2019

DIMENSIONAREA ECHIPEI DE TESTARE

CUM PUTEM IDENTIFICA ACEL “SWEET SPOT”

FLORIN MANOLESCU

Cuprins

Abstract

Lumea IT este complexa, iar echipele de testare intampina dificultati in a estima: ce activitati de testare putem livra, de cate persoane avem nevoie pentru a livra activitatile curente de testare, de cate persoane avem nevoie pentru a livra activitatile critice de testare din proiect, ce senioritate cautam spre angajare, cum putem livra mai multe activitati de testare cu un numar optim de persoane?

Acest lucru se poate transpune, cu usurinta, in lipsuri existente in procesul de testare, care pot cauza defecte scapate in productie, intarzieri in validarea produsului si o abordare neuniforma a testarii de la versiune la versiune.

In lucrarea de fata, propun un model care se doreste sa devina ajutorul echipei de testare. Intai, prin a intelege volumul de munca pe care il poate livra in dimensiunea actuala, apoi, oferind vizibilitatea necesara, prin facilitarea cresterii maturitatii echipei si a calitatii livrabilelor ei.

Introducere

Pe 30 iunie 1945, John Von Neumann publica raportul privind EDVAC, prima discuție documentată despre conceptul de program stocat și planul de arhitectură a calculatoarelor. Desi tanara, industria a castigat teren rapid. Tim Berners-Lee, Linus Torvalds, Bill Gates, Martha Lane Fox sau Steve Jobs au dezvoltat produsele si serviciile IT, amplificand impactul pe care acestea il au asupra lumii. Urmatorul pas? Standardizarea modului de lucru, primul pas spre maturizare.

Odata cu implementarea standardelor in modul de livrare, eficientizarea anumitor practici a capatat mai mult sens, scopul fiind obtinerea unor procese cat mai mature de dezvoltare, precum testarea automata, DevOps, Behavior-Driven Development (BDD) sau Test-Driven Development (TDD).

Testarea automata castiga tot mai mult teren in procesul general de testare ale produselor software. Acest lucru vine din cresterea complexitatii produselor pe care le dezvoltam. Exista insa si destule provocari care infraneaza explorarea si perfectionarea acestui tip de testare, dintre care dimensionarea corecta a echipei de testare este printre cele mai intalnite.

Cum echilibram activitatile de testare manuala si automata, raportat la volumul de munca din proiect? 1:2, 1:5, 1:10 sau 2:1 sunt cateva exemple de corelatii folosite pentru determinarea marimii echipei de testare raportate la marimea echipei de dezvoltatori. Limitarea acestei teorii vine din imposibilitatea aplicarii ei la un nivel general valabil. IT-ul este un Univers al particularitatilor, iar testarea trebuie sa ne raporteze la ele.

In continuare, va propun un model de dimensionare al echipei de testare automata, ce poate fi usor extins asupra intregii echipe de testare. Cu siguranta mai anevoioasa decat aplicarea unui model de dimensionare prestabilit, care nu tine seama de particularitati, aceasta metoda ofera companiilor numeroase avantaje pe termen lung.

Pentru stabilirea particularitatilor, este necesar insa sa intelegem complexitatea procesului de dezvoltare si care sunt factorii ce pot influenta acest model de calcul.

Factorii complexitatii: cum influenteaza modelul de calcul

Complexitatea isi are bazele in metodologiile aplicate, specificul organizatiei si structura ei, instrumentele folosite, maturitatea procesului de testare, dar si cum impacteaza Costul Calitatii evolutia organizatiei.

Metodologii

Agile, Waterfall si Lean aduc dupa ele modele diferite de lucru si, implicit, moduri diferite de a livra proiecte software. Complexitatea creste daca luam in calcul si diversele implementari ale acestor

metodologii precum V-Model, SCRUM, SaFe, Less sau Kanban, care fac cu atat mai dificila standardizarea proceselor de livrare.

Modelul de dezvoltare V-model

V-model aloca un tip de testare fiecarei activitati (analiza sau implementare): unit testing, component integration testing, system integration testing, system testing, user acceptance testing.

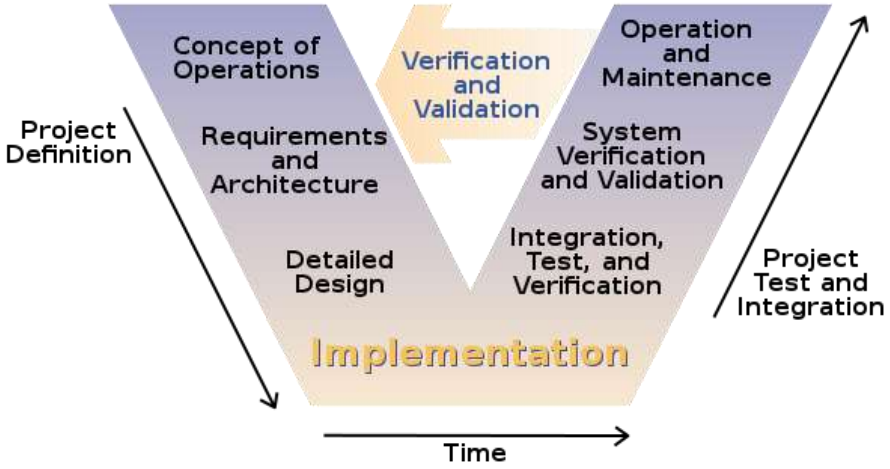

Fig.1 – Modelul de dezvoltare V-Model

Sursa – Wikipedia https://en.wikipedia.org/wiki/V-Model_(software_development)

Pentru a introduce si estima dimensiunea echipei de testare automata in acest model, trebuie sa definim diferite aspecte legate de scopul testarii. In literatura de specialitate, se recomanda distributia de mai jos:

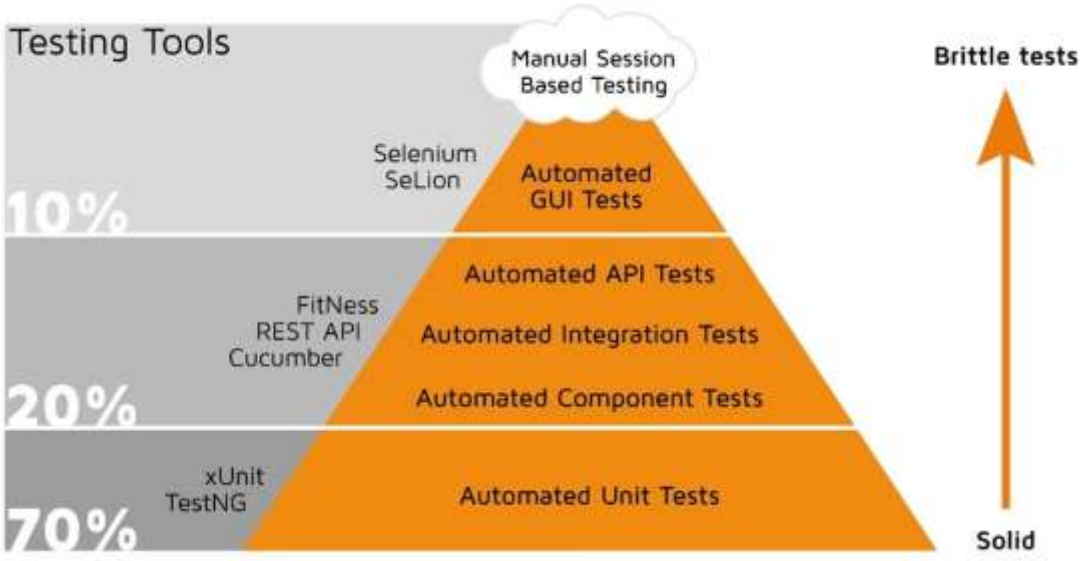

Fig.2 – Bune practice in distributia testelor automate grupate pe tipul testelor

Sursa: Testingreferences http://www.testingreferences.com/here_be_pyramids.php

Printre posibilitatile de implementare in proiecte, identificam doua momente distincte in care putem interveni:

- **In faza de inceput a proiectului**

Moment in care sunt definite toate procesele ce vor guverna dezvoltarea produsului, inclusiv cele specifice testarii automate. Infrastructura este si ea definita, luand in calcul si necesitatile testarii automate. Cu cat aplicam mai devreme testarea automata in cadrul procesului de dezvoltare, cu atat creste eficienta ei in proiect.

- **Pe parcursul proiectului**

Moment in care livrarea produsului software este in desfasurare, iar procesele, infrastructura si implementarea testarii automate fie nu sunt definite, fie pot fi imbunatatite. Exista masuri eficiente de imbunatatire, cum ar fi: prioritizarea scopului testarii, definirea si masurarea ROI (Return of Investment) al testarii automate.

Prioritizarea se poate face atat asupra tipului de testare (ex: regression, smoke), a componentelor sistemului dezvoltat, cat si asupra etapei din procesul de dezvoltare (ex: SIT, UAT). Definirea si calcularea indicelui ROI ofera incredere in privinta maturitatii procesului (definirea) si a eficientei lui (masurarea).

Cu siguranta ca, momentele fiind diferite, prioritatile testarii automate vor fi diferite. Important este sa ne asiguram ca release-ul curent si dezvoltarea produsului sunt automatizate in paralel.

Agile (SCRUM)

SCRUM e modelul de dezvoltare a carui caracteristica principala o reprezinta livrarea iterativa de functionalitati catre utilizatori. Principiile SCRUM trebuie adoptate de catre intreaga echipa de livrare, fapt ce implica testarea atat a functionalitatilor noi, cat si a celor deja livrate. Pentru ca timpul iteratiilor este scurt, importanta aplicarii testarii automate in toate fazele proiectului devine tot mai mare. Haideti sa vizualizam grafic implicatiile fiecarei parti in procesul de livrare:

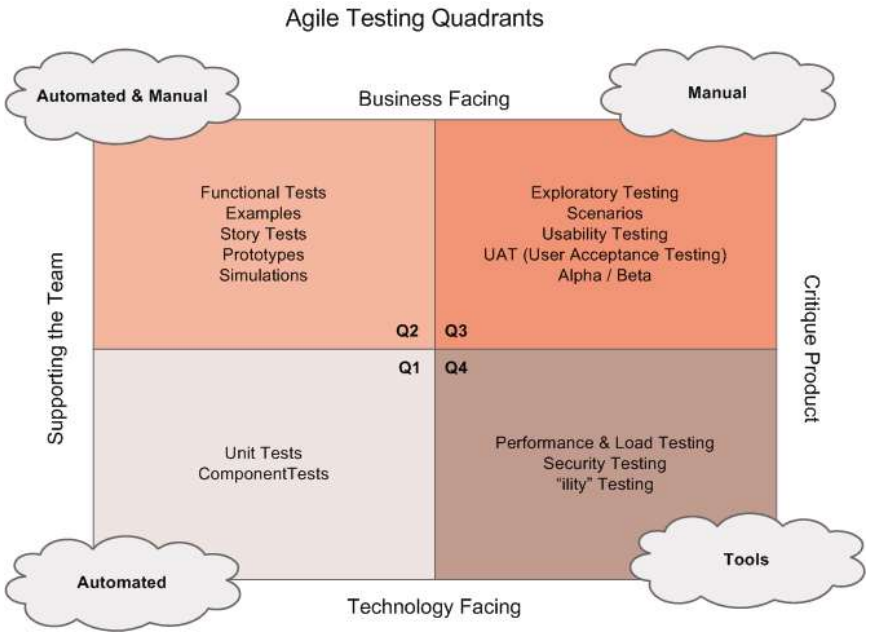

Fig.3 – Cele 4 cadrane ale testarii si distributia testelor

Sursa: Lisa Crispin www.lisacrispin.com

Daca ne uitam la cvadrantul testarii, recomandarea de implementare a testarii automate este preponderenta in zonele “Automated & Manual” (Q2) si “Automated” (Q1). Acesta este locul in care dorim sa patrundem cel mai mult cu testarea automata. In acest sens, cerintele produsului trebuie aliniate la scopul acestor cvadrante si la tipurile de testare asociate fiecaruia. Existenta unui proces matur de grupare a cerintelor devine pilonul procesului de livrare, deoarece acestea reprezinta datele de intrare pe baza carora se planifica si executa testarea.

Testarea automata aplicata in faza *sprintului* poate fi privita la nivelul criteriilor de acceptanta ale story-urilor. Pentru a avea o distributie corecta de teste manual si automate, este recomandat sa pastram o distributie similara piramidei din Fig 2. Aceasta distributie trebuie avuta in vedere din faza de conturare a criteriilor de acceptanta, avandu-se in vedere testarea lor. Story-ul va fi considerat livrat, prin completarea cu succes a Definition of Done (DoD), acest lucru insemnand ca toate testele planificate spre executie vor fi validate cu succes. Aceasta se face, bineinteles, pornind de la prezumţia ca infrastructura si arhitectura necesare sustinerii unei astfel de testari sunt deja implementate.

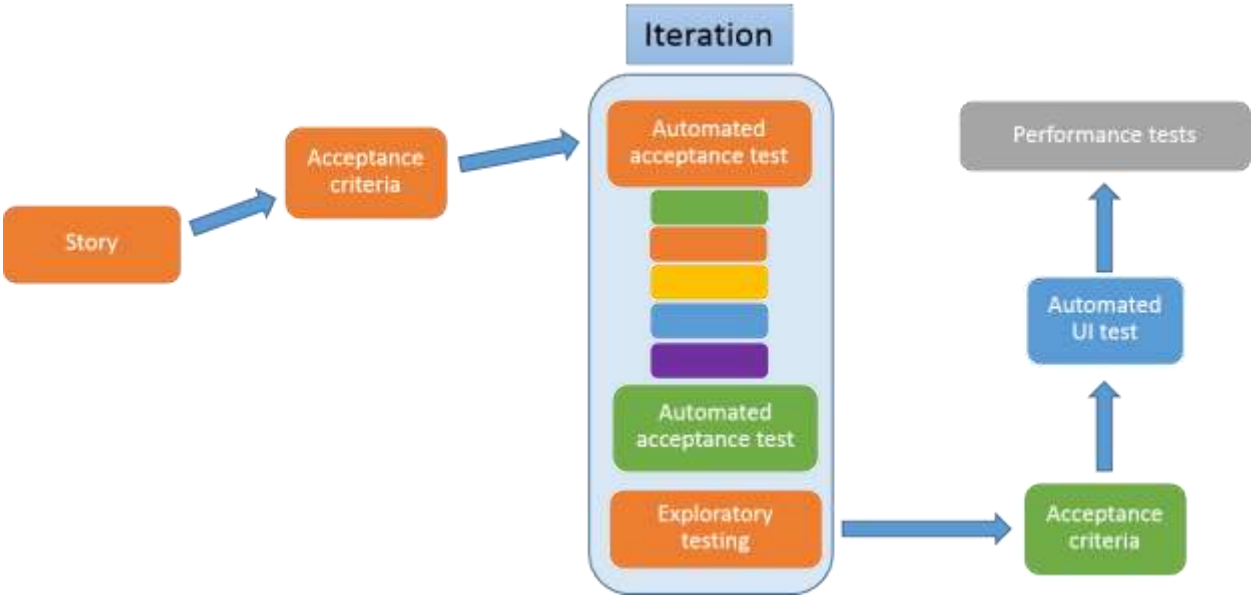

Fig.4 - Distributie a testelor necesare pentru acoperirea unui Story

Si aici, avem aceleasi doua perioade distincte de implementare:

- **Initierea proiectului**: momentul in care este definit DoD (Definition of Done). La inceputul proiectului, testarea automata este parte constitutiva a procesului de dezvoltare (infrastructura, criterii de acceptanta, masurare). Cu cat implementam testarea automata mai devreme in proces, cu atat are mai multe sanse de reusita.
- **Pe parcursul proiectului**: moment in care sprint-urile sunt in desfasurare si echipa livreaza, insa fara ajutorul testarii automate, care fie nu este implementata deloc, fie implementarea ei poate fi imbunatatita. Pentru revizuirea intregului proces, testarea identifica si masoara impactul sprint-urilor tehnice (cunoscut sub numele de *Technical Debt*). Faptul ca sprint-urile sunt in desfasurare permite masurarea CoQ al proiectului, iar masurarea costului permite si calcularea ROI dupa introducerea testarii automate.

Specificul organizatiei

Un lucru e sigur: de la agricultura pana la comert online, orice companie existenta pe piata in zilele noastre are o legatura mai mica sau mai mare cu industria IT. Exista companii care au ca activitate principala producerea de software si companii care folosesc si dezvolta software pentru intreprinderea activitatilor necesare functionarii departamentelor de suport precum Resurse Umane, Contabilitate, Marketing etc.

Asa cum este normal, alocarea unui anumit numar de persoane care sa se asigure de implementarea cu succes a proceselor de livrare software este direct proportionala cu specificul companiei, numarul produselor software utilizate si nivelul de business asupra caruia acestea au impact: operational, management sau strategic. Iar acest lucru nu poate fi schimbat.

Desi distributia echipei de testare, in cadrul echipei de livrare software este in general subunitara, exista si particularitati, in special in cazul industriilor in care functionarea corecta a software-ului este critica (ex. Industria medicala, spatiala, militara). Un exemplu in acest sens este NASA unde raportul intre testeri si dezvoltatori este de 5 la 1 (http://www.softwaretestingtricks.com/2007/04/software-testing-from-testers-eye.html). In acelasi timp, companiile care dezvolta software pentru indeplinirea activitatilor de suport vor aloca testarii resurse mai putine.

Structura organizatiei

Pe langa specificul ei, exista insa si alti factori care influenteaza contextul in care se afla organizatia. Aici includem modul in care este strucurata organizatia. Am identificat trei clasificari, asupra carora putem interveni:

- **Structurare in functie de echipele de proiect**

Scalarea se realizeaza pe verticala si accentul se pune mai mult pe zona de livrare, cu riscul de a nu avea standardizare per ansamblu. Depinzand de marimea organizatiei si de complexitatea proiectului, pot exista, in medie, intre 0 si 3 testeri per echipa.

- **Structurare in jurul profesiei**

Scalarea se realizeaza atat pe verticala, cat si in functie de profesie. Desi standardizarea este mai usor de implementat, ea poate atrage dupa sine diminuarea agilitatii echipei. In acest caz, echipa de testare este o echipa de sine statatoare, iar resursele sunt alocate in proiecte, in functie de cerere si de competenta fiecaruia in parte. Modul de lucru este definit in acest caz la nivelul intregii echipe de testare.

- **Structurare mixta**

Guvernanta este mixta, incluzand si echipe externe sau contractori individuali. Este cel mai des intalnita in cadrul companiilor mari, unde exista nevoia de organizare pe mai multe criterii. Pentru a facilita standardizarea, aici accentul se pune atat pe zona de proiect, cat si pe gruparea in functie de profesiei. Complexitatea livrarii rezida din faptul ca nevoia de coordonare vine din partea tuturor partilor implicate: profesie, proiect, departament.

Instrumentele si tehnologia utilizate

Complexitatea este generata de suita de intrumentele folosita, atat pentru dezvoltarea produselor software cat si pentru managementul procesului de development. Numarul instrumentelor utilizate in cadrul unui proiect poate varia in functie de maturitatea echipelor si a proceselor. Aceasta complexitate, multiplicata la nivelul organizatiei, poate genera un numar mare de instrumente in portofoliu. Acestui factor i se adauga si flexibilitatea organizatiei in a adopta instrumente noi.

Infrastructura, un alt factor de luat in calcul, este cel mai des intalnita in etapele de dezvoltare, testare si productie. Ea poate impacta si software-ul folosit in managementul procesului de development, ca de exemplu: mentenanta deficitara a produsului de *defect tracking* poate rezulta in intarzieri neprevazute.

Maturitatea testarii

In mod traditional, evaluarea maturitatii echipei de testare se face inspectand practicile care compun si guverneaza procesul de testare.

Cea mai folosita clasificare este **TMMi** (Test Maturity Model Integration). Acesta defineste cinci niveluri de maturitate carora le asociaza cate o tinta si obiective specifice. Desi este un model bun de evaluare, TMMi face abstractie de impactul factorului uman.

Un alt model de calcul este **Tuckman**, unde factorul uman este urmarit. Cu toate acestea, **Tuckman** nu defineste criterii exacte care sa permita masurarea evolutiei pe scara de referinta.

Sa luam un exemplu. Pentru a atinge o maturitate de nivel trei, obiectivul este sa integram testarea in ciclul de viata al procesului de dezvoltare software. Ce nu se specifica este definirea integrarii si a modului de implementare, lucru care impacteaza factorul uman in zona senioritatii echipei dar si a erorii umane.

O clasificare simplificata a maturitatii testarii, separata de principiile TMMi, ar putea fi:

Fara testare: Strategia de testare nu este definita in cadrul proiectului, ceea ce conduce la validarea deficitara a produsului. Situatia este caracterizata prin:

- Testare ad-hoc
- Nedocumentarea testarii si rezultatelor ei
- Absenta standardelor de calitate

Testare disfuncțională: Strategia de testare este definita in cadrul proiectului, insa nu este implementata corespunzator. Se caracterizeaza prin:

- Existenta unui proces de testare rigid
- Documentarea testarii si rezultatelor ei
- Testarea este definita doar in anumite etape ale procesului de dezvoltare

Testare corespunzătoare: Strategia de testare este definita si implementata corect in cadrul proiectului, si presupune:

- Existenta unui process flexibil si eficient de testare
- Definirea si implementarea testarii in toate etapele dezvoltarii produsului
- Imbunatatirea continua a procesului de testare

In literatura de specialitate exista mai multe clasificari ale maturitatii echipei de testare. Daca privim lucrurile prin spectrul automatizarii, si pentru a nu ingreuna prea mult lucrurile, putem grupa aceasta maturitate in cinci categorii:

1. Lipsa testarii automate

   Rezervata echipelor care nu au implementat practici de testare automata iar intreaga metodologie de testare se rezuma la cea manuala. In acest caz, testarea de regresie fie dureaza foarte mult, fie este rulata doar in subgrupuri de teste, frecventa cu care este executata fiind foarte redusa.

2. Implementare insuficienta (disfunctionala)

In aceasta categorie intra echipele care au implementat automatizarea proceselor de testare, insa nu pentru toate tipurile de testare recomandate. Specifice acestei clasificari este si lipsa senioritatii necesare in cadrul echipei, precum si rularea testelor la nivel local cauzata de lipsa unei infrastructuri dedicate testarii automate.

3. Implementare insuficienta (best practices)

Aici incadram echipele care corespund criteriilor de mai sus, la care adaugam si senioritatea necesara pentru automatizarea diferitelor tipurilor de teste. In ciuda existentei unei infrastructuri corespunzatoare de testare automata, tipurile de teste acoperite sunt limitate.

4. Implementare completa (disfunctionala)

Caracteristica echipelor de testare ce au implementata testarea automata in toate tipurile de testare recomandate. Disfunctionalitatea poate fi cauzata de mai multi factori: lipsa senioritatii necesare in cadrul echipei de testare, neconcordanta cu scopul testarii automate, lipsa infrastructurii sau distributia necorespunzatoare a testelor automate.

5. Implementare completa (best practices)

Aceasta categorie este rezervata echipelor ce au implementata testarea automata pentru toate tipurile de testare recomandate. De asemenea, senioritatea echipei permite indeplinirea cu success a tuturor sarcinilor atribuite. Aceasta categorie poate fi vazuta ca un model spre care pot aspira toate echipele incadrate in celelalte categorii.

Costul calității (CoQ)

Orice companie care dezvolta si livreaza software are un cost, denumit **Costul Calitatii**, care reprezinta suma tuturor costurilor pe care oganizatia le suporta cu scopul de a asigura calitatea fiecaui produs oferit.

Pentru a masura distributia costului, au fost definiti termenii de **Cost de Conformanta** (gasirea defectelor, prevenirea defectelor) si **Cost de Non-Conformanta** (repararea defectelor, re-testing, imaginea companiei, pierderea de business etc).

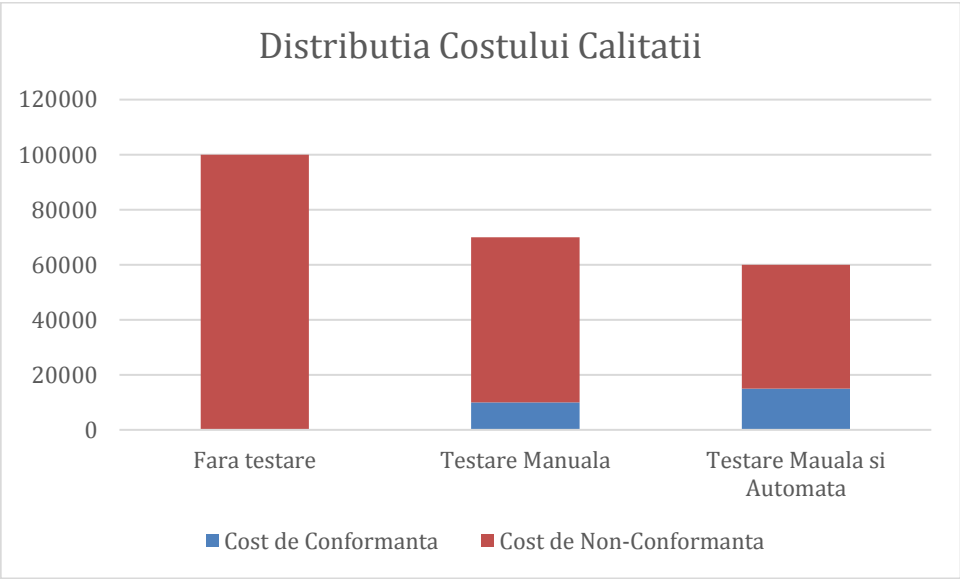

Fig. 5 - Distributia Costurilor de Conformanta si Non-Conformanta in Costul total al Calitatii

Comparata cu situatia in care nu exista testare in procesul de livrare software, introducerea testarii manuale scade dramatic Costul Calitatii, alocand o parte din el echipei de testare si nevoilor ei. Prin adaugarea componentei de testare automata, desi Costul de Conformanta creste prin adaugarea de tool-uri specifice testarii automate si a costurilor cu salariile pentru Automation Engineers, compania reuseste sa scada si mai mult Costul total al Calitatii.

Cum putem identifica “sweet spot-ul”?

In aceasta distributie complexa, definirea unui *sweet spot* universal-valabil este dificila. Solutia vine din aplicarea unor concepte si abordari vaste, care sa satisfica mai multe din modelele identificate.

Mai intai, trebuie sa regandim modul in care privim situatia. Daca pana acum plecam de la factorul uman, acum incepem prin a analiza activitatile specifice testarii automate.

Primul pas este listarea tututor activitatilor de testare necesare pentru finalizarea proiectului, la care adaugam caracteristicile specifice fiecaruia in parte. La sfarsitul exercitiului, va rezulta o matrice de care ne vom putea folosi pentru dimensionarea echipei de testare automata si, de ce nu, a intregii echipe de testare.

Matricea

Desi putem construi matricea raportandu-ne doar la activitatile curente ale echipei de testare, o alta abordare e includerea tuturor activitatillor ce ar putea fi livrate de catre echipa de testare, pe tot parcursul proiectului, intr-un caz ideal. Astfel, putem folosi maturitatea echipei de testare, atat cea actuala cat si cea ideala, pentru a dimensiona echipa (Anexa 1).

Activitate	Zona	Senioritate	Alocare (MD	Prioritate
Defect management - clarity of defects	Defect mgmt	Junior	0.1	High
Test management - Prioritization	Management	Medium	0.2	High
Test planning - test case review	Planning	Medium	0.4	High
Test automation planning - script code review	Planning	Medium	0.4	High
Test automation planning - script review	Planning	Medium	0.3	High
Test analysis - functional requirement analysis	Analysis	Medium	0.3	High
Test analysis - User story acceptance criteria analysis	Analysis	Junior	0.3	High
Test analysis - Non-functional requirement analysis	Analysis	Senior	0.3	High
Test execution - test environment readiness	Execution	Medium	0.1	High
Test execution - Test case execution	Execution	Junior	0.6	High
Test execution - Automation test suite execution	Execution	Junior	0.4	High
Test execution - Performance test scenario execution	Execution	Medium	0.3	High
Test execution - Unit test suite execution	Execution	Medium	0.1	High
Test execution - Component test suite execution	Execution	Medium	0.1	High
Test execution - System tests suite execution	Execution	Medium	0.2	High
Test execution - System integration suite execution	Execution	Medium	0.6	High
Test execution - User acceptance suite execution	Execution	Medium	0.8	High
Test execution - Smoke test suite execution	Execution	Medium	0.1	High
Test execution - Sanity test suite execution	Execution	Medium	0.1	High
Test execution - Regression suite execution	Execution	Medium	0.5	High
Test automation planning - framework mainenance	Planning	Medium	0.3	High

Foto: Matrice de activitati filtrate pe baza prioritatii in executie

Linii

Toata complexitatea de care vorbeam pana acum va crea o lista specifica fiecarei organizatii. Activitatile sunt unele generice si se pot grupa in:

- Arhitectura
- Analiza
- Planificare
- Executie
- Investigare
- Mentenanta
- Training
- Raportare.

Spre referinta in definirea activitatilor, putem folosi standarde internationale (ex. ISO-9001), bune practici din industrie sau din experienta profesionala. Cu cat descrierea activitatilor este mai granulara, cu atat vom putea estima mai precis, iar rezultatele obtinute vor fi mai relevante.

Vom identifica componentele fiecarei activitati principale, obtinand asadar subactivitati relevante pe care le transpunem grafic prin adaugarea de linii noi in tabele.

Distributia activitatilor de testare poate sa varieze si ea in cadrul profesiilor, prin existenta sau inexistenta unor echipe de support (ex. DevOps). Diferitele niveluri de maturitate ale unei activitati generice de testare pot ajuta in detalierea unei liste precum cea de mai jos.

Lista exhaustiva de subactivitati confera diversele niveluri ale matricei: unde suntem acum, punctele cheie pe care trebuie sa le parcurgem pentru cresterea maturitatii si a calitatii, dar si unde ne dorim sa ajungem, acel ideal cu care trebuie sa incepem acest exercitiu.

Primul pas va fi definirea unei prime liste de activitati pe care le putem livra in testarea automata, adaptata contextului organizatiei. Lista trebuie vazuta ca o grupare de categorii, sub care vom putea grupa una sau mai multe sub-activitati:

- Framework architecture design
- Framework component development
- Framework maintenance
- Script development
- Automation test suites execution
- Root cause investigation
- Defect management
- Infrastructure management
- Coaching & training
- Code & test case review
- *Activitati planning*
- *Analiza a requirement-urilor*
- *Exploratory testing*
- *Scriere de test case-uri (manual)*
- *Research & development*
- *Raportare*

Fig. 6 – Lista initiala de activitati de testare de la care se poate incepe definirea maticei

Putem creste si mai mult complexitatea acestei matrici prin adaugarea activitatilor de testare manuala, obtinand astfel dimensionarea intregii echipe de testare.

Spre exemplu, o activitate precum S*cript Development* poate fi detaliata, incluzand atat activitati manuale cat si automate, ca mai jos:

- Scope analysis
- Scope clarification
- Scope definition
- Requirement review session
- System architecture review session
- Infrastructure analysis
- Manual test case review
- Manual test case updates
- Test automation framework analysis

- Test automation component identification
- Component analysis & creation
- Impact assessment of component definition on existing scripts
- Updates on existing scripts
- Script definition
- Design patterns for script definition
- Script integration into test suites
- Script execution
- Script results reporting

Fig. 7 – Detalierea unei activitati principale in sub-activitati

In urma acestei detalieri, vom obtine subactivitati duplicate, subactivitati ce compun mai multe activitati majore din cadrul procesului de testare.

Coloane

Asa cum spuneam mai sus, odata ce determinam activitatile, este necesar sa le evaluam raportandu-le la anumite caracteristici. Acestea pot fi atat generale (aplicabile in majoritatea proiectelor si organizatiilor) dar si specifice (aplicabile intr-un grup mai restrans de proiecte si organizatii).

In completarea coloanelor, va trebui sa facem un exercitiu interesant: pe unele le vom completa pentru toate activitatile listate (ex. Senioritatea), pe altele le vom completa avand in vedere doar activitatile livrate in acest moment de catre echipa (ex. Alocarea).

Senioritatea

Senioritatea este un atribut ce poate fi alocat fiecarei activitati. Tinand cont de complexitatea proiectului in evaluare, putem identifica senioritatea necesara realizarii fiecarei activitati. Putem, pentru anumite activitati, sa variem senioritatea, acest lucru oferindu-ne mai tarziu un interval in cadrul caruia care putem jongla cu senioritatea necesara in echipa. Apreciem astfel ca respectiva activitate poate fi realizata de persoane de senioritate diferita. Acest lucru trebuie insa avut in vedere in momentul in care vom aplica urmatorul atribut: alocarea.

Alocarea

Alocarea este un atribut foarte important in cadrul acestei matrici deoarece ea determina efortul ce va fi depus in proiect pentru acea activitate. Vom alege o unitate simpla de masura: *Man Day* (MD), care reprezinta o zi de lucru a unei persoane din echipa, de o senioritate variabila, identificata deja in cadrul fiecarei activitati.

Valorile ce trebuie completate reprezinta o estimare a alocarii ideale, si nu a alocarii curente din proiect. Aceasta abordare permite identificarea activitatilor deja livrate de catre echipa de testare, dar care sunt livrate intr-un mod neoptimizat.

Deoarece valoarea acestui atribut este data in urma unei estimari, o putem rafina pe parcursul proiectului, lucru care va modifica si rezultatul dimensionarii, permitandu-ne atat sa optimizam cat si sa imbunatatim activitatile identificate.

Putem folosi alocarile definite si pentru a identifica posibilele activitati ce pot fi detaliate si mai mult, pentru o mai buna acuratete a matricei. Pentru asta, putem seta o anumita limita pentru alocare, similar cu *Agile Story Points* (ex. “Maximul alocarii/activitate – 1.5 MD”). Ca in planificarea tipica practicii Agile, cautam sa aducem toate activitatile intr-o zona de complexitate cat mai usor de controlat. Aceasta practica poate evidentia foarte usor activitatile ce necesita revizuire si poate fi folosita pentru evaluarea informatiei din matrice.

Prioritatea

Un bun criteriu de filtrare a tabelului, prioritatea fiecarei activitati poate fi alocata intr-o faza initiala si modificata pe parcurs in functie de gradul de implementare al fiecarei activitati.

Aplicata pe un subgrup de activitati cum ar fi cele necesare testarii automate , matricea despre care vorbeam poate arata asa:

Activitate	Senioritate	Alocare (MD)	Prioritate
Framework architecture design	Senior	0.3	High
Framework component development	Mid-Senior	0.7	High
Framework maintenance	Mid	0.3	Low
UI Script development	Junior-Mid	1	Medium
Database script development	Junior-Mid	2	Medium
API script development	Mid-Senior	2	Low
Automation test execution	Junior	0.3	Medium
Root cause investigation	Junior-Mid	0.3	Medium
Defect management	Junior-Mid	0.3	Low
Infrastructure management	Mid-Senior	0.2	Low
Coaching & training	Mid-Senior	0.3	Medium
Code & test case review	Mid	0.2	Medium

Fig.8 – Alocarea de prioritati pentru fiecare activitate de testare definite

Alte coloane

Fiecare proiect are particularitatile lui pentru care foloseste un set de tehnologii mai mult sau mai putin comune. In acest context, in fiecare proiect poate aparea nevoia de adaugare a altor coloane relevante care sa adreseze particularitatile sau nevoile specifice (ex. cunostinte de Load Runner, JIRA, Java).

Fiecare poate deveni un atribut, in acest caz, in coloanele respective, alocandu-se note intr-o scala unitara (aceeasi pentru toate coloanele definite; recomandare 0-5).

Utilizare

Odata completata matricea, putem aduce un plus de relevanta intr-un proces care, pana acum, a avut in vizor doar modelul “ideal” al testarii automate. Din lista exhaustiva de activitati, vom alege doar activitatile de testare pe care echipa le livreaza in acest moment.

Filtrarea

Aceasta prima filtrare ne poate ajuta sa identificam dimensiunea actuala necesara echipei de testare automata. Ea poate fi aplicata asupra fiecarui atribut, in functie de relevanta lui in cadrul proiectului. Putem, de exemplu, filtra doar activitatile cu prioritate ridicata sau doar activitatile ce necesita senioritate ridicata. Este important sa cunoastem si posibilele dependente intre activitati (ex. nu vom putea executa teste automate fara a avea un framework de testare minim dezvoltat).

Activitate	Senioritate	Alocare (MD)	Prioritate
Framework architecture design	Senior	0.3	High
Framework component development	Senior	0.7	High
UI Script development	Mid	1	Medium
Automation test execution	Junior	0.3	Medium
Code & test case review	Mid	0.2	Medium

Fig.9 – Filtrarea aplicata pe activitati High si Medium

Alinierea alocarilor

Pentru aliniere si relevanta, dupa ce vom finalia matricea, rezultatele obtinute pentru fiecare activitate vor fi inmultite cu numarul zilelor din luna, sprint sau release. Acest rezultat, va fi apoi impartit cu un numar de zile relevant etalonului ales: numarul de zile din sprint sau release.

Activitate	Senioritate	Alocare (MD)	Prioritate
Framework architecture design	Senior	(0.3 x 5)/10 = 0.15	High
Framework component development	Senior	(0.7 x 5)/10 = 0.21	High
Framework maintenance	Mid	(0.3 x 10)/10 = 0.3	Low
UI Script development	Mid	(1 x 10)/10 = 0.1	Medium
Automation test execution	Junior	(0.3 x 10)/10 = 0.3	Medium
Code & test case review	Mid	(0.2 x 8)/10 = 0.16	Medium

Fig.10 – Alinierea alocarilor cu perioadele relevante de release

Ajustarea estimarilor

Exceptand activitatile de testare, valorile introduse in restul coloanelor reprezinta estimari. Cum estimarile nu sunt niciodata exacte, cel putin nu din prima incercare, putem rafina, uneori si zilnic aceste estimari. Astfel va rezulta un document viu, care va reflecta nevoile proiectului in orice moment.

De aceea este important ca aceste valori sa fie facute cunoscute intregii echipe.

Dimensiunea echipei	Senior	Mid	Junior
TOTAL	0.36	0.56	0.3

Fig.11 – Necesarul de alocare pe activitati de testare in functie de senioritate

In momentul gruparii activitatilor si a determinarii dimensiunii echipei, nu este obligatoriu sa grupam doar activitatile de o anumita senioritate. Putem grupa activitati de senioritate diferita, caz foarte des intalnit in viata de zi cu zi, unde nu intreprindem activitati de aceeasi senioritate, constant, pe tot parcursul zilei.

Concluzie

Exista doua posibile deviatii prin care putem alinia dimensiunea echipei la valorile identificate in matricea completata:

- **Numarul de resurse din echipa de testare automata este mai mic decat valoarea alocarilor identificate cu ajutorul matricei**.

In acest caz, este foarte importanta comunicarea rezultatelor si identificarea actiunilor ce pot fi derivate din responsabilitatea echipei de testare automata, activitate necesara pentru alinierea asteptarilor (activitatile) cu livrabilele (capacitatea echipei).

- **Numarul de resurse din echipa de testare este mai mare decat valoarea alocarilor identificate cu ajutorul matricei**.

Acesta este un foarte bun punct de plecare in imbunatatirea procesului de testare, deoarece asigura capacitatea de manevra in cadrul echipei. Aceasta poate fi redirectionata atat pentru a facilita imbunatatirea unor activitati de testare deja livrate, cat si pentru alocarea unor activitati noi.

Putem extinde foarte usor aceasta matrice, raportandu-ne la intreaga echipa de testare, si nu doar la activitati din sfera testarii automate. Pe langa evaluarea dimensiunii echipei, aceasta matrice poate, de asemenea, sa contribuie la:

- Ajustarea implementarii activitatilor de testare, in cazul unor discrepante intre dimensiunea echipei si rezultatul obtinut.
- Evaluarea senioritatii si dimensionarea echipei necesare pentru cresterea numarului de activitati de testare acoperite.
- Colaborarea intr-un mod mai transparent cu echipele de HR in recrutare, oferind predictibilitate, atat in zona asteptarilor cat si a senioritatii.

Acest livrabil, poate fi folosit si in organizatie pentru a creste maturitatea echipei de testare si pentru a crea o structura mai eficienta a echipei. Pentru aceasta, vom realiza o matrice la nivel de organizatie, in care datele de intrare vor veni din matricea fiecarui proiect. Eliminarea duplicatelor va constitui primul pas, acest lucru asigurand elemente unice in lista.

In momentul in care avem o lista de elemente unice, valida la nivelul intregii organizatii, putem trece la urmatoarele actiuni:

- **Standardizare.** Pentru a capata maturitate si predictibilitate, procesul de testare automata si activitatiile de testare automata identificate trebuie standardizate intr-o masura in care sa satisfaca nevoia organizatiei. Acelasi tip de matrice poate fi definita si pentru identificarea prioritatilor in standardizare. Ea poate fi construita in jurul activitatilor identificate si a proceselor ce pot compune sau impacta respectiva activitate (ex. activitatii de *API Script Development* ii sunt identificate trei procese cu impact direct: test case management, requirement management, deployment). Odata implementata, standardizarea permite dezvoltarea urmatoarei zone.

- **Automatizare.** Aceasta zona poate fi augmentata si prin alocarea unor procese din zona DevOps (*Build & Deploy*, *Environment Preparation*). Atragerea practicilor DevOps, in cazul in care exista senioritate, poate reprezenta o buna oportunitate de a eficientiza testarea (atat automata, cat si manuala).

- **Definire.** Avem in vedere echipele dedicate sa deserveasca una sau mai multe activitati de testare automata (*Framework Architecture Design*, *Framework Component Development*).

Este recomandat ca, in cazul implementarii unor astfel de echipe, modelul de dezvoltare sa fie adecvat echipei si scopului ei, si nu neaparat aliniat cu modelul de livrare al proiectului.

De aceea, organizarea de tip Kanban este mai eficienta in cazul acestor echipe dedicate: ele nefiind nevoite sa se alinieze lansarilor din proiecte, ci mai degraba sa livreze cat mai repede, in orice proiect, pe baza cererii, a incarcarii si a prioritatilor.

Aplicand o limitare a WIP (Work in Progress), echipa centrala poate aduce vizibilitate si predictibilitate in livrarea si mentinerea *Framework-ului* de testare al organizatiei. Acestei echipe ii pot fi alocate si cerinte din zona DevOps, asigurand astfel o incarcare constanta.

Actionand asupra activitatilor identificate, putem lucra in fiecare zi la scaderea Costului Calitatii si redistribuirea din Cost de Non-Conformanta, in Cost de Conformanta. Acest lucru face ca, in principiu, fiecarei activitati sa ii poata fi calculat si atribuit un cost estimat.

Desi importante de luat in calcul pentru definirea matricei, instrumentele folosite la nivelul organizatiei sau al echipei nu impacteaza cu nimic completarea sau posibilitatea implementarii rezultatelor obtinute prin folosirea ei. Maturitatea echipei reprezinta insa diferentiatorul cheie, atributul pe baza caruia rezultatele se pot schimba.

Aceasta metoda poate fi aplicata in orice companie, atat pentru crearea unei echipe noi sau, in cazul unei echipe existente pentru a dimensiona efortul de testare la marimea echipei. Metoda poate evidentia rapid o incarcare inadecvata, facilitand comunicarea acestui lucru catre management, dar poate servi si ca un indicator al maturitatii curente dar si ideale a echipei de testare.

Anexa 1

Activitate	Zona	Senioritate	Alocare (MD)	Prioritate
Test management - test culture	Management	Senior	0.3	Medium
Defect management - clarity of defects	Defect mgmt	Junior	0.1	High
Test mindset - team effort	Management	Senior	0.2	Medium
Test management - Prioritization	Management	Medium	0.2	High
Test management - Risk based testing approach	Management	Senior	0.2	Medium
Tools - toolstack onboarding	Tools	Medium	0.1	Low
Tools - toolstack R&D	Tools	Medium	0.2	Low
Tools - toolstack maintenance	Tools	Medium	0.2	Medium
Tools - toolstack administration	Tools	Medium	0.1	Low
Tools - toolstack support	Tools	Junior	0.2	Medium
Test execution - documentation to support test execution	Execution	Medium	0.2	Medium
Test planning - communication	Planning	Senior	0.2	Medium
Test execution - communication	Execution	Medium	0.2	Low
Test management - communication	Management	Senior	0.2	Medium
Test reporting - communication	Reporting	Medium	0.2	Low
Test planning - test case design	Planning	Medium	0.5	Medium

Test planning - test case review	Planning	Medium	0.4	High
Test planning - test case update	Planning	Medium	0.1	Medium
Test automation planning - script design	Planning	Junior	0.6	Medium
Test automation planning - script code review	Planning	Medium	0.4	High
Test automation planning - script review	Planning	Medium	0.3	High
Test automation planning - script maintenance	Planning	Junior	0.2	Low
Test reporting - written status communication	Reporting	Junior	0.2	Low
Test reporting - deliverable preparation	Reporting	Medium	0.2	Low
Test analysis - business requirement analysis	Analysis	Medium	0.2	Medium
Test analysis - functional requirement analysis	Analysis	Medium	0.3	High
Test analysis - User story acceptance criteria analysis	Analysis	Junior	0.3	High
Test analysis - Architecture analysis	Analysis	Medium	0.3	Medium
Test analysis - Non-functional requirement analysis	Analysis	Senior	0.3	High
Test analysis - Product workshops	Analysis	Medium	0.1	Low
Test analysis - Product trainings	Analysis	Medium	0.1	Medium
Test execution - test environment readiness	Execution	Medium	0.1	High
Test execution - Test case execution	Execution	Junior	0.6	High
Test execution - Automation test suite execution	Execution	Junior	0.4	High
Test execution - Performance test scenario execution	Execution	Medium	0.3	High
Test execution - Execution results investigation	Execution	Medium	0.2	Medium
Test planning - Test environment planning & preparation	Planning	Medium	0.2	Medium
Test planning - test data preparation	Planning	Medium	0.3	Medium
Test planning - test planning documentation	Planning	Medium	0.2	Low
Defect management - root cause analysis	Defect mgmt	Medium	0.2	Medium
Defect management - Workflow standardization	Defect mgmt	Junior	0.1	Medium
Defect management - Approach standardization	Defect mgmt	Junior	0.1	Medium
Test analysis - Change reques analysis	Analysis	Medium	0.2	Low

Test analysis - Production defect analysis	Analysis	Junior	0.1	Medium
Test execution - Unit test suite execution	Execution	Medium	0.1	High
Test execution - Component test suite execution	Execution	Medium	0.1	High
Test execution - System tests suite execution	Execution	Medium	0.2	High
Test execution - System integration suite execution	Execution	Medium	0.6	High
Test execution - User acceptance suite execution	Execution	Medium	0.8	High
Test execution - Exploratory suite execution	Execution	Medium	0.4	Medium
Test execution - Smoke test suite execution	Execution	Medium	0.1	High
Test execution - Sanity test suite execution	Execution	Medium	0.1	High
Test execution - Regression suite execution	Execution	Medium	0.5	High
Test management - release planning	Management	Medium	0.2	Medium
Test management - Risk mitigation	Management	Senior	0.2	Medium
Test management - Release test documentation	Management	Medium	0.1	Low
Test automation planning - framework development	Planning	Senior	0.3	Medium
Test automation planning - framework mainenance	Planning	Medium	0.3	High
Test Management - Release reporting	Reporting	Medium	0.1	Medium
Test Management - Test status reporting	Reporting	Medium	0.1	Medium