planului și ceea ce poate fi îmbunătățit. Rezultatul acestei ceremonii ar trebui să fie o listă de acțiuni/activitati în care cel puțin cea mai importantă dintre ele este abordată în următorul Sprint.

Prin natura sa, Scrum (si implicit Framework-ul AFIR) are un model foarte dinamic. Permite Product Owner-ului să construiască și să actualizeze constant un Product Backlog în sincronizare cu cerințele de business și schimbările din piață. Productivitatea ridicată se bazează, de asemenea, pe faptul că echipa discută funcționalitatea pe parcursul mai multor sesiuni de Refinement, le desparte în sarcini de lucru (task-uri) în cadrul întâlnirii de planificare și atinge un momentum în cursul Sprint-ului atunci când le implementează. Prin urmare, aceasta trebuie să fie o perioadă stabilă, când elementele importante nu se pot schimba, altfel orice schimbare va avea un impact negativ. Acest lucru se realizează prin înțelegerea comună a ceea ce inseamna ca un Story este gata să intre într-un Sprint (Definition of Ready) și ce criterii trebuiesc îndeplinite ca un Story sa fie considerat complet (Definition of Done).

În continuare sunt descrise mai in detaliu ceremoniile, rolurile si artifactele care sunt utilizate in cadrul procesului de dezoltare.

CEREMONII

Sprint Planning

Obiectiv: Echipa selecteaza ce trebuie sa fie implementat in noua iteratie/sprint.

Pregatirea ceremoniei:

- Responsabil: Scrum Master
- Participanti: Echipa, Product Owner, Scrum Master
- Story-urile trebuie sa fie definitie si estimate in story points
- Ceremonia are loc la inceputul fiecarei iteratii

Moderarea ceremoniei:

Prima parte din Sprint Planning

- Ideal sa dureze maxim 2 ore
- Product Owner-ul prezinta echipei cerintele surprinse in prima parte a Product Backlog-ului, incepand cu cele mai prioritare
- Echipa selecteaza o lista de elemente din Product Backlog, pe care cred ei ca le pot implementa in urmatorul sprint
- Echipa isi da angajamentul catre Product Owner ca vor face tot ce le sta in putinta pentru a finaliza Story-urile selectate

A doua parte din Sprint Planning

- Ideal sa dureze maxim 2 ore
- Echipa creaza un plan pentru urmatorul sprint prin mutarea Story-urilor in Sprint Backlog

- Story-urile din Sprint Backlog sunt despartite in task-uri de dezvoltare
- Desi nu mai este necesar ca Product Owner-ul sa participe la aceasta parte, el/ea trebuie sa fie disponibil (“on-call”) pentru a raspunde la orice intrebari de clarificare pe care le-ar avea echipa in procesul de creare al task-urilor
- Dupa terminarea ceremoniei, Scrum Masterul notifica pe toate persoanele implicate (“stakeholderi”) despre angajamentul echipei cu privire la ce se va implementa in urmatoarea perioada

Rezultate in urma ceremoniei:

- Sprint Backlog-ul a fost creat
- Story-urile au task-uri asignate
- Stakeholderi informati
- Echipa si-a luat angajamentul pentru sprintul abia inceput

Daily Meeting

Obiectiv: Sincronizarea echipei.

Pregatirea ceremoniei:

- Responsabil: Scrum Master
- Participanti: Echipa
- Se desfasoara in acelasi loc, la aceeasi ora

Moderarea ceremoniei:

- Ceremonia se desfasoara pe parcursul a 15 minute
- Fiecare membru al echipei porneste de regula prin a raspunde la 3 intrebari:

- Ce task-uri a terminat de la ultima intalnire de sincronizare. Task-urile finalizate sunt trecute din “In Progress” in “Done”
- Ce task-uri planuieste sa finalizeze pana la urmatoarea Daily Meeting
- Daca task-ul nu este in Sprint Backlog, il creeaza. Daca nu este realizabil intr-o singura zi, de preferabil sa il sparga in task-uri mai mici.
- Task-urile preluate sunt trecute in “In Progress”
- Cu ce impedimente se confrunta.
- Fiecare membru al echipei ar trebui sa asculte si sa observe daca exista impedimente care nu sunt explicit mentionate si sa le faca explicite
- Daca se porneste o discutie mai in detaliu intre doua sau mai multe persoane din echipa, care ar ocupa mai mult timp si nu este nici de interes general, ar trebui amanata pentru dupa incheierea ceremoniei

Rezultate:

- Sprint Backlog-ul este la zi
- Burn-down chart-ul este la zi

Sprint Review

Obiectiv: Echipa demonstreaza ce au realizat in Sprint-ul care tocmai se incheie

Pregatirea ceremoniei:

- Responsabil: Scrum Master
- Participantii sunt invitati: Echipa, Product Owner, si alte persoane implicate sau interesate de rezultatul Sprint-ului
- Pregatirea unui demo care trece prin toate Story-urile terminate in cadrul Sprint-ului, inclusiv a mediului si a datelor peste care se va face demo-ul

- Pregatirea salii unde va avea loc ceremonia si a echipamentului necesar

Moderarea ceremoniei:

- Intalnirea nu ar trebui sa dureze mai mult de 2 ore
- Story-urile care nu sunt finalizate nu ar trebui de regula discutate

Rezultate:

- Product Owner-ul poate redeschide Story-uri
- Scrum Master-ul ar trebui sa ceara intotdeauna feedback cu privire la Sprint-ul curent
- Product Owner-ul sau alti participanti din business pot sa solicite adaugarea de noi cerinte in Product Backlog

Sprint Retrospective

Obiectiv: Învătăm din experiența anterioară pentru a îmbunătăți productivitatea echipei.

Prima directivă

Indiferent de ceea ce descoperim, trebuie să înțelegem și să credem cu adevărat că fiecare membru al echipei si-a făcut datoria cat mai bine posibil, având în vedere ceea ce era cunoscut atunci, abilitățile, resursele disponibile și situația existentă.

Pregatirea ceremoniei:

- Responsabil: Scrum Master
- Participanti: Echipa, (optional) Product Owner

Moderarea ceremoniei:

- In introducere, se prezinta pe scurt obiectivul ceremoniei si prima directiva
- Fiecare participant raspunde la 3 intrebari, raportandu-se la sprintul care tocmai s-a incheiat:

- Ce a mers bine?
- Ce nu a mers bine?
- Ce poate fi imbunatatit?

- Pentru a conduce discutia, pe langa evenimentele intampinate in timpul sprintului, echipa se poate uita si la burndown chart-ul din timpul sprintului, velocitatea obtinuta, diferenta (daca e cazul) dintre story-urile pentru care si-a luat angajamentul in sprint planning si ce a fost realizat in final.
- De regula, lucrurile care nu au mers bine sau care merita imbunatatite au la baza anumite impedimente. Unele impedimente tin de echipa, altele nu (pot tine de exemplu de organizatie), si este responsabilitatea Scrum Master-ului de a le comunica in exterior si de a urmari sa fie rezolvate. Indiferent de sursa, impedimentele se tin intr-un Backlog de impedimente, cel putin pana sunt rezolvate sistematic.
- In momentul in care se gasesc impedimente, echipa trebuie sa se gandeasca la posibile solutii si actiuni pe care trebuie sa le faca in viitorul apropiat, actiuni care se prioritizeaza si sunt asignati responsabili dintre membri echipei.

Rezultatele ceremoniei:

- Backlog-ul de impedimente este actualizat
- Echipa preia cel mai prioritar impediment sau lucru de imbunatatit in urmatorul sprint
- Artefactele “Definition of Ready” si “Definition of Done” sunt actualizate

Refinement

Obiectiv: Se clarifica cerintele (User Story-urile) si se (re)prioritizeaza Product Backlog-ul

Pregatirea ceremoniei:

- Responsabil: Scrum Master
- Participantii sunt invitati: Echipa, Product Owner
- Echipa poate petrece până la 10% din timpul său pentru a rafina cerintele, de regula 2-4 ore pe saptamana
- Frecvența ideala: o dată pe săptămână
- Confirmați cu Product Owner-ul că ordinea elementelor din partea de sus din Product Backlog este corectă
- Amintiți echipei să examineze Product Backlog-ul cu 24 de ore înainte de sesiunea de rafinare
- Story-urile sunt făcute înainte de întâlnirea rafinare, unde ele trebuie discutate
- Story-urile sunt publicate în Product Backlog înainte de sesiunea de rafinare

Moderarea ceremoniei:

- Respectarea timpului alocat - daca echipa a convenit sesiuni de 2 ore saptamanal, acestea trebuiesc respectate
- Echipa pune întrebări, iar Product Owner-ul răspunde la ele până când poveștile sunt bine înțelese de către Echipă
- În cadrul acestei întâlniri, participanții pot:
    - adăuga Story-uri noi
    - descompune Story-uri care sunt prea mari (Epics)
    - imbunătățiți Story-urile care nu sunt bine scrise
    - estima Story-uri
    - adăuga criterii de acceptanta

- arunca o privire in adancime in Product Backlog pentru o planificare tehnică pe o rază mai lungă de timp

Rezultatele ceremoniei:

- O parte din story-uri sunt acceptate de catre echipa ca fiind gata pentru a fi preluate in dezvoltare
- Partea de sus a Product Backlog-ului este prioritizata corect
- User Story-urile sunt imbogatite cu informatii
- Crearea de noi Story-uri sau impartirea lor in altele mai granulare
- Story-urile discutate sunt estimate

ROLURI

Echipa

Echipa este un grup cross-functional de persoane, responsabil de livrarea unui increment de produs la sfarsitul fiecarui Sprint. Echipa are control total asupra volumului de munca pe care il preia in cadrul unei iteratii, atat timp cat respecta prioritatile stabilite de catre Product Owner. Numarul de User Story-uri pe care echipa in preia intr-un Sprint este decis in cadrul ceremoniei de planificare a Sprintului (Sprint Planning). Echipele sunt responsabile pentru inspectarea si adaptarea propriului proces de munca, ceea ce se intampla indeosebi in ceremonia de retrospectiva (Sprint Retrospective) de la sfarsitul fiecarui Sprint.

Metodologia nu diferentiaza intre rolurile din cadrul echipei, precum Analist de Business, Tester, Arhitect sau Dezvoltator, etc. Asta nu inseamna ca aceste roluri nu exista, doar ca Scrum-ul ii priveste pe toti ca membrii ai echipei de dezvoltare, care lucreaza impreuna pentru a implementa produsul definit de Product Owner. Altfel spus, fiecare membru al echipei participa activ la implementare:

- Arhitectul dezvolta modele, scheme de baze de date, arhitectura sistemului
- Designer-ul dezvolta interfata si experienta utilizatorului

- Analistul produce cerinte detaliate, documentatie si manuale utilizator
- Programatorii dezvolta codul propriu-zis
- Testerii dezvolta test cases, teste de acceptanta si testarea propriu-zisa

In functie de dimensiunea echipei si a proiectului, de aptitudinile fiecarui membru al echipei, de la caz la caz, unele din aceste roluri pot fi indeplinite si de aceleasi persoane.

Caracteristici:

- cross-functional
- foarte colaborativa
- se auto-organizeaza singura
- angajata sa isi ineplineasca obiectivele in fiecare sprint
- orientata catre prioritati si valoare de business

Ceremonii in care este necesar sa participe:

- Refinement
- Planificarea Sprintului (Sprint Planning)
- Intalnirea zilnica (Daily Meeting)
- Revizuirea Sprintului (Sprint Review)
- Retrospectiva Sprintului (Sprint Retrospective)

Product Owner

Product Owner-ul (PO) este de obicei un actor cheie al unui proiect. Este important ca un PO să aibă o viziune bună asupra a ceea ce vrea să construiască și să poată transmite acea viziune echipei. PO este proprietarul Product Backlog-ului și se axează in principal pe livrarea valorii către business. El este, de asemenea, responsabil pentru păstrarea Product Backlog-ului intr-o formă

prioritizată astfel încât elementele aflate în partea de sus a listei să aducă cea mai mare valoare catre business și sunt bine definite si detaliate astfel incat să fie luate în următorul Sprint.

PO este considerat parte a echipei și poate fi necesar ca alte părți interesate să ajute la prioritizarea User Story-urilor, asigurându-se că valoarea de business și estimările sunt luate în considerare atunci când se efectuează ordonarea lor.

Caracteristici:

- Mandat (sa poate decide in ce mod se implementeaza o functionalitate sau sa hotarasca asupra prioritatii)
- Business value awareness
- Cunoasterea domeniului
- Disponibilitate

Responsabilitati:

- Este vocea clientului sau a departamentului de business din care face parte
- Creează și menține Product Backlog-ul, asigură faptul că Story-urile sunt bine definite, conform principiului INVEST (Independent, Negotiable, Valuable, Estimable, Small, Testable)
- Prioritizează și secvențează Backlog-ul în funcție de valoarea de business sau de ROI
- Asistă la elaborarea de cerinte sub forma de User Story-uri care sunt suficient de granulare pentru a fi preluate de echipa si realizate intr-un singur Sprint
- Transimite viziunea si obiectivele la inceputul fiecarui Sprint
- Participa la intalnirile de sincronizare zilnica, cele de planificare de la inceputul Sprint-ului, le cele de rafinare a Product Backlog-ului, precum si la cele de revizuire de la finalul unui Sprint
- Verifică progresul la sfârșitul fiecărui Sprint și are autoritatea completă de a accepta sau respinge User Story-urile efectuate
- Poate schimba cursul proiectului la sfârșitul fiecărui Sprint

- Comunica starea proiectului in exteriorul echipei
- Poate termina un Sprint inainte de finalul lui dacă se constată că este necesară o schimbare drastică a direcției (a fi privit ca un caz de exceptie si a nu se abuza de aceasta optiune)

Ceremonii in care este necesar sa participe:

- Refinement
- Planificarea Sprintului (Sprint Planning)
- Intalnirea zilnica (Daily Meeting) - in modul ideal, nu neaparat la toate, dar prezenta sa fie predictibila
- Revizuirea Sprintului (Sprint Review)

Scrum Master

Un Scrum Master este un lider servant, ajutând echipa să fie responsabilă in primul rand față de ei înșiși pentru angajamentele pe care le fac. Ca atare, unele dintre responsabilitățile sale principale sunt de a ajuta echipa sa se auto-organizeze. În acest proces, Scrum Master-ul trebuie, de asemenea, să ajute echipa în identificarea și înlăturarea impedimentelor, făcând munca lor cât mai eficientă posibil.

Rolul de Scrum Master este preluat de cineva care face parte din echipă, indiferent de rol și disciplină. El / ea este un antrenor (coach) și protector al echipei. Scrum Master-ul este acolo pentru a ajuta echipa în utilizarea framework-ului de lucru, pentru a aplica cat mai bine practicile si in final pentru a respecta si imbunatati un mod de lucru agreat de toata lumea la inceputul proiectului.

Caracteristici:

- Abilități de leadership si coaching
- Disponibilitatea pentru echipă și PO
- Cunoasterea in detaliu a framework-ului de lucru
- Un susținător pasionat al metodologiilor Agile

Responsabilitati:

Ajutând Product Owner-ul

- Prioritizarea Product Backlog-ului, în conformitate cu cele mai recente informații, provenite atât de la echipă, cât și PO sau de la alte părți interesate
- Asigurandu-se că cerințele și dorințele PO-ului sunt surprinse în product Backlog
- Asigurarea unui Product Backlog care conține un număr ușor de gestionat de User Stories, cu lucruri mai granulare în partea de sus și eventual mai generale din partea de jos. (Este contraproductiv să analizați prea mult partea de jos a Product Backlog-ului. Sunt sanse mari ca cerințele dvs. sa se schimbe odata cu evolutia produsului, prin conversatii ulterioare cu părțile interesate / clienți).
- Validarea faptului că User Story-urile sunt independente, negociabile, valoroase, estimabile, mici și testabile, în măsura posibilităților
- Educarea Product Owner-ului in concepte precum datoria tehnică și modul de păstrare a acesteia intr-o marjă limitată
- Asigurandu-se că Product Backlog-ul este un radiator de informații, imediat vizibil tuturor părților interesate
- Ajutarea Product Owner-ului să organizeze Product Backlog-ul în grupuri de Story-uri cu diferite prioritati, în livrabile, în roadmap
- Verificarea faptului că Prouct Owner-ul validează sau ajustează planul de lansare după fiecare revizuire de Sprint

Ajutând echipa

- Înțelegerea planificării pe termen lung
- Înțelegerea și practicarea agilității
- Utilizarea artefactelor de echipă și a radiatoarelor de informații: DoR, DoD, Sprint Backlog, Burndown-chart

- Se asigură că necesitatea rambursării datoriei tehnice a fost făcută explicit, imbunatatind treptat calitatea codului
- Ajuta echipa să fie auto-organizată și responsabil colectiv pentru toate aspectele ce tin de mdelul de livrare (dezvoltare, testare, documentare etc.)
- Menținerea atenției asupra obiectivelor Sprint-ului
- Se asigură că Backlog-ul Sprint este mereu actualizat
- Crearea unor sarcini (task-uri) clare și concise în unealta de management de proces și asignarea acestora User Story-urilor aferente
- Se asigură că User Story-urile se fac în ordinea corectă
- Facilitarea ceremoniilor Scrum
- Ajută echipa să ajungă la un consens cu privire la ceea ce se poate obține într-o anumită perioadă de timp
- Protejarea echipei de orice distragere a atenției (adică asigurarea că proiectul are capacitatea planificată până la sfârșitul sprintului)
- Eliminarea oricăror impedimente care împiedică exercitarea de către echipa a obiectivelor sale din Sprint
- Cauzează schimbări care sporesc productivitatea echipei

Ajutând Organizația

- Are loc o comunicare/cooperare corespunzătoare între echipe?
- Se intalnesc si colaboreaza Scrum Master-ii din diverse proiecte? Lucrează la lista de impedimente organizaționale?
- Creăm și dezvoltăm o cultura de învățare in cadrul organizatiei?

Ceremonii necesare

- Refinement
- Planificarea Sprintului (Sprint Planning)
- Intalnirea zilnica (Daily Meeting)
- Revizuirea Sprintului (Sprint Review)
- Retrospectiva Sprintului (Sprint Retrospective)

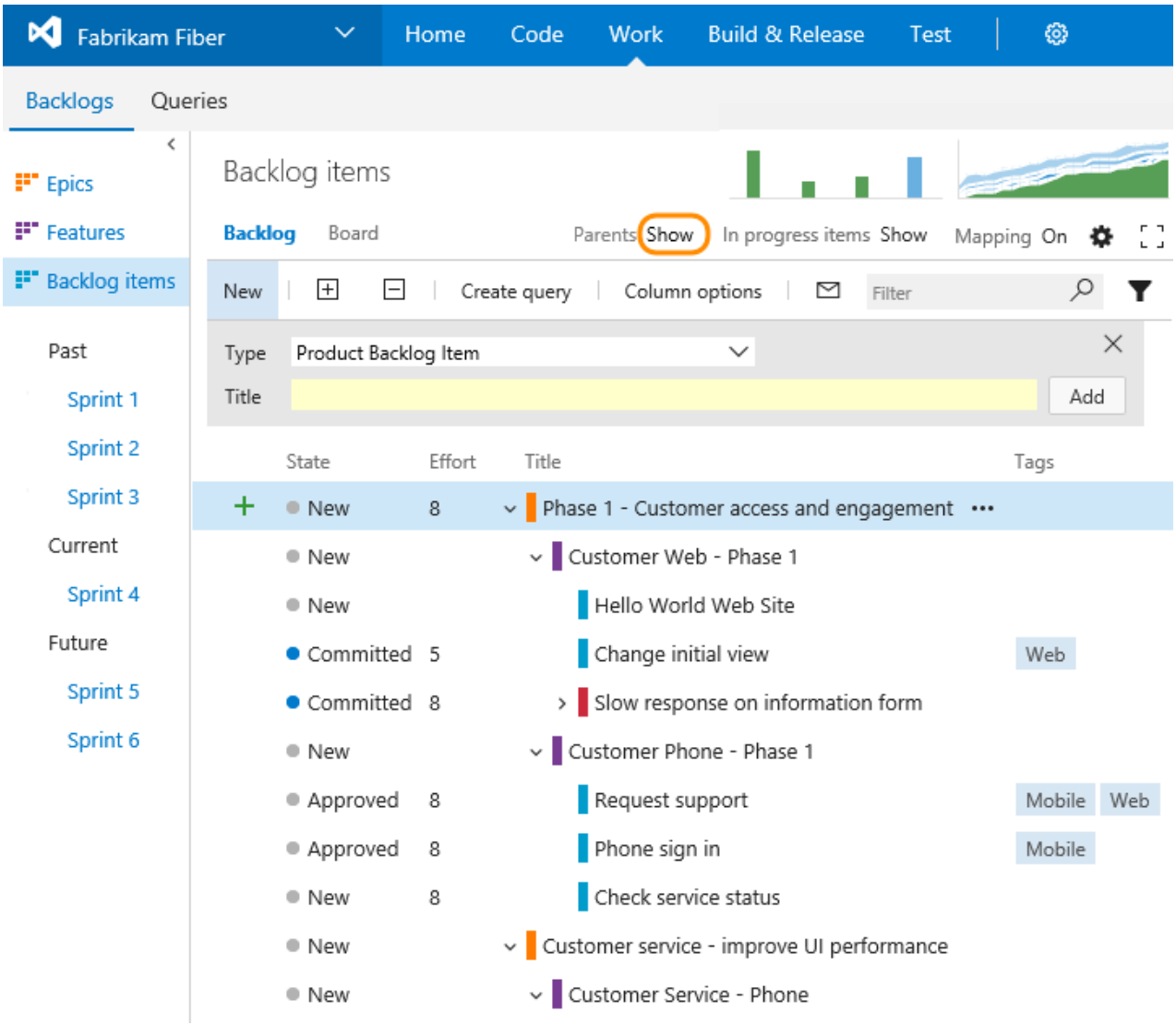
ARTEFACTE

Product Backlog

Product Backlog-ul este o lista ordonata de elemente pe care echipa trebuie sa le implementeze. Acestea pot fi orice care in final ajunge intr-un increment de produs: cerinte functionale, cerinte ne-functionale (securitate, performanta etc.), instalari de medii de test, investigari sau cercetare pe anumite subiecte. Toate aceste elemente pot fi proiritizate si bugetate de catre Product Owner.

Elementele care pot aparea in Product Backlog sunt de 3 categorii principale, si sunt folosite pentru a planifica si urmari evolutia proiectului:

- User story: descriere scurta si simpla a unui feature, de obicei din perspectiva persoanei care doreste noua functionalitate
- Epic: un story mare, complex, cu multe necunoscute, ceva pe care echipa nu il poate estima sau lua intr-un sprint, fara a-l rafina si sparge in prealabil in mai multe story-uri.



- Bug: un defect al unei functionalitati a sistemului

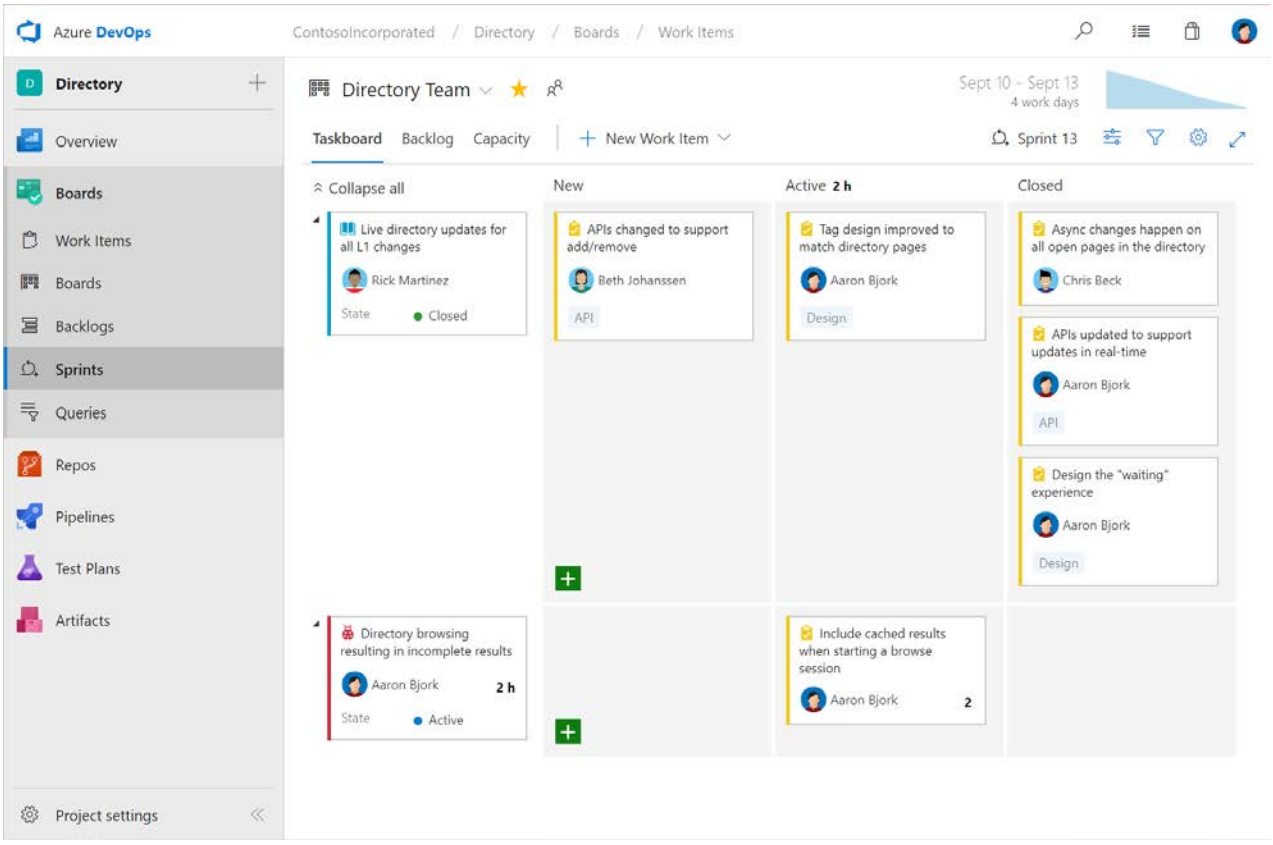
Un element din Product Backlog (epic, story sau bug) poate trece prin urmatoarele stari:

- New: este starea initiala, pe care o are in momentul cand este introdus in Backlog
- Approved: trecerea unui story din New in Approved este facuta de catre Product Owner, daca functionalitatea respectiva este aprobata pentru a fi luata in discutie in Refinement-uri si apoi preluata de catre echipa in Sprint-uri

- Ready: trecerea in Ready se face de catre echipa, in momentul cand aceasta considera ca are suficiente informatii pentru ca elementul respectiv din Product Backlog sa treaca in dezvoltare
- Committed: un element se afla in aceasta stare in moemntul in care se regaseste in sprintul curent de dezvoltare
- Done: starea trece in Done atunci cand elementul din Product Backlog (fie bug sau story) a fost implementat cu succes, si validat de catre Product Owner

Sprint Backlog

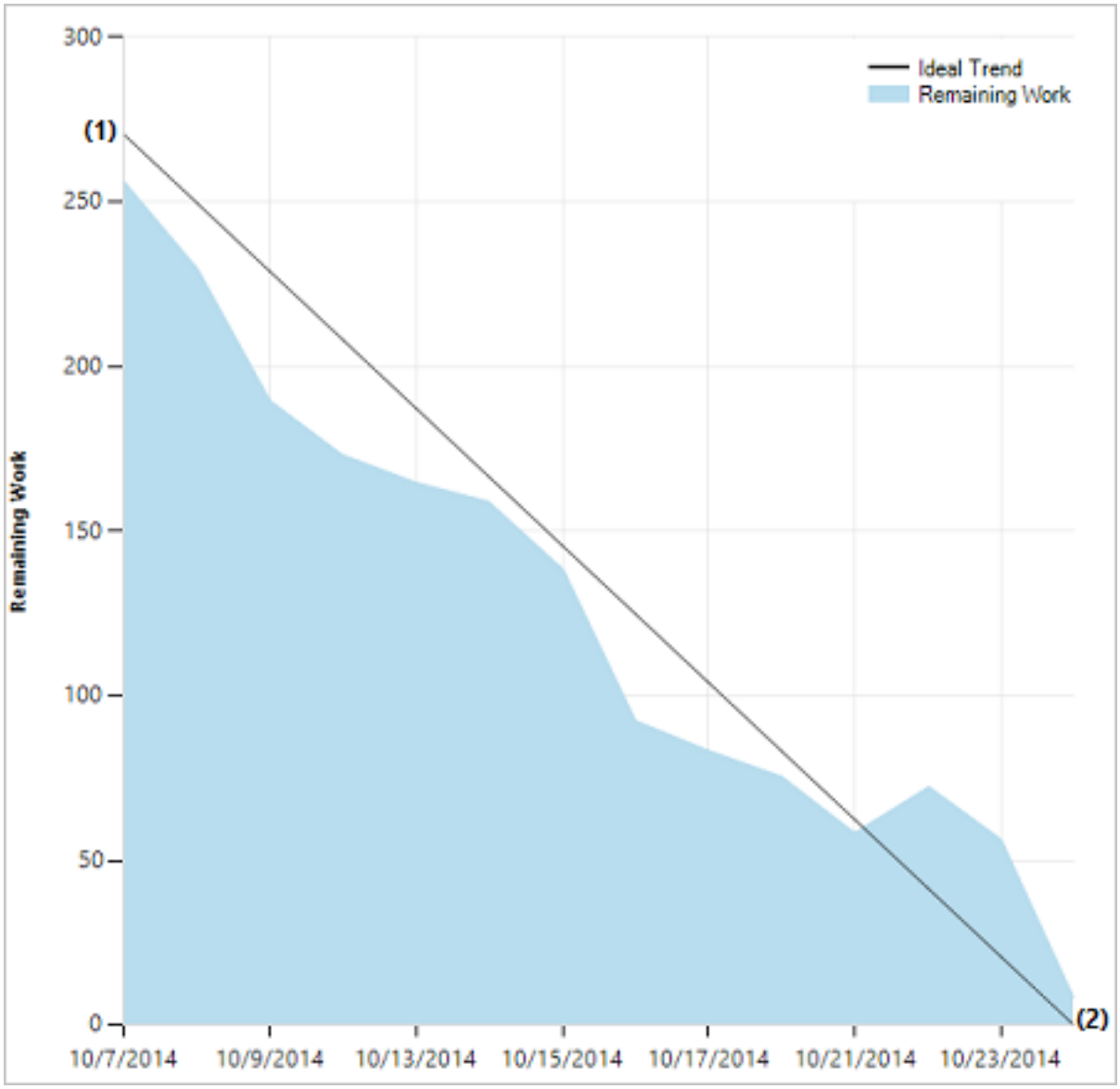
La inceputul fiecarui Sprint, story-urile care sunt preluate de catre echipa sunt transferate in Sprint Backlog si despartite in task-uri de inginerie. Cea mai buna reprezentare a Sprint Backlog-ului este printr-un board, la care echipa se raporteaza in permanenta, in special in timpul Daily



Meeting-urilor.

Burndown chart

De-a lungul Sprintului, echipa poate monitoriza acest grafic pentru a determina daca este sau nu pe cale sa isi realizeze planul facut la inceputul Sprintului.



Un grafic sănătos de sprint va arăta cam așa.

Linia ideala conectează cele două puncte:

- (1) Capacitatea totală a echipei la începutul sprintului

- (2) 0 efort necesar ramas la sfârșitul sprintului.

Panta reprezintă rata la care echipa are nevoie pentru a arde munca pentru a termina sprintul la timp.

Graficul real, zona albastră, reprezintă cantitatea totală de lucruri planificate in sprint și modul în care volumul de munca ramas se schimbă pe parcursul sprintului.

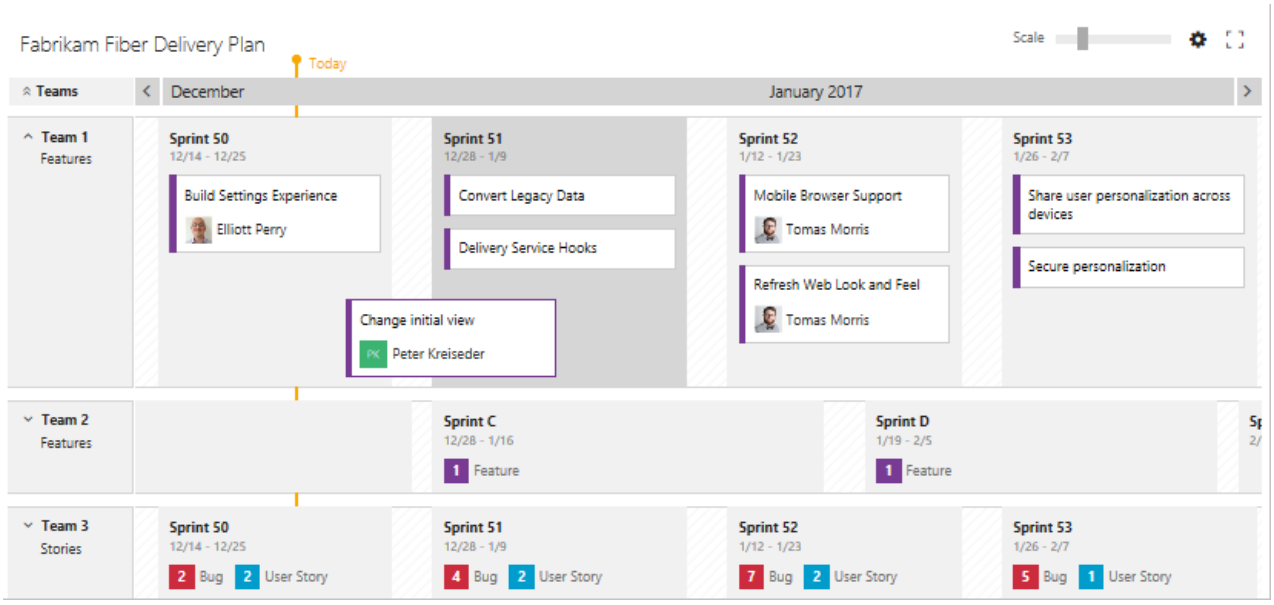
DELIVERY PLANS

Organizatia, si de multe ori mai ales management-ul, are nevoie de o privire de ansamblu asupra activitatilor de dezvoltare software sau mentenanta derulate in paralel de catre mai multe echipe. In orice moment din timp, putem avea o combinatie dintre situatiile de mai jos:

- mai multe echipe interne lucrand impreuna pentru realizarea unui proiect software complex
- echipe externe lucrand in paralel pentru realizarea unui proiect extrenalizat de AFIR
- echipe AFIR lucrand impreuna cu echipe ale contractorilor externi
- echipe AFIR deruland sprinturi de mentenanta si imbunatatiri mici/medii asupra proiectelor existente

Echipele autonome isi gestioneaza propriul Backlog si propriile sprinturi, iar toate aceastea contribuie pentru o directie unitara a proiectului sau a organizatiei.

Revizuirea periodica a calendarului echipelor asigura faptul ce ele functioneaza si deruleaza activitati in aceasi directie, spre un obiectiv comun. Vizualizarea acestui calendar este data de



modului Delivery Plans, din AzureDevOps.

Cateva din intrebarile pe care aceasta privire de ansamblu le-ar putea declansa:

- Sunt corect adresate dependintele dintre echipe, in special in ceea ce priveste livrabilele?
- Sunt corect gestionate resurse comune intre mai multe echipe?
- Sunt goluri in calendar cand nu sunt programate livrabile? Care este cauza?
- Care este gradul de incredere cu privire la atingerea unor obiective sau livrabile ale proiectului?
- In cazul schimbarilor de prioritate, sau aparitiei lucrurilor neprevazute, care este echipa cea mai potrivita pentru a le prelua? Si care sunt consecintele asupra planului existent?

TEHNICI DE ESTIMARE

Premisele pentru estimările Agile

Estimările bazate pe timp nu funcționează din mai multe motive:

• De obicei, oamenii estimează în "timpul ideal", adică încep de la ipotezele că știu exact ce trebuie făcut, au tot timpul să lucreze la sarcina la îndemână și nimic nu ii va întrerupe. În realitate, acest lucru nu se întâmplă niciodată

• Este dificil (aproape imposibil) să se ajungă la un consens în echipă, deoarece oamenii au domenii de expertiză diferite (dezvoltare, testare etc.), senioritati diferite și nu pot estima aceeași unitate de lucru cu aceeași măsură de timp

• Conul incertitudinii (Alistar Cockburn), care ne spune practic că există un grad ridicat de necunoscut într-un proiect software și estimările bazate pe timp devin din ce în ce mai exacte pe măsură ce ne apropiem de sfârșitul proiectului. În fazele incipiente, ele pot varia chiar cu până la +/- 400%

Metodologiile agile propun un mod diferit de abordare a estimărilor. Estimarea este folosită pentru stabilirea priorităților, nu pentru a stabili calendarul proiectului. Estimăm dimensiunea relativă a poveștilor, măsurăm o viteză medie pe sprint și abia in functie de asta obținem un plan de release. În plus, echipele sunt sfătuite să evalueze în mod continuu în timpul unui proiect, nu doar la inceput.

Nu este o estimare bazată pe timp, scopul fiind de a grupa povestiri în diferite dimensiuni - ar putea fi la fel de bine: super dificil, dificil, mediu, ușor, foarte ușor.

Cu toate acestea, cea mai obișnuită modalitate de estimare a dezvoltării software-ului Agile este folosirea Story Points și Planning Poker. Aceasta este strategia de estimare pe care o adoptă AFIR, aplicabilă în proiectele lor și este ceea ce vom acoperi în secțiunea de mai jos.

Estimarea in Story Points

Estimările sunt date în Story Points, folosind un set de numere ce provin din seria Fibonacci: 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100.

Deci, ce intra intr-un Story Point? O serie de factori trebuie luați în considerare atunci când se estimează complexitatea unui Story:

- Cantitatea de muncă
- Risc sau incertitudine
- Complexitatea
- Orice altceva ce intra in “Definition of Done” (de exemplu gradul de testare automata pe care il urmarim in proiect)

Fiecare Story este estimat prin atribuirea unei mărimi din numerele de mai sus, comparativ cu un Story de referință. Story-ul de referință este aleas la începutul proiectului și este foarte recomandat să nu se schimbe pe parcursul proiectului. Un candidat bun pentru un Story de referință este un Story suficient de mic pentru ca volumul de munca necesar să fie clar pentru toată lumea din echipă, dar în același timp ar trebui să conțină o anumită logică de business care să fie relevantă pentru compararea cu alte povești. De exemplu, Story-urile legate de infrastructură sau activități de administrare, cum ar fi implementarea unui ecran de login, nu sunt, de obicei, alegeri bune.

Odată ce ați ales o referință, trebuie doar să îi dați 3 sau 5 ca estimare. Nu trebuie să mergeți mai jos de 3 deoarece puteți găsi întotdeauna un Story mai mic decât dimensiunea de referință.

Poker Planning

Poker Planning este tehnica de estimare preferată atunci când estimam in Story Points. De ce? Pentru că se bazează pe consens și elimină presiunea celor care estimează.

La începutul acestui exercițiu de planificare agil, fiecare estimator primește un pachet de cărți Planning Poker. Fiecare carte are cate una dintre estimările valide, de exemplu: 0, 1/2, 1, 2, 3, 5, 8 , 13, 20, 40, 100 și infinit.

Pentru fiecare Story sau temă care urmează a fi estimată, un moderator citește descrierea. Vor fi niște discuții în care Product Owner-ul răspunde la orice întrebări pe care le au estimatorii, dar obiectivul Planning Poker în Scrum nu este de a obține o estimare care să reziste la toate examinările viitoare. În schimb, dorim o estimare valoroasă care să poată fi obținută cat mai repede.

După discuție, fiecare estimator selectează în mod privat o carte de Planning Poker care reprezintă estimarea sa agilă. Odată ce fiecare estimator a făcut o selecție, cărțile sunt simultan aratate, astfel încât toți participanții să își poată vedea estimările reciproc.

Estimările vor diferi probabil în mod semnificativ. Și e în regulă. Estimatorii cu cea mai mare și cea mai mică estimare vor explica perspectiva, astfel încât echipa să știe de unde provin. Moderatorul ia note în timpul acestei sesiuni de planificare agilă, ceea ce va fi util atunci când povestea este implementata și testată.

După discuție, fiecare estimator va reevalua estimarea prin selectarea unui nou card. Deseori, estimările se vor apropia una de cealalta in cel de-al doilea tur. Dacă nu, se repeta procesul până când echipa este de acord cu o singură estimare pe care să o folosească pentru User Story. Rar durează mai mult de trei runde în estimarea agilă pentru a se ajunge la un consens.

PRACTICI DE DEZVOLTARE

SOURCE CONTROL SI STRATEGII DE BRANCHING

Pentru a facilita colaborarea intre dezvoltatori pe acelasi cod sursa si pentru a urmari cat mai usor schimbarile, se va folosi Git ca si sistem de versionare al codului sursa. Acesta se integreaza cu AzureDevOps si controlat prin modul Azure Repos.

Folosind Git, fiecare dezvoltator are o copie a depozitului de surse, inclusiv toate informațiile despre ramura și istorie, pe mașina lor de dezvoltare. Fiecare dezvoltator lucrează direct cu propriul depozit local. Modificările sunt partajate între depozite ca un pas separat.

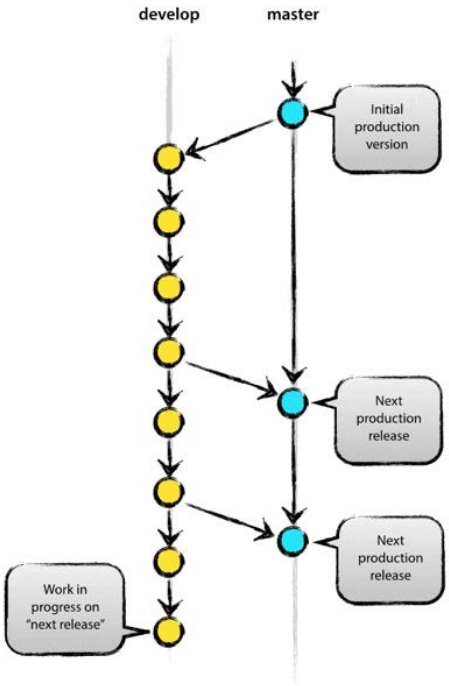
Dezvoltatorii pot comite fiecare set de modificări și pot efectua operații de control al versiunii, cum ar fi vizualizarea istoricului și compararea mai multor versiuni fără o conexiune la rețea. Ramurile sunt ușoare. Atunci când dezvoltatorii trebuie să schimbe contexte, ei creează o ramura locală privată. Dezvoltatorii pot schimba rapid dintr-o ramificație în alta pentru a pivota printre variații diferite ale bazei de cod. Ulterior, dezvoltatorii pot fuziona, publica sau elimina ramura.

Alegerea unei strategii de branching este un pas important, in cazul in care mai mulți membri ai echipei, sau chiar mai multe echipe, lucrează pe acelasi codebase și au nevoie de izolare pentru dezvoltare și livrare.

Modelul propus mai jos, este de fapt un fragment din modelul de branching al Git-ului (initiat de Vincent Driessen) si care poate fi o abordare comuna pentru multe echipe de dezvoltare.

Modelul propune doua ramuri de baza cu o durata de viata infinita:

- master - ramura principală în care codul sursă reflectă o stare pregătită de producție
- develop - ramura de dezvoltare în care codul sursă reflectă o stare cu cele mai recente modificări de dezvoltare livrate pentru următoarea versiune



Pe lângă ramurile de bază, modelul de dezvoltare poate utiliza o varietate de ramuri de sprijin pentru a ajuta la dezvoltarea paralelă între membrii echipei, pentru a facilita urmărirea caracteristicilor proiectului, a pregăti livrarea in producție a versiunilor urmatoare și pentru a ajuta la rezolvarea rapidă a problemelor din mediul de producție.

Diferitele tipuri de ramuri pe care le putem folosi sunt:

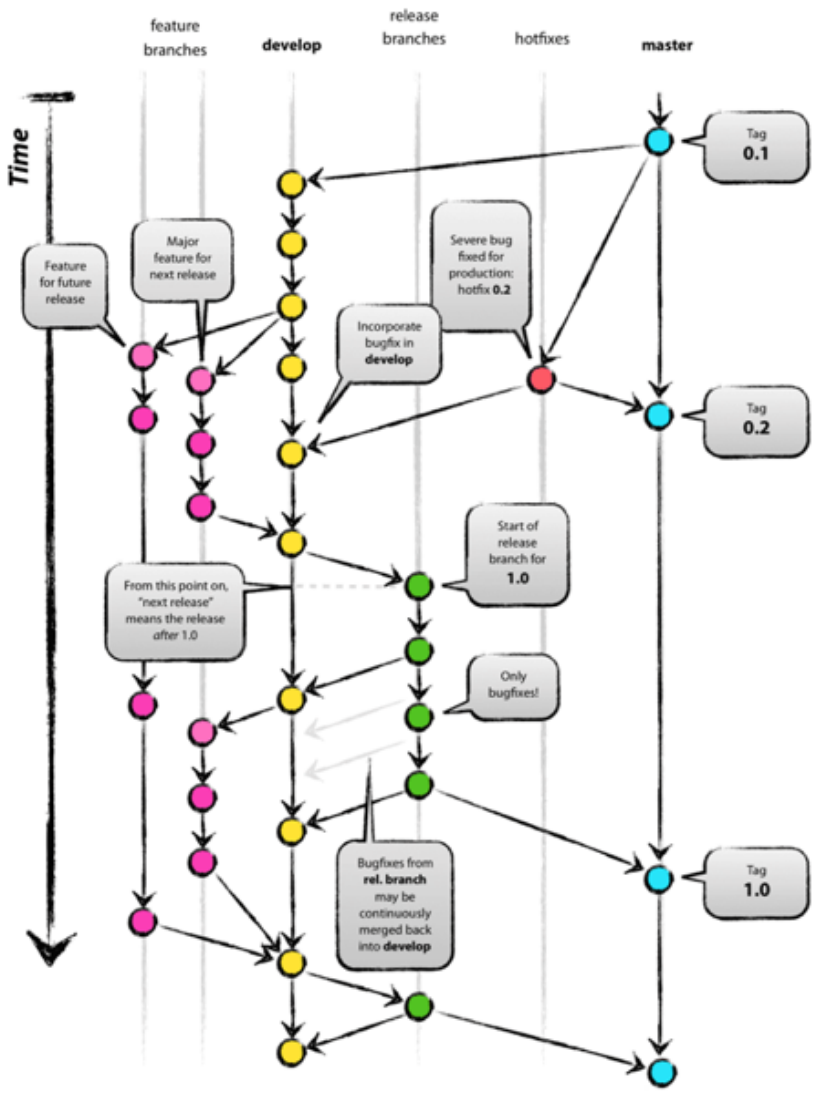
- feature branches
- release branches
- hotfix branches

Aceste ramuri sunt de scurtă durată, au un scop specific și sunt legate de reguli stricte în ceea ce privește ramurile care pot fi originea lor și care ramuri trebuie să fie obiectivele lor de fuziune (merge).

Feature branches sunt folosite atunci când aveți nevoie să lucrați cu o functionalitate noua a produsului. Ramificați din ramura de dezvoltare (develop) atunci când începeți să lucrați la o nouă caracteristică și să îmbinați modificările în ramura de dezvoltare (ca set unic de modificări) atunci când terminați implementarea caracteristicilor (de exemplu, cand DoD este îndeplinită).

Prin reunirea modificărilor într-un singur grup de modificări, istoricul ramurilor arată în mod clar ce caracteristici au fost implementate.

Release branches sunt create din ramura de dezvoltare atunci când dezvoltarea produsului este completă și nu se va mai scrie code pentru alte functionalitati. Ramura este folosită și pentru modificările / corecțiile de ultim minut și pentru efectuarea testelor de regresie. Când pregătirea pentru lansarea in producție, ramura de release este îmbinată în master ca set unic de schimbare și orice modificări făcute pe ramura de release trebuie să fie reunite in ramura de dezvoltare (develop), astfel încât versiunile viitoare să conțină, de asemenea, aceste modificari.

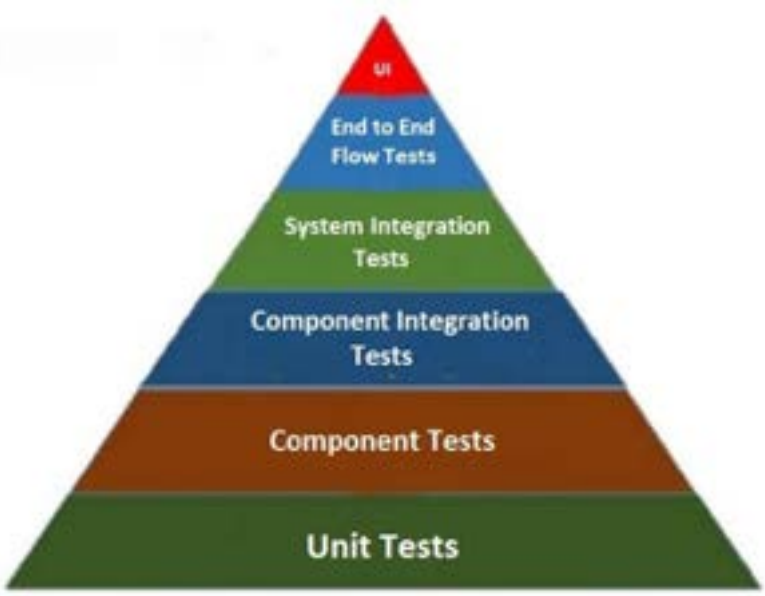


Hotfix branches sunt utilizate pentru a aborda bug-uri critice în producție și sunt ramificate de la eticheta corespunzătoare din ramura principală care marchează versiunea din producție. Acestea se comportă ca o ramură de release. Dupa ce sunt efectuate modificarile, bug-fix-ul trebuie să fie reintrodus în master, dar trebuie de asemenea să fie reintrodus în ramura de dezvoltare (astfel încât versiunile viitoare să includă și aceste remedii).

TESTARE AUTOMATA

Strategia de testare determină abordarea proiectului față de testare. Strategia analizează caracteristicile sistemului care urmează a fi construit, calendarul și bugetul proiectului și planifică efortul de testare.

Testarea automată este foarte recomandată în proiectele Agile. Dacă sunt implementate astfel încât să fie ușor de întreținut și extensibile, testele automate pot aduce beneficii mari în ceea ce privește acoperirea functionalitatilor cu teste de regresie și a pastra efortul de testare continua in parametrii realizabili. Este important să se țină seama de abordarea automatizării din timp, când testele manuale sunt proiectate și construite.



Un alt concept care apare mult in lumea dezvoltarii Agile, este piramida testelor, care tine tot de testare automata si de principiul testarii unui element cat mai devreme posibil.

Piramida de testare spune că testele de pe nivelele inferioare sunt mai ieftine pentru a fi scrise și întreținute, și mai rapide pentru a rula. Testele de pe nivelele superioare sunt mult mai costisitoare pentru a fi scrise și întreținute, și mai încete pentru a rula. Prin urmare, ar trebui să avem o mulțime de teste unitare, mai putine teste de servicii și foarte puține teste UI.

La baza piramidei de testare se află testarea unitara. Testarea unitara este fundamentul unei strategii solide de automatizare a testelor. Tindem să ne gândim la unitatea de testare ca fiind cel mai bine definită prin faptul că are un domeniu redus, timpul de execuție rapid și este apropiata de dezvoltare. Beneficiul de a trata cu un domeniu de aplicare mic înseamnă că, atunci când un test de unitate nu reușește, există un spațiu limitat de probleme de explorat, ceea ce ajută la accelerarea depanării. Acest lucru îi ajută pe dezvoltatori să-și imagineze ce sa întâmplat și le dă mai mult timp să se concentreze pe rezolvarea lucrurilor.

Stratul de servicii se află la mijlocul piramidei de testare (teste de componente, teste de integrare a componentelor și teste de integrare în sistem). Testarea acestui strat include utilizarea stratului de servicii sau a interfețelor stratului API sau a componentelor. Testele adreseaza mai multe elemente ale sistemului în același timp. Există mai multe dintre aceste teste decât în cazul testelor UI.

Stratul UI, cunoscut și sub denumirea de strat de tip end-to-end sau de testare a sistemului, necesită testarea întregului sistem prin interfața cu utilizatorul sau cu o altă interfață externă. Acest nivel de testare testează întreaga funcționalitate a sistemului dintr-o dată și reprezintă ceea ce ar experimenta utilizatorul sau ceea ce ar experimenta sistemul în ansamblu. Aceste teste sunt ușor de rupt în cazul în care codul se schimbă, mai ales când trebuie să simuleze acțiunile UI. Ele pot fi greu de scris și pot dura, de asemenea, un timp pentru a rula. Ele oferă beneficii în sensul că acestea sunt a doua linie de aparare a testarii. Dezvoltatorul poate observa erori funcționale care pot expune in acelasi timp teste unitare lipsă sau incorecte.

CALEA SPRE PRODUCTIE

Pentru a putea adopta o strategie de Continuous Delivery, avem nevoie in prealabil sa stabilim care este calea spre productie pe care o va avea proiectul. Scopul caii spre productie este de a defini niste etape in cliclul de dezvoltare care vor servi ca niste puncte de control. Pentru a trece de la o etapă la alta, un build trebuie să treacă prin aceste porți de calitate. Porțile vor fi separate prin diferite tipuri de teste. Odată ce testele sunt transmise, calitatea la acel nivel a fost asigurată, iar build-ul poate trece mai departe.

La un nivel general, mediile de dezvoltare si testare de care este nevoie sunt urmatoarele:

- Local:
    - Acest mediu este privat, limitat la un singur dezvoltator. Acest mediu este cel mai simplu în care să faceți schimbări și să testați propria implementare.
    - Testele recomandate înainte de a ieși dintr-un mediu local sunt teste unitare, testarea integrării cu componente mocked și testarea UI în măsura în care este posibil. Cu cât mai multe teste pe care le puteți face în acest mediu, cu atât mai multe sanse să aveți succes în mediul urmator.
- Mediul de integrare (CI)

- In mediul anterior, devezvoltatorul isi testeaza codul in izolare. Acesta este primul mediu in care codul lui este pus impreuna cu cel al altor dezvoltatori din echipa.
- Acesta este mediul cu cel mai scurt timp de viață. Se creează atunci când un build se declanșează și se distruge după ce se realizează build-ul. Este, de asemenea, mediul cel mai instabil. Deoarece mai multi dezvoltatori ar putea lucra în același timp, mediul CI are o mulțime de activități de build in desfășurare care se întâmplă în același timp. Ca rezultat, acest mediu poate se poate "rupe", și deseori se intampla. E în regulă - este menit să se rupă. Cheia este fixarea atunci când acest lucru se întâmplă.
- Teste: teste unitare, testarea integrării cu componente mocked, testarea UI cu date false

- Mediul de testare
  - Este mediul in care se testeaza functional fiecare story/schimbare care are loc intr-un sprint.
  - În acest mediu, avem aceeași infrastructură ca și în producție. Folosim un subset reprezentativ de date de producție - destul de aproape de datele de producție pentru a le testa.
  - Acest mediu este folosit si de Product Owner sa testeze functionalitatile dintr-un sprint.
  - Teste: Această fază este un moment potrivit pentru testarea exploratorie deschisă. În acest stadiu, sperati să nu prindeți doar probleme tehnice, ci și probleme potențiale ale experientei utilizator.

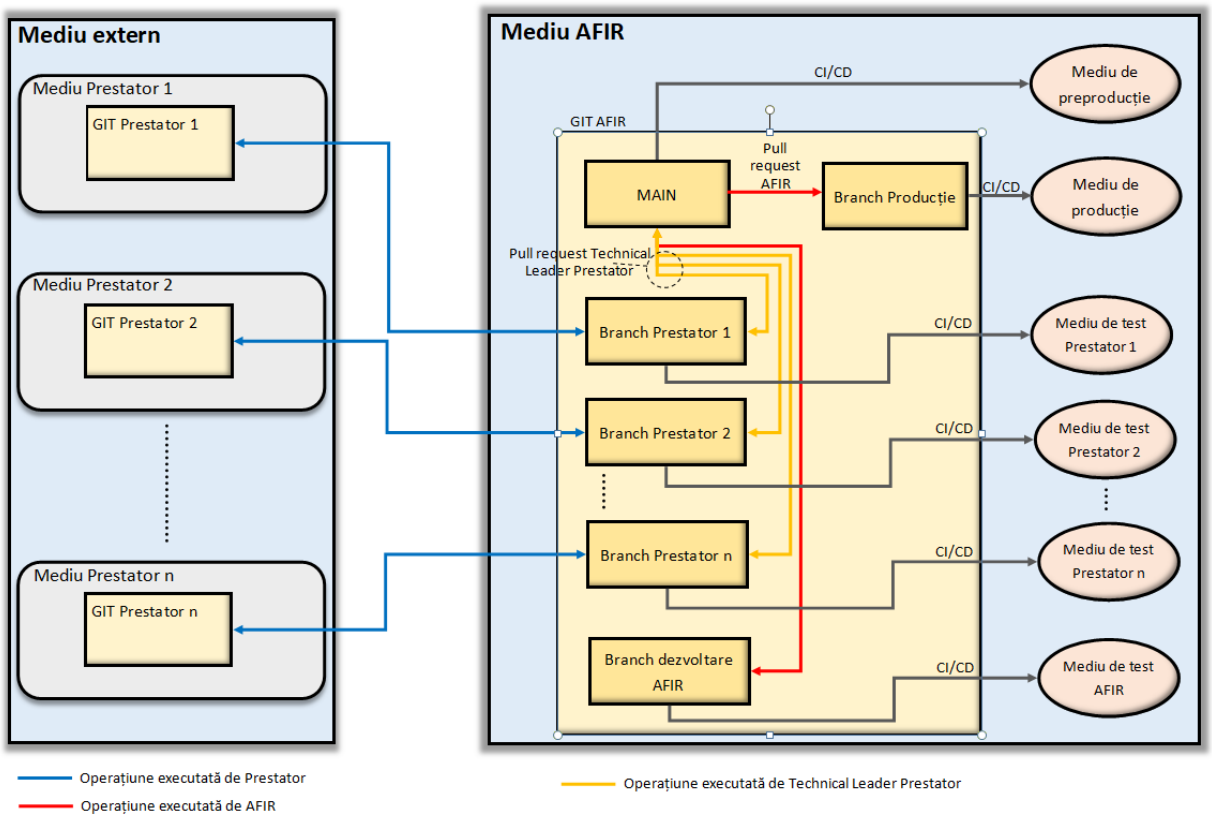
- Staging/Acceptanta:
  - Acesta este ultimul mediu înainte de producție. Scopul este acela de a avea un mediu aproape exact la fel ca și producția. Când ceva funcționează in acest mediu, puteți fi în mod rezonabil sigur că aceeasi versiune nu va eșua în producție și nu va provoca o întrerupere. Toate mediile vă ajută să găsiți probleme potențiale; acceptanta este verificarea finală. Aici este important să aveți aproape același număr de date ca și în producție. Acest lucru vă permite să efectuați testarea încărcării și să testați scalabilitatea aplicației în producție.

- Teste: testarea sistemului, testarea încărcării, testarea performanței și testarea securității.

- Productie

In cazul intr-un proiect derulat impreuna cu un contractor extern, mediile de dezvoltare, cel de CI si de testare interna sunt create si gestionate de catre prestator, mediul de testare de acceptanta si mediul de productie vor fi configurate initial de catre Prestator si mediul de productie va fi ulterior administrat de AFIR prin CI/CD

Metoda de lucru in acest caz este descris in schema de mai jos:



CONTINUOUS DELIVERY

Continuous Delivery (CD) este procesul de build, testare, configurare si deployment, pornind de la un mediu de build pana la un mediu de productie. Se formeaza un Release Pipeline care poate trece prin multiple medii de test si staging. In orice proiect, primul pas este de a stabili calea spre productie despre care vorbeam in capitolul anterior. Continuous Delivery este o practica Lean, al carei scop este obtinerea drumului cel mai scurt de la momentul in care o noua versiune de cod

sursa devine disponibila, pana cand aceasta versiune este livrata in productie. Prin automatizare, CD minimizeaza timpul de deployment, atat pentru functionalitati noi, cat si pentru a remedia incidente aparute in mediul operational.

Continuous Delivery a devenit o cerinta obligatorie pentru AFIR, in proiectele derulate sub acest framework. Pentru a livra valoare utilizatorilor finali, AFIR trebuie sa livreze in mod continuu si sa minimizeze erorile aparute in procesul de dezvoltare si deployment. Pentru aceasta se va folosi AzureDevOps, care ofera urmatoarele:

- O integrare deplina a modulelor pentru automatizarea proceselor de compilare, testare, si deployment, de exemplu oferind posibilitatea de a compila codul sursa la fiecare modificare facuta pe server, de a rula testele automate existente in sistem la fiecare compilare, si de a instala aplicatia compilata pe o gama variata de medii de testare. AzureDevOps ofera suport pentru o multitudine de limbaje de programare si platforme (inclusiv C#, Java, Python, HTML5, JavaScript, etc.), suport pentru modificarea/ personalizarea procesului de compilare, suport pentru auditarea pasilor executati in cadrul unui proces de compilare, rezultatele pasilor respectivi, dar si actiunile sau utilizatorii care au lansat procesul respectiv .

- Suport pentru automatizarea procesului de instalare (deploy), oferind module care sa ofere posibilitatea de a instala pachetele rezultate din procesul de build pe diverse masini de calcul, posibilitatea de a adauga pasi specifici proiectului in procesul de deploy, posibilitatea de a instala aplicatii/framework-uri predefinite pe respectivele medii de calcul (de exemplu IIS, Java, .NET Framework, etc). In plus, ofera suport pentru utilizarea de module externe, dezvoltate de terte persoane sau de catre echipa de dezvoltare.

- Suport pentru automatizarea procesului de testare, atat pe partea de unit testing, cat si pe partea de integration testing. Modulele respective se integreaza cu modulele de compilare si deployment, oferind astfel posibilitatea de a urmari in mod unitar intregul proces de integrare si livrare continua, de la compilare, la testare unitara, la deployment, la integration testing.

- Suport pentru gestiunea pachetele de instalare rezultate din procesul de compilare, suport pentru medii dedicate de testare, staging, sau productie, suport pentru identificarea si diagnosticarea mediilor in care instalarea nu s-a efectuat cu succes.

MONITORIZAREA CALITATII CODULUI

Monitorizarea calitatii codului are ca scop identificarea zonelor bogate in datorie tehnica (technical debt) ajutand echipele sa pastreze aceasta datorie intr-o marja controlabila. Spunem asta pentru ca datoria technica este axiomatica, va aparea intotdeauna pe parcursul unui proiect, ramanand la latitudinea echipelor modul in care ea este adresata in timp.

Datoria tehnica reprezinta setul de probleme, in cadrul unui proiect de dezvoltare software, care face ca progresul in a adauga noi functionalitati, si implicit valoare pentru client, sa fie ineficient. Aceasta submineaza productivitatea, facand codul greu de inteles, fragil, orice schimbare devenind consumatoare de timp, greu de validat si introducand fel de fel de activitati neplanificate care blocheaza progresul.

Datoria tehnica vine sub multe forme, de la probleme gasite in analiza codului, cod duplicat, complexitate ridicata, test insuficiente, sau nerelevante, pana la probleme din punct de vedere arhitectural sau tehnologic.

Deci, aceasta datorie trebuie gestionata, nu neaparat eliminata. Si prin urmare, trebuie masurata. Cand vorbim de rezolvarea unor probleme cauzate de datoria tehnica, acestea trebuie prioritizate printre dezvoltarea de noi functionalitati. Ca atare, impactul rezolvarii lor trebuie masurat si el, pentru a putea justifica necesitatea lor catre Product Owner.

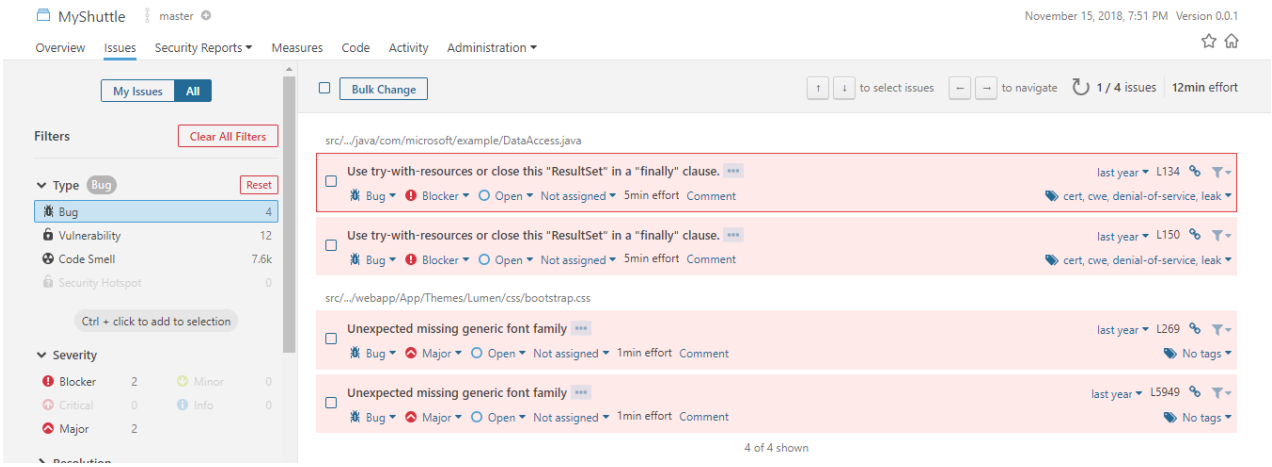
Solutia adoptata de AFIR pentru monitorizarea continua a calitatii codului este SonarQube, integrat cu AzureDevOps. Acesta ofera printre altele:

- Capabilitati de analiza a codului sursa pentru multiple proiecte si limbaje de programare, din perspectiva celor mai bune practici in domeniu, a vulnerabilitatilor percepute, a defectelor probabile, si a mentenabilitatii viitoare a codului. SonarQube ofera de asemenea indrumare, explicatii si sugestii pentru rezolvarea problemelor identificate.

- Capabilitati de monitorizare a nivelului de calitate al unui proiect, a progresului acesteia in timp, a dimensiunii proiectului si a riscurilor posibile.

- Suport pentru identificarea celor mai frecvente abateri (vulerabilitati, defecte, sau datorie tehnica), suport pentru activarea/dezactivarea abaterilor urmarite, suportand inclusiv definirea de abateri specifice proiectului si monitorizarea automata a acestora.

- Capabilitati de analiza pentru multiple limbaje de programare (include C#, Java, Python, JavaScript), capabilitati de integrare cu modulele de compilare din Team Foundation Server, suportand inclusiv oprirea procesului de compilare a codului in cazul in care nivelul dorit de



calitate a codului nu a fost atins.

INITIEREA UNUI PROIECT

Desfasurarea unui proiect, mai exact o privire de ansamblu asupra modului cum se imbina practicile descrise in metodologie/framework.

Initierea proiectului

1. Crearea viziunii si a backlog-ului initial

   Adesea, initiatorul proiectului are o idee despre produsului si posiblitatile sale, dar lipseste backlog-ul initial. O buna practica consta in organizarea unui workshop atat impreuna cu persoane din business dar si tehnice, de preferat si echipa de dezvoltare daca este cunoscuta la momentul acesta, cu scopul de a crea un Product Backlog initial impreuna. In acest moment se rafineaza si viziunea produsului, asigurandu-se si o intelegere reciproca.

2. Identificarea stakeholder-ilor, formarea echipei

Stakeholderi: aceastea sunt persoane care au cerinte de la produs si sunt practic motivul pentru care echipa se formeaza sa dezvolte software-ul. Ele colaboreaza cu Product Owner-ul pentru a defini User Story-urile
Product Owner: Reprezinta interesele stakeholder-ilor si are o intelegere solida a utilizatorilor finali, a business-ului si a tipului de sistem dezvoltat
Scrum Master: liderul servant care face tot posibilul pentru a ajuta echipa sa performeze
Echipa: un grup auto-organizat dedicat sa livreze un increment al produsului la finalul fiecarui sprint

In cazul proiectelor de anvergura, demersul poate implica mai multe echipe. De la caz la caz, se poate agrea daca fiecare echipa are propriul ei Scrum Master, sau exista un Scrum Master dedicat pentru toate echipele.

In cazul proiectelor dezvoltate cu prestatori externi, Product Owner-ul va fi din partea AFIR, iar echipa/echipele si Scrum Master-ul din partea prestatorului.

3. Crearea unei estimari de bugetare a Product Backlog-ului initial

   In faza aceasta se doreste crearea unei imagini de ansamblu asupra efortului necesar de a derula proiectul. Maniera preferata si pentru estimarea de bugetare este cea Agila, folosind story points, dar si extrapolate la nivel de man-days si durata. Pornind in acelasi timp de la intelegerea ca aceasta este pur si simplu o aproximare, avand marja de eroare aferent, iar rafinarile ulterioare ale backlog-ului vor tine cont de aceasta, avand grija ca functionalitatile agreate sa fie incluse, cu spatiu de manevra in ceea ce priveste nivelul de detaliu din story-urile granulare.

4. Prioritizarea functionalitatilor din Product Backlog

   Aceasta este momentul in care se poate face realmente o prima prioritizare a elementelor din backlog. In mare parte vorbim de Epic-uri, adica functionalitati mari care vor fi sparte ulterior. Se va lua in calcul valoare de business pe care o aduce fiecare functionalitate, dar in acelasi timp si complexitatea ei (data de estimare), precum si anumite constrangeri sau dependinte care au fost identificate pana acum si care probabil nu permit tot timpul permutarea Epic-urilor sau Story-urilor intre ele.

5. Definirea unei strategii de release

Este important ca in aceasta etapa sa se formuleze o strategie de release pentru a duce produsul in productie, strategie care trebuie agreata de toate partile implicate. Aceasta poate insemna o singura livrare la sfarsitul proiectului, sau livrari intermediare de versiuni ale aplicatiei fie dictate de prezenta unor anumite functionalitati din backlog (ex. livram versiunea 1.0 in momenul in care avem incorporate functionalitatile x, y, z), fie de un anumit interval de timp (ex. livram in productie o noua versiune la fiecare 2 luni). De aceasta strategie trebuie tinut cont in momentul in care se face (re)prioritizarea backlog-ului sau cand echipa si Product Owner-ul planifica munca dintr-un sprint (ex. in sprintul premergator unui release accentul va fi probabil mai mult pe stabilizare decat pe implementarea de noi functionalitati).

Sprintul 0

1. Intalnire de start al proiectului

   Momentul formal de incepere a proiectului, concretizat printr-o intalnire la care participa toate persoanele implicate: Echipa, Scrum Master, Product Owner, eventual si stakeholderi. In aceasta intalnire se defineste modul de lucru, sa clarifica care este “Definition of Done” (ce inseamna pentru fiecare cand un element din backlog este finalizat), inclusiv se agreaza asupra lungimii sprinturilor (1-4 saptamani). Scopul principal este ca toti cei implicati in proiect sa fie aliniati cu privire la modul in care se va desfasura implementarea.

2. Crearea proiectului in AzureDevOps

   Se creaza proiectul in AzureDevOps, se asigura ca toti cei implicati, inclusiv stakeholderii, au acces si drepturi conform cu rolul lor. Printre altele, Product Backlog-ul este completat cu toate informatiile existente in acest moment, se definesc echipele, sprinturile.

3. Instalarea mediilor de dezvoltare si testare

   Se instaleaza si configureaza mediile de dezvoltare si testare astfel incat echipa sa poate lucra cat mai eficient din prima zi a primului sprint.

   In cazul proiectelor derulate cu un prestator extern, de regula mediile de dezvoltare si testare

sunt in sarcina prestatorului, iar cele de acceptanta si productie vor fi detinute de AFIR. Totodata, Product Owner-ul si stakeholderii vor avea acces cel putin dupa fiecare sprint review pe mediul de testare.

4. Repository in Git

   Se configureaza si source control-ul, Repository in Git, definindu-se branch-urile necesare pentru modul de lucru agreat de echipa.

5. Arhitectura initiala si o prima structura a solutiei

   Tot in sprintul 0, se pun bazele unui prim design tehnic al aplicatiei. Acesta va fi o fundatie peste care echipa va implementa primele functionalitati, dar care va fi refactorizat si imbunatatit pe tot parcursul proiectului.

6. Pregatirea Story-urilor pentru Sprint 1

   Se continua rafinarea Product backlog-ului, pana cand sunt suficiente User Story-uri clare si bine definite in partea de sus pentru a fi preluate de echipa in Sprintul 1. Se vor prioritiza pentru primul sprint sau primele sprinturi Story-urile care tin de configurarea mecanismelor de Continuous Delivery.

7. Ceremoniile aferente sprinturilor sunt adaugate in calendar

In acest moment, echipele sunt pregatite sa deruleze sprinturile de dezvoltare.