Universitatea Politehnică Bucureşti
Facultatea de Electronică,Telecomunicaţii şi Tehnologia Informaţiei

INGINERIE SOFTWARE

SISTEME SOFTWARE CRITICE

Studenţi: Ghiga Alexandru,441A
Ilie Marian-Danut.443A
Zamfir Oana-Liliana,441A
Zăpuc Gabriela,443A

CUPRINS

I Introducere [12]

Un sistem software critic este acel sistem a carui eroare sau proasta functionare poate duce la pagube umane cat si economice severe.Astfel sistemul este dependent de urmatoarele patru atribute:disponibilitate, fiabilitate, siguranta si securitate .Pentru a obtine aceasta dependenta trebuie evitate greselile ,erorile eliminate iar in cazul aparitiei acestora limitarea pagubelor.

Proprietatea de dependenta a unui sistem critic este reflectata de gradul de incredere care il acorda utilizatorul,sistemului .Altfel spus ,sistemul nu va da gresi in conditii normale de folosire.Disponibiliataea din acest punct de vedere poate fii considerata o conditie de cat de practic este sistemul ,un sistem poate fii practic si fara sa fie fiabil.

Costul dependentei tinde sa creasca exponential pe masura ce sunt cerute nivele inalte ale dependentei.

Sistemele software critice pot fi impartite in trei mari sub categorii :

1. Sisteme de siguranta critice .Sunt acele sisteme a caror proasta functionare sau stricare poate duce la ranirea sau chiar pierderea de vieti umane cat si la distrugeri ecologice de proportii.Un exemplu in acest caz fiind sistemul software al unei uzine chimice.
2. Sisteme de scop critice. .Sunt acele sisteme a caror proasta functionare sau stricare poate la neindeplinirea scopului activitatii.De exemplu sistemul folosit in procesul de navigare al unei nave spaciale.
3. Sisteme economice critice .Sunt acele sisteme a caror proasta functionare sau stricare pot provoca pagube financiare semnificative firmei care le folosesc.

Costul mare pe care un sistem critic il are la defectare implica folosirea unei tehnologii de incredere.Astfel se folosesc tehnologii si metode relativ mai vechi dar bine testate in practica decat cele noi alecaror puncte forte si slabiciuni sunt necunoscute in practica.Din aceasta cauza creatorii arhitecturilor sistemeor critice tind sa fie mai conservatori.Costul verificarii si validarii sistemului poate chiar depasi 50% din totalul costului de dezvoltatre.

Sistemele de control sunt complet automate in numar foarte mic, cele mai multe

sistemele critice sunt sisteme socio-tehnice unde oamenii monitorizează si controlează

functionarea sistemelor. Costurile de esec a sistemelor critice sunt, de obicei atât de mari, încât avem nevoie de oameni care poat face fata unor situatii neprevăzute si care poate redresa situatia de multe atunci când lucrurile nu merg bine.Desigur ca operatorii umani pot cauza probleme daca gresesc.Exista trei componente ale sistemului care daca cedeaza pot duce la cedarea sistemului critic:

1. Sistemul hardware poate esua din cauza greselilor de rhitectura, deoarece componentele au cedat ca urmare a unor erori de fabricatie, sau pentru ca componentele au a ajuns la sfarsitul duratei de viata asteptata.
1. Software-ul de sistem poate esua din cauza greselilor sale în specificatiile tehnice, de proiectare sau punere în aplicare.

2. Operatorii umane ai sistemului pot sa nu il folosesca corect s. Hardware-ul si software-ul au devenit mai fiabile, esecuri în exploatare sunt acum probabil cea mai mare cauza a unor disfunctionalitati ale sistemului.

Aceste esecuri pot fi corelate. O componentă hardware care cedeaza poate insemna ca operatorilor de sistem sa faca fata cu o situatie neasteptata si cu volumul de muncă suplimentar.

Acest lucru ii pune sub stres iar persoanelor aflate sub stres fac adesea greseli care adesea poate provoca software-ul să cedeze, ceea ce înseamnă mai mult de lucru pentru operatori, chiar mai mult stres, si asa mai departe.

Ca urmare, este deosebit de important ca designerii de sisteme critice sa aibe o viziune de perspectiva, mai degrabă decât se concentreze asupra unui singur aspect al sistemului.

Daca procesele hardware, software si operationale sunt concepute separat, fara a lua potentialele deficiente ale altor parti ale sistemului in considerare, atunci este mult mai probabil ca erorile vor avea loc la interfetele intre diferitele parti ale sistemului.

1. Exemplu de implementare al unui sistem critic

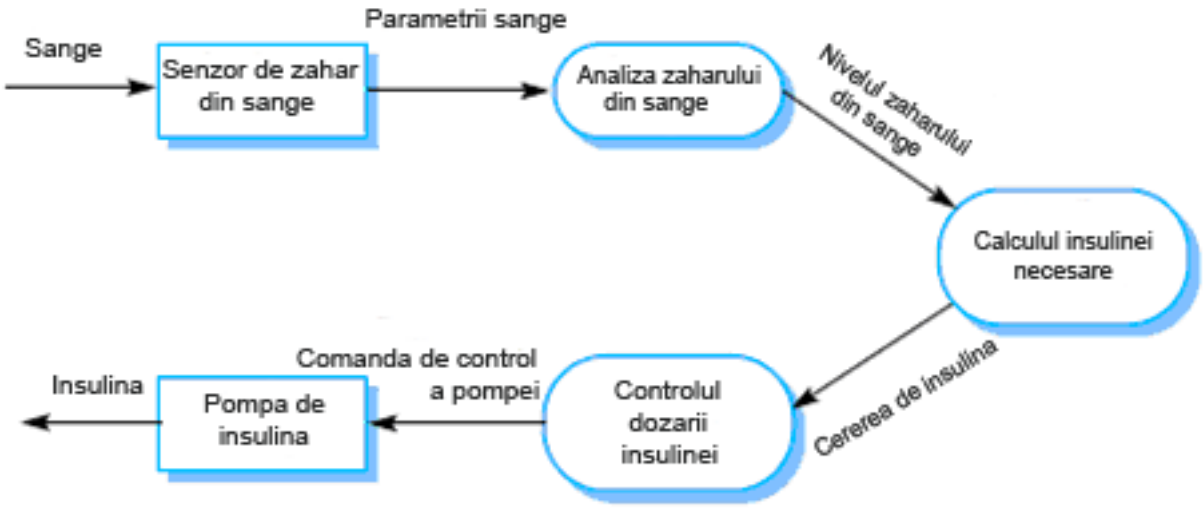
Există mai multe tipuri de sisteme critice software, variind de la sisteme de control pentru dispozitive și utilaje pentru sistemele de informare și comert electronic. Acestea ar putea fi excelente studii de caz pentru o carte de inginerie software, deoarece sunt folosite tehnici avansate de inginerie software în dezvoltarea lor. Cu toate acestea, întelegerea acestor sisteme poate fi foarte dificila, deoarece este nevoie de întelegerea caracteristicilor si constrangerilor domeniului de aplicare în care acestea isi desfasoara activitatea.

Exemplul urmator este un sistem critic folosit in domeniul medical care simulează functionarea pancreasului .Am ales acest lucru deoarece este clar de ce siguranta și fiabilitatea sunt atat de importante pentru acest tip de sistem.Sistemul ales este destinat pentru a ajutorul oamenii care sufera de diabet zaharat.

Diabetul este o afecțiune relativ comuna in care pancreasul uman nu mai produce un hormon numit insulina. Insulina are rolul de a metabolizează glucoza în sânge.Tratamentul conventional al diabetului implică injectii regulate de insulina modificate genetic. Diabeticii isi masura nivelul de zahar in sange folosind un dispozitiv extern si apoi isi calculează doza de insulina pe care acestia ar trebui sa o injecteze.

Problema cu acest tratament este faptul ca nivelul de insulina din sânge nu depinde numai de nivelul de glucoză din sange, este o functie de momentul in care a fost luată injectia de insulinei. Acest lucru poate duce la niveluri foarte scăzute de glucoza in sânge (dacă există prea multa insulina) sau foarte ridicat de zahar din sânge (în cazul în care nu există suficienta insulina). Minima de zahar din sânge este, pe termen scurt, o stare mai grava, deoarece poate duce la o functiune defectuoasa a creierului si, in cele din urma, pierderea cunostintei respectiv decesul pacientului. Pe termen lung continutul ridicat al nivelului de zahar din sange poate duce la leziuni oculare, leziuni renale, si probleme cu inima.

Avansul tehnologic actual in dezvoltarea de senzorii miniaturali insemna ca acum este posibil sa se dezvolte sisteme complect automate de administrare a insulinei. Aceste sisteme de monitorizare a nivelul glucozei in sange si livreaza o doza corespunzatoare de insulina atunci când este necesar. Sistemele de administrare a insulinei ca aceasta exista deja pentru tratamentul pacientilor in spital. În viitorul apropiat, ar putea fi posibil pentru diabetici sa aibe astfel de sisteme atasate permanent de corpul lor.



Există două cerințe de nivel înalt în functionarea sistemului de pompa cu insulină:

1. Sistemul trebuie să fie disponibil pentru a livra insulina atunci când este necesar.
2. Sistemul functionează fiabil livrezea cantitatea corecta de insulina pentru a contracara nivelul actual de zahar din sange.

Defectarea sistemului ar putea, in principiu, sa provoace doze excesive de insulina si acest lucru ar putea ameninta viata utilizatorului. Este deosebit de important ca supradoze de insulina, sa nu apară.

2. Fiabilitatea sistemului

Toata lumea este familarizata cu problema stricarii sistemului informatic. Aparent fara nici un motiv sistemele informatice se strica uneori si nu reusesc să livreze servicii care au fost programate. Programele care rulează pe aceste calculatoare poate să nu functioneze conform asteptarilor

si ocazional, pote duce la coruperea datelor, care sunt gestionate de către acestea.In timp am invatat sa traim cu aceste esecuri si putini dintre noi avem complecta incredere in computerele personale pe care le utilizeazam în mod normal.

Fiabilitate a unui sistem informatic este o proprietate a sistemului, care echivaleaza

cu incredere in el. Incredere inseamnă in esenta, gradul de încredere al utilizatorilor

ca sistemul va functiona cat ei se asteapta si că sistemul nu va claca in conditii normale de utilizare. Această proprietate nu poate fi exprimat numeric, dar vom folosi termeni relativi, cum ar fi "nu de incredere", "foarte de incredere" si "ultra-incredere" pentru a reflecta gradele de incredere pe care le-ar putea avea un sistem.

Credibilitatea si utilitatea nu sunt, desigur, acelasi lucru.Cu toate acestea, pentru a reflecta lipsa de incredere in sistemul utilizatorii au dendinta de a salva proiectele si sa pastreze mai multe copii de rezervă ale acestora. Am compensa lipsa de fiabilitatea a sistemului, prin actiuni care limitează prejudiciul care ar putea fi cauzat, daca sistemul ar claca.

In afara de aceste patru dimensiuni principale, si alte proprietati de sistem poate fi, de asemenea, luate în considerare sub titlul de fiabilitate:

1. Gradul de reparare esecuri de sistem sunt inevitabile, dar intreruperea cauzată de stricare pote fi minimizata daca sistemul ar fi reparat rapid. Pentru ca sa se întample asta, trebuie sa fie posibila diagnosticarea problemei, accesare componentei care a cedat si sa se faca modificarile pentru a remedia această componenta. Gradul de reparare in software-ul este imbunatatit atunci cand organizatia care utilizeaza sistemul are acces la codul sursa si are aptitudinile necesare pentru a face modificarile. Din pacate, acest lucru este din ce in ce mai putin frecvent pe masură ce avansam in directia de dezvoltare a sistemului folosind a terta parte.
2. Mentenanta. Este important pentru a mentine utilitatea unui sistem prin schimbare la noile componente aparute pe piata pentru a se adapta acestor noile cerințe. Software-ul de intretinere este un software care poate fi adaptat din punct de vedere economic pentru a face fata noilor cerinte si in cazul in care exista o probabilitate mica ca ar face schimbări vor introduce erori noi in sistem.
3. Supraviețuirea. Un atribut foarte important pentru sistemele bazate pe Internet este acela de a supravietui, care este strans legat de securitate si disponibilitate .Supravietuire este capacitatea unui sistem de a continua sa ofere servicii în timp ce acesta este sub atac si, eventual, in timp ce o parte a sistemului este dezactivata. Lucrarile la acest atribut se concentreaza pe identificarea componentelor cheie ale sistemului si asigurarea ca ele pot oferi un serviciu minim. Trei strategii sunt folosite pentru a imbunătati supravietuirea si anume, rezistenta la atac, recunoasterea atacuui si de recuperare de la prejudiciul cauzat de un atac .
4. Toleranță la erori Aceasta proprietate poate fi considerata ca parte a utilizabilitatii si reflecta masura in care sistemul a fost proiectat astfel încat eroarea de intrare de utilizator sunt evitate și tolerate. Atunci cand apar erori de inserare cauzate de utilizator,Sistemul ar trebui, pe cat posibil sa detecteze aceste erori si, fie capabil sa le repare in mod automat sau sa solicite utilizatorului sa re-introduca datele.

Deoarece disponibilitatea, fiabilitatea, siguranta si securitatea sunt proprietatile fundamentale ale fiabilitatii ,desigur, aceste proprietati nu sunt toate aplicabile tuturor sistemelor.Pentru sistemul de pompa de insulina, introdus în sectiunea 1.1, cele mai importante proprietati

sunt disponibilitate (trebuie să functioneze atunci cand este necesar), fiabilitatea (acesta trebuie sa livreze doza corectă de insulina) si de siguranta (niciodata nu trebuie să livreze o doza periculoasa de insulina).Din vederea securitati, este mai putin probabil să fie o problema, deoarece pompa nu va mentine informatii confidentiale și nu este în retea astfel încât nu poate fi atacat cu rea intentie.

Designerii trebuie sa faca, de obicei, un compromis intre performanta sistemul si fiabilitate. In general, un nivel ridicat de fiabilitate poate fi realizată numai in detrimentul performantei sistemului. Software-ul de încredere include suplimentar, de multe ori structuri de cod redundante, pentru a efectua controlul

necesar pentru starile sistemului exceptionale si sa recupereze de la defectele de sistem. Acest lucru reduce performanța sistemului si mareste valoarea pretului de dezvoltare.

Din cauza costurilor suplimentare de proiectare, implementare si de validare, in cresterea

fiabilitatii unui sistem poate creste in mod semnificativ costurile de dezvoltare. In special,

Costurile de validare sunt ridicate pentru sistemele critice. Precum si validarea ca sistemul satisface cerintele sale, procesul de validare poate poate fi extern cum ar fi Autoritatea Federală a Aviatiei,care va acredita ca sistemul este de incredere.

Natura relatiei intre cost / fiabilitate,implica ca nu este posibil sa se demonstreze ca

un sistem este de 100% de incredere, deoarece costurile de asigurare fiabilitate ar atunci

sa fie la infinit.

II Dezvoltarea sistemelor software critice

[13]

Exista o tendinta pentru companii sa-si transforme sistemele software si practicile de dezvoltare a produselor intr-o forma mai incrementala folosind un ciclu de viata special de dezvoltare software agil si modele de manangement de proiect. Procesele de dezvoltarea incrementala au fost folosite si anterior, dar procesele agile adauga o noua planificare de proiect cat si principii de management si executie. Principala valoare a proceselor agile vine din cresteri controlate si lansari, pe care le produc mai des decat modelele anterioare. Lansarile controlate ar trebui sa fie in special folositoare in obtinerea feedback-ului de la clienti si pentru riscuri managerial de proiect.

Motivatii-cheie pentru dezvoltarea iterativa:

- Dezvoltarea iterativa are un risc redus;
- Diminuarea riscului timpuriu;
- Gazduieste si provoaca schimbari timpurii;
- Complexitate gestionabila;
- Incredere si satisfactie de la succesele repetate, anterioare;
- Productie partiala timpurie;
- Urmarire relevanta a procesului; previzibilitate mai buna;
- Calitate mai inalta;mai putine defecte;
- Potriviri mai bune ale produsului final cu dorintele clientilor;
- Imbunatatire timpurie si regulata a procesului;
- Comunicare si angajament necesar.

Acestea sunt toate beneficiile pe care o companie le cauta cand incepe sa foloseasca metodele agile. Procesele agile moderne promit si mai multe beneficii in comparatie cu modelele iterative/incrementale, in special urmatoarele:

- Timp mai redus pentru primele lansari si mai frecvente;
- Cantitate mai redusa de documentatie a procesului si proiectului dar o mai buna comunicatie si astfel un proces mai lin;
- Participare crescuta a clientului.

Cele mai importante beneficii depind de tipul dezvoltarii: in principal, daca este vorba de o dezvoltare orientata pe client a sistemului adaptat sau o dezvoltare orientata maselor. Uneori abordarile pot fi combinate daca produsele sunt dezvoltate pentru o mica clientela cheie si oferta este orientata catre pietele de masa. Astfel de abordari si nevoi sunt totusi diferite.

In dezvoltarea sistemelor adaptate, nevoile esentiale ale clientului, care poate beneficia de sistemul agil, sunt lansarile timpurii si intelegerea sistemului prin folosirea sa, lansarile regulate intr-un ritm sensibil astfel incat noul sistem poate fi invatat si toate adaptarile necesare pot fi facute la timp, si de a face modificari ale planului in timpul procesului intr-un mod flexibil. In dezvoltarea produselor orientate catre pietele de masa, nevoile pot fi bazate mai mult pe dorinta producatorului de a controla riscurile si de a raspunde rapid competitiei si nevoilor emergente ale pietei.

Scopul atingerii acestor tinte influenteaza modul cum companiile abordeaza procesul de dezvoltare. Unele pot porni procese agile cu o idee destul de vaga despre concept, unele il pot vedea in mod predominant ca pe un mod de a face produsul software mult mai controlabil, si altele pot urmari pur si simplu un rand de solutii productive si noi lansari pentru clienti. Orientarea si disponibilitatea lansarii sunt, de fapt, vazute ca un element cheie al abordarilor agile.

Organizatiile publice si private au inlocuit modelele lor in cascada sau chiar modelele cu incrementare cu modele variate de proiecte agile si practicile lor corespunzatoare pentru anumite perioade. Acest lucru s-a intamplat in toate tipurile de domenii de dezvoltare: sisteme simple cat si complexe. Sistemele agile au fost folosite cel mai mult in proiecte mici; aplicatiile sale in proiectele mari sunt inca in faza de cercetare. O companie numita VersionOne a publicat anual studii despre adaptarea dezvoltarilor agil, rapoartele continand o multime de informatii detaliate. Sunt, totusi, unele contexte de dezvoltare in care este o intelegere insuficienta a modului in care procesele agile pot fi folosite pentru a aduce beneficii si a nu pune in pericol calitatea operatiei si a produselor, afacerilor produselor sau operatiilor clientilor.

Dezvoltarea sistemelor critice de siguranta este o asemenea arie. Ar trebui notat faptul ca aceasta nu este o zona eterogena, dar este formata din multe culturi de dezvoltare definite de:

- Tipul produsului si sistemului – de la dispozitive medicare la masinarii si centrale nucleare.

- Rolul software-ului in sistem – este in principal un sistem bazat pe software sau este un software inca intr-un rol limitat, iar produsul este perceput ca fiind, de exemplu, un dispozitiv mecanic.
- Dimensiunile si scopul sistemului – dispozitivele mici, personale este clar ca necesita o abordare total diferita fata de sistemele de la nivelul unei fabrici.
- Nivelul de risc al sistemului – masinariile din fabrici au un nivel de risc foarte diferit fata de cele din centralele nucleare.

In acest fel, sunt multe variabile si putine raspunsuri generice, si noi nu trebuie sa copiem practicile orbeste dintr-un alt domeniu, ci ar trebui sa intelegem contextul si sa vedem posibilitatile pe care abortarile agile le pot aduce si modul in care acestea trebuie aplicate.

1. Anatomia dezvoltarii agile

Dezvoltarile sistemelor software agile sunt formate din urmatoarele elemente:

- Principii
- Model de proiect
- Ciclul de viata al dezvoltarii software
- Metodele, tehnicile si practicile ingineresti software.

Principiile constau din valori, principii de dezvoltare, politici si modele de gandire ale persoanelor fizice si grupuri profesionale si parti interesate. Modelul de proiect este conceptul planificarii proiectului si executiei. O mare parte a executiei proiectului consta in dezvoltarea software folosind un ciclu de viata specific acestei dezvoltari, care la randul sau, foloseste diferite metode de inginerie software, tehnici si practici, dintre care unele pot fi dezvoltate special pentru dezvoltarea agila, iar altele vor avea o origine mai generica.

Pana acum, procesele de baza agile sunt intelese ca si-ar fi ‘pierdut’ unele practici de inginerie software, si trebuie sa fim atenti sa analizam faptul ca acestea nu vor ramane pierdute in contextul siguranta-critica. Practici:

- Arhitectura
- Manipularea dependentelor dintre cerinte
- Fundamente pentru utilizare
- Documentare
- Bunul simt, gandirea si ingrijirea.

Au exitat unele abordari metodologice pentru a aduce unele dintre elementele care lipsesc, la locul lor, dar in contextul siguranta-critica, elementele analitice au radacini adanci, care sunt susceptibile de a fi in masura sa abordeze deficientele de agilitate cand sunt aplicate in mod corespunzator. In toate cazurile, procesele agile de baza gasite in manuale trebuie sa fie completate cu orice practici care sunt necesare in anumite situatii sau contexte de dezvoltare. Acesta este unul din lucrurile care se doreste – pentru a evalua modul in care procesele comune

agile trebuie sa fie modificate pentru a fi corespunzatoare in siguranta dezvoltarii critice. Un proces de dezvoltare agil nu este defapt chiar ‘agil in totalitate’. Asa-numitele procese hibride combina practicile agile cu modurile traditionale de a face lucrurile.

2.Orientari privind schimbarea procesului pentru a-l face mai agil

Cerinte preliminare

Dezvoltarea agila nu este un scop in sine, ci un mijloc de a satisface unele afaceri sau alte obiective.

Pentru orice schimbare intr-o organizatie, ar trebui sa fie un sens clar, comun de urgenta, pentru a schimba procesele precum si un sentiment ca modul actual de actiune nu mai este suficient. Fara acest lucru, probabilitatea de succes nu este suficient de ridicata – trebuie sa existe dorinte de schimbare. O cerinta pentru acest lucru poate fi formularea unui caz de afaceri pentru procesul de dezvoltare: ce beneficii ar aduce o alta lansare practica. Toata lumea ar trebui sa simta ca exista probleme comune pe care modelul de dezvoltare agil ar trebuie sa le rezolve.

Procesul de intelegere solida si viziune

Ar trebui sa existe o intelegere a software-ului si a proceselor de dezvoltare a produsului – agil si altele – care ar trebui sa ajute la intelegerea unei intrebari de baza: tintim cu adevarat pentru agilitate maxima sau pentru o abordare echilibrate? Acest lucru nu poate fi repetat de prea multe ori: Scopul nu este de a fi agil, ci obtinerea celor mai bune rezultate prin utilizarea unor procese care se potrivesc cel mai bine scopurilor unui mediu in particular.

Profesionalism in actiune

Abordarea traditionala la schimbarea unui proces este ca nici o piesa nu este la locul ei si ca o incercare de a schimba un proces poate duce la haos si esec. Inainte de transformare, organizatia trebuie sa fie in masura sa-si indeplineasca, de exemplu, toate cerintele IEC 61508 sau alte standarde aplicabile fara probleme. Acest lucru ofera o baza pentru dezvoltarea de proces si ofera posibilitatea de vizualizare a problemelor in mod clar.

Calitatea activitatilor curente ar trebui sa fie ridicata si executia procesului curent, profesionala. Daca nu este cazut, modificarea nu ar trebui incercata.

Experti si colaborare

Expertiza este necesara pentru a ghida restul de oameni in procesul de transformare. Sunt necesare persoane pentru proces sau consultanti care sa conduca transformarea. Aici ar trebui notat ca acesti consultanti pentru sisteme agile nu inteleg neaparat toate cerintele procesului de dezvoltare critica si principiile de conducere de siguranta si gestionare a riscurilor. Chiar si expertii de clasa mondiala pot avea limitari in intelegerea testarii, verificarii si validarii. Toate zonele necesare de expertiza trebuie sa fie prezentate in proces.

3.Zece notiuni generale de orientare pentru dezvoltarea unui proces

Urmatoarele sunt considerate ca principii directoare importante pentru transformarea modelelor de proces traditional in altele mai agile:

- Respectati-va abilitatile de inginerie si propriile experiente in procesul de dezvoltare
- Nu urmati moda. Cautati practici dovedite si analizati daca sunt adevate pentru mediul si cultura dumneavoastra.
- Intelegeti ca nu totul trebuie sa fie agil, si ca 'mai agil' nu inseamna 'mai bun'.
- Amintiti-va cu cat un sistem si dezvoltarea lui e mai critica, cu atat este necesar un mai mare control pentru proces, lucru care va insemna, de obicei, mai putina agilitate.
- Nici o paradigma unica nu este suficienta. Fiind doar agil nu este suficient. Unele zone pot solicita agilitate mai mica decat altele, pentru nevoile lor specifice si pentru echilibru.
- O mare parte din activitatea actuala a companiilor este tacita si nu e documentata. Astfel, ati putea fi orb la factorii de succes actuali. Consultatiile externe sunt deseori necesare pentru a vedea ce este esential in activitatile dumneavoastra.
- Dezvoltarea sigura si critica este despre cum se gestioneaza riscurile. In acelasi mod, aveti nevoie sa gestionati riscurile la schimbari in procesele de creare.
- 'Gasiti stilul cel mai potrivit pentru dumneavoastra' este mesajul in toate dezvoltarile. Daca te porti ca toata lumea, cum poti fi cel mai bun din bransa ta?
- Dezvoltarile de procese nu sunt doar probleme de ordin mecanic ale proceselor. Agilitatea trebuie sa se incadreze in cultura corporativa, care este un lucru greu de schimbat.
- Cooperarile si colaborarile dintre partile interesate este o problema cheie a sistemului agil. Aceeasi regula se aplica pentru procesul de dezvoltare.

4.Analiza de risc a transformarii

O analiza de risc ar trebui sa fie efectuata pentru o schimbare a procesului, in scopul de a identifica potentialele capcane. Se pleaca de la ideea ca transformarile practice de proiect sunt in mare parte o schimbare culturala. Procesele pure sunt usor de schimbat, dar in dezvoltarea software, este o chestiune ce tine de modul in care oamenii lucreaza impreuna. Elemente ale procesului software – practici, procese, tehnici si instrumente – pot fi considerate ca formand un set de instrumente cu elemente din care suntem liberi sa ne alegem.

5.Strategie pas cu pas

Elemente de baza care ofera valoare pentru orice model de proces

O strategie este de a pune in aplicare prima data cateva practici permisive si alte elemente de procese care aduc beneficii oricarui proces de dezvoltare. Acestea includ:

- Unitati de testare mai amanuntita si asigurarea codului de calitate, pentru a nu exista o baza solida pentru schimbarea produsului

- Integrare continua, astfel ca exista intotdeauna un produs de lucru la indemana pentru a evalua modul in care se comporta sa se invete de la el.
- Automatizare de testare. Automatizarea cat mai multor sarcini posibile care sunt necesare pentru o eficienta validare si verificare si totodata sa sprijine cat mai multe paradigme de testare pentru diversitate si eficienta a testelor.
- Automatizarea proceselor agile – unelte automate de raportare si sisteme de informatii care pot fi adaptate cu usurinta la situatii schimbatoare intr-un proiect.
- Urmarire automatizata. Cand o modificare la o cerinta, de proiectare sau implementare este planificata, sau a fost facuta, sunt necesare instrumente pentru a vedea imediat ce afecteaza si ce alte nevoi sunt adresate. Sistemele informatice bune furnizeaza aceasta functionalitate.

Claritatea regulilor

Sistemul agil impune regul clare pentru succes – libertatea este intotdeauna cea mai buna in cazul constrangerilor. Este nevoie de urmatoarele informatii:

- Cine detine produsele, afacerea produselor si siguranta?
- Cine detine procesul de dezvoltare?
- Care sunt principiile de dezvoltare a produsului – este condusa de client sau de inovatii? Ambii pot solicita diferite procese si participare.
- Cum putem cantari opiniile partilor interesate? Este dezvoltarea produsului un proces democratic sau e condusa de o anumita parte?
- Cat de deschisi vrem sa fim cu adevarat fata de clienti?
- Cat de mult dorim sa se angajeze sucontractanti?

Comunicare si colaborare

Se doreste sa existe comunicare pentru ca oamenii sa invete constant si trebuie sa fie creat un mediu propice ca sa se intample asta.

- Este necesara mai multa auto-reflexie in acest proces. Toti participantii trebuie sa se uite frecvent in produs si in proces pentru a putea vedea ce trebuie imbunatatit. Asta solicita lectii frecvente/ retrospectiva intalnita in proces.
- Team-bulding si o echipa de conducere mai buna. Echipele trebuie sa fie capabile si sa invete sa lucreze ca o echipa; pentru a invata sa conduci o echipa sa colaboreze necesita timp.
- Imbunatatirea instrumentelor de comunicatii. Orice lucru care ajuta oamenii sa comunice mai bine furnizeaza un mediu mai linistit pentru a indeplini sarcinile.
- Imbunatatirea comunicarii dintre site-urile distribuite (inclusiv cele de subcontractanti). Nici o parte nu trebuie sa aiba un rol secundar in comunicare.

Asigurarea eficientei in consumul de timp

Validarea si, in special, certificarea externa, poate fi consumatoare de timp si, astfel, dificil de implementat in procesul agil. Prin urmare, procesele trebuie sa fie dezvoltate in asa fel incat validarea si certificarea sa fie cat mai eficiente posibil.

- Crearea de procese externe in afara procesului incremental principal de dezvoltare.
- Intelegerea clara a orelor la care sarcinile legate de validare trebuie sa fie efectuate.
- A face o distinctie foarte clara intre nivelurile critice pentru siguranta si alte cerinte si nivelele SIL, astfel incat elementele care necesita validare sau revalidare pot fi identificate exact.
- Sisteme eficiente de informare, astfel incat orice documente, rutine sau gasirea si inregistrarea verificarii dezvoltarii sa nu ia timp.
- Aranjamente flexibile pentru validare. In cazul in care se utilizeaza validatoare externe, situatia poate fi reevaluata – Ar putea validarea sa fie efectuata pe plan intern?

Asigurarea controlului

Se creeaza caracteristicile proceselor care ajuta la controlul si aducerea la lumina a posibilelor probleme cat mai curand posibile.

- Divizarea cerintelor in elemente mai mici, cu scopul de a ajuta la planificare si estimare.
- Imbunatatirea reuniunilor – a le face mai frecvente, pentru a participa toate persoanele; dezvoltarea culturii meetingurilor.
- Creare tablouri de bord si sisteme de informare pentru ca toata lumea sa vizualizeze starea proiectului.

Impartirea procesului

Orice proces de dezvoltare poate fi impartit in mai multe iteratii independente. Factorii esential pentru acestea:

- Un lant de evenimente, de la planificare pana la evaluarea a ceea ce s-a realizat.
- Urmarind atingerea fiecarei incrementari.
- Planificare adevarata la inceputul unei incrementari.
- Utilizarea practicilor reale de dezvoltare a produsului in definirea cerintelor. Intrarea ar trebui sa fie ceva care ofera valoare: caz nou de utilizare, povestea utilizatorului sau ceva similar si nu o specificatie tehnica sau o cerere de schimbare. Cererile de schimbare apartin domeniului de intretinere si nu domeniului unui nou ciclu de viata al dezvoltarii de produs.

Adaugarea practicilor care aduc valoare

Momentan sunt si vor mai fi multe practici bune in cultura agila. Ideea nu este de a le pune in aplicare pe toate, ci doar pe cele care aduc cu adevarat valoare. Orice dezvoltare software matura nu ar trebui sa limiteze punctul sau de vedere la o singura paradigma.

Cultura actuala nu a aparut pur si simplu: este un rezultat al evolutiei procesului, de combinare a ideilor si practicii intr-un nou ansamblu. In acelasi mod, procesele din organizatii ar trebui sa evolueze prin modificari care ofera imbunatatiri de calitate sau de performanta proiectelor.

Prin urmare, pentru noile idei de dezvoltare de procese, multe zone relevante ar trebui aduse in vizor:

- Dezvoltarea noilor proiecte
- Stiinta software
- Analiza sigurantei , siguranta si gestionarea riscului
- Gestionarea calitatii
- Testarea
- Gestionarea informatiei
- Comunicatia
- Controlul procesului

Imbunatatirea continua

Imbunatatirea continua a procesului. Se continua imbunatatirea procesului si se rezolva toate problemele cate una la un moment dat.

Procesele agile care sunt aplicate trebuie sa fie dezvoltate in continuare, la fel ca orice procese. Procesele sunt optime doar intr-un context dat. Imbunatatirea continua ar trebui sa aiba un rol important in orice organizatie.

III.Validarea şi verificarea sistemelor critice

1.Introducere

În timpul şi după procesul de implementare, programul care este dezvoltat trebuie să fie verificat pentru a ne asigura ca îşi îndeplineşte specificațiile şi că oferă funcționalitatea aşteptată de către utilizatori. Cu ajutorul celor două activități se verifică dacă software-ul satisface specificațiile sale, în timpul fiecărei faze a ciclului său de dezvoltare.Validarea şi verificarea sau V&V porneşte de la reluarea cerințelor şi continuă cu trecerea în revistă a design-ului şi inspectarea codului, oprindu-se la testarea produsului.

V&V nu înseamnă acelaşi lucru, deşi, în mod eronat sunt confundate adesea.
Diferența între ele este exprimată succint prin:

- *Verificarea*: sunt produsele in conformitate cu cerintele lor specificate?
- *Validarea* : sunt produsele in conformitate cu cerintele clientului (utilizatorilor)?[1]

Verificarea înseamnă actul de a stabili şi documenta faptul că articolele, procesele, serviciile,documentele,etc. sunt în conformitate cu cerințele specificate.

Validarea reprezintă procesul de a evalua un sistem sau o componentă în timpul sau la sfarşitul procesului de dezvoltare pentru a determina dacă satisface cerințele specificate.

Diferența dintre verificare și validare este importantă numai pentru teoreticieni. În practică, verificarea și validarea se referă la toate activitățile care asigură că software-ul va funcționa conform cerințelor.

Validarea este activitatea de acceptare a produsului final de către beneficiar.Acesta e mai puțin interesat de metodele folosite în realizarea programului. El urmăreşte dacă programul face ceea ce el aşteaptă să facă, dacă rezultatele sunt obținute sub forma dorită şi dacă sunt complete. Testarea făcută de către beneficiar nu se bazează pe date de test ci pe date reale. Se verifică funcționarea programului în condiții reale. În plus beneficiarul va verifica dacă există documentație pentru programul ce-l recepționează, în primul rând cea de utilizare, dar un beneficiar avizat nu uită să ceară documentația de realizare.

Verificarea este o activitate de durată, care începe din prima fază a realizării programului şi se încheie la predarea lui.
Ea constă din:

- analiza specificațiilor;
- inspectarea proiectării, codificării şi documentării;
- execuție simbolică a algoritmilor folosiți;
- testarea produsului;
- citirea documentației.[3]

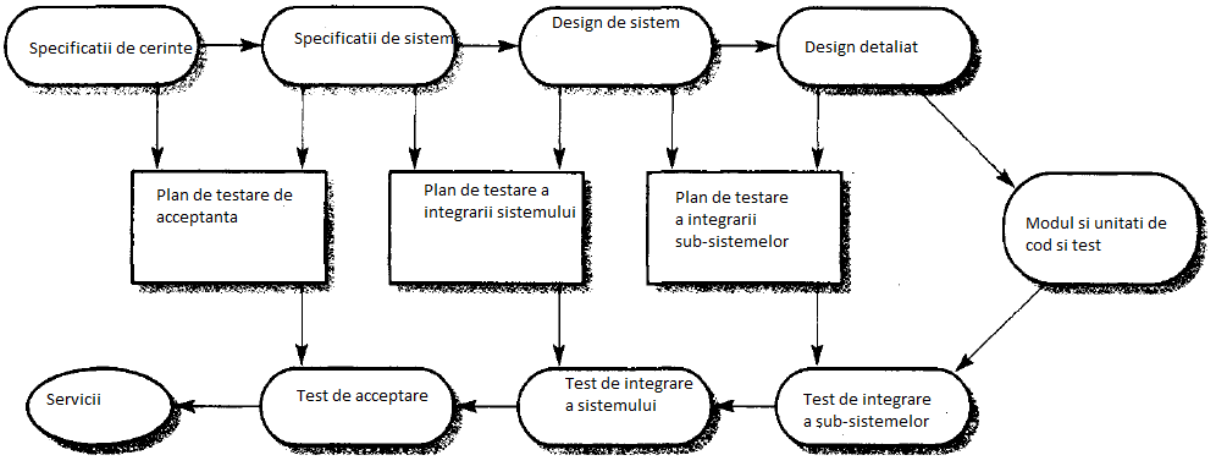
Scopurile verificării sunt acelea de a demonstra că programul este corect (dacă metoda şi programul permite) și de a găsi erori. Detecția de erori este consistentă când erorile raportate sunt reale,complete și când s-au găsit toate erorile.Validarea, în schimb, este un proces mai general. Scopul validarii este de a asigura că sistemul software întruneşte aşteptările utilizatorului.

Scopul final al V&V este să stabilească încrederea că sistemul software se potriveşte scopului său.
Obiectivul activităților de verificare și validare este de a reduce erorile software la un nivel acceptabil. Efortul necesar poate reprezenta 30-90% din totalul resurselor proiectului, in functie de complexitatea și gradul de risc al funcționării necorespunzătoare a software-ului. Organizarea activităților de verificare și validare este inclusă in activitățile de management ale proiectului software.[1],[4]
Ca o concluzie validarea este o activitate mult mai scurtă ca timp de desfăşurare. Accentul e pus pe testare cu date reale, testarea robusteței, verificarea timpului în care se obțin rezultatele şi a formei sub care sunt obținute. Pentru beneficiar este foarte importantă nu numai corectitudinea rezultatelor ci şi forma sub care sunt ele obținute.

2.Planificarea verificării si validării

Verificarea şi validarea sunt procese costisitoare. Pentru unele sisteme, cum ar fi cele în timp

real cu constrângeri nefuncționale complexe, mai mult de jumătate din bugetul de dezvoltarea a sistemului este cheltuit pe cele două procese.O planificare îngrijită este necesară nu doar pentru a reduce costurile.



-model de dezvolate sistem software-

Planificarea verificării şi validării trebuie pornită la începutul procesului de dezvoltare.În figura de mai sus este prezentat un model de dezvolate de produs soft. El figurează că planurile de testare sunt derivate din specificațiile de sistem şi design. Acest model împarte sistemul de V&V într-un anumit număr de stagii. Fiecare stagiu este condus de teste care au fost definite să inspecteze conformitatea programului cu designul şi specificațiile produsului soft.

Din procesul de planificare V&V fac parte *decizia* privind raportul de tehnici de abordare statice şi dinamice pentru validare şi verificare, definirea de standarde şi proceduri pentru *inspecția software-ului*,trebuie și definirea *planului de testare*.

Efortul relativ depus pentru inspeţtii şi testări depinde de tipul de sistem care este dezvoltat. Ca o regulă generală, cu cât un sistem este mai critic, cu atât mai mult efort ar trebui alocat tehnicilor de verificare statică.

Planificarea testării se concentrează cu stabilirea de standarde pentru procesul de testare, nu doar cu descrierea testării de produs. Pe lângă faptul că ajută managerii să aloce resurse şi să estimeze programul de testare, planurile acestea sunt destinate şi inginerilor implicați în design şi realizarea testării. Ele ajută echipa tehnică să îşi facă o imagine generală asupra testelor de sistem.

Structura unui plan de testare software este alcătuită din următoarele etape:

- **Procesul de testare**

În această etapă se realizează o descriere a fazelor majore ale procesului de testare.

- **Urmărirea cerințelor**

Utilizatorii sunt interesați în mod deosebit ca sistemul să îşi îndeplinească cerințele şi de aceea testarea ar trebui planificată astfel încât toate cerințele să fie testate individual.

- **Componentele testare**

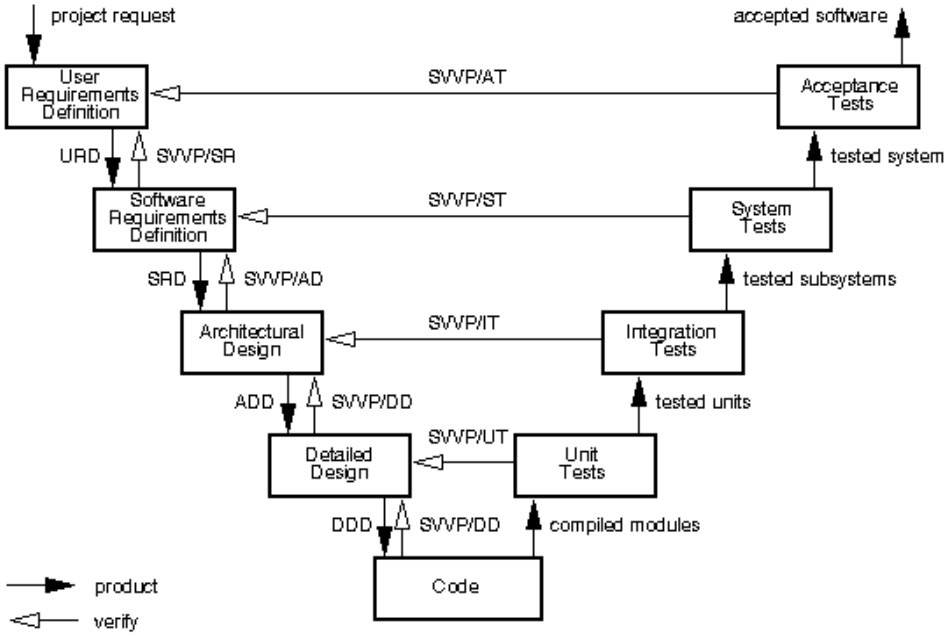
Componentele care trebuie testate trebuie specificate.

- **Programul de testare**

Etapa cuprinde programul general de testare şi alocarea resurselor.

- **Inregistrarea procedurilor de testare**

Nu este suficient ca testele sa fie



realizate,rezultatele trebuie sa fie inregistrate sistematic pentru trebuie a se face posibilă inspectarea faptului că testul a fost realizat corect.

○ **Cerințele hardware şi software**

În această secțiune trebuie să se stabilească ustensilele software utilizate şi incarcarea hardware estimată.

○ **Constrangeri(**constrangerile care afectează procesul de testare).

Planurile de testare nu sunt documente statice, ci evoluează pe parcursul procesului de dezvoltare. Ele se modifică datorită întarzierilor de la alte stagii. Dacă o parte a proiectului nu este completă, sistemul ca un intreg nu poate fi testat.[1]

3.METODE DE VALIDARE

Validarea poate fi clasificată astfel:

3.1.**Validare în perspectivă** reprezintă activitațile desfășurate înainte ca articolele noi să fie lansate pentru a se asigura caracteristicile de interes care întrunesc standarde de funcționare corecte si sigure.

3.2.**Validare retrospectivă** reprezintă un proces pentru articolele deja in uz, distribuție sau producție. Se reia evaluarea specificațiilor și așteptărilor de la produs, iar daca lipsesc date se întrerupe etapa curentă. Acest proces se realizează dacă se constată lipsa realizării unei validări de perspectivă, la schimbarea unor legislații sau standarde, la repunerea pe piață a unor produse anterior excluse.

3.3.**Validare curentă** are loc concomitent cu procesul de proiectare/dezvoltare.[6]

4.METODE DE VERIFICĂRI

Activitățile de verificare includ:

4.1.**Recenzii**;

4.2.**Verificări formale**.

- Activitățile de verificare pe parcursul Ciclului de Viață-

4.2.RECENZII(REVIEWS) de software sunt reprezentate de întâlniri pe parcursul cărora un produs este examinat de personalul care s-a ocupat de proiect, manageri, utilizatori, clienți sau alte părți interesate să comenteze sau să aprobe.

Clasificare recenziilor:

1.**Analiza codului**(recenzia codului) care examinează sistematic a codul sursă;

2.**Inspecțiile software** urmăresc o procedură strictă de depistare a defectelor.;

3.**Prezentări**(Walkthrough)-autorii organizează pentru toate parțile interesate o prezentare a proiectului, participanții având oportunitatea să afle răspunsuri, să sesizeze defecte sau neconcordanțe;

4.**Recenziile tehnice** înseamnă o echipa formată din personal calificat sa care au rolul examineze dacă produsul este adecvat pentru uzul destinat și identifică discrepanțele cu standarde și specificații.

5**.Auditurile informatice** sunt realizate de personalul din exteriorul proiectului pentru a evalua conformitatea produsului cu specificațiile, standardele și acordurile contractuale inițiale.[6]

1.Analiza codului

Analiza codului verifică dacă programul codificat implementeaza corect proiectul verificat şi nu violează cerințele de siguranță.În această fază, o buna parte din întrebări vor primi răspuns pentru prima dată. De exemplu, numărul liniilor de cod, resursa de memorie, încărcarea CPU pot fi observate şi măsurate.Pot avea loc chiar reproiectări semnificative bazate pe parametrii codului actual. Codul permite o măsurare reală a dimensiunii, complexității şi gradului de utilizarea al resurselor.

Analiza codului include :

a.Analiza logica a codului;

b.Analiza codării datelor;

c.Analiza codării interfețelor;

d.Măsurarea complexității;

e.Analiza limitelor codului;

f.Subsetul sigur al limbajelor de programare;

g.Metodele formale şi considerațiile critice relativ la siguranță.

a.Analiza logică a codului

Analiza logică a codului evaluează secvența de operațiuni reprezentate prin program. Analiza logicăma codului va detecta erorile logice din software.Această analiză este condusă pentru a realiza reconstrucția logică, reconstrucția ecuațiilor şi decodarea memoriei.

Reconstrucția logicii determină formularea diagramelor de flux pornind de la cod, şi compararea acestora cu diagramele de flux din documentele de descriere ale proiectului.

Reconstrucția ecuațiilor este îndeplinită prin compararea ecuațiilor din cod cu cele provenite din documentele proiectului.

Decodificarea memoriei identifică secvențele de instrucțiuni critice, chiar dacă acestea au fost definite ca date. Analistul va trebui să determine, pe această cale, ce instrucțiuni sunt valide, precum şi condițiile sub care execuția acestora este validă. Decodificarea memorie se va realiza pe codul final.

b.Analiza codului pentru date

Analiza codului pentru date se concentrează asupra structurii şi utilizării datelor în software-ul rezultat (codat). Obiectivul principal al acestei analize este de a verifica faptul că itemuri de date ce sunt definite şi organizate sunt corect, complet şi coerent definite.Această verificare este realizată prin compararea utilizării cu valorile tuturor itemelor de date din cod cu descrierea furnizată de materialele proiectului.

c.Analiza codului pentru interfete

Analiza codului pentru interfețe verifică compatibilitatea internă si externă a componentelor software. O componentă software este compusă dintr-un număr de segmente de cod care lucrează împreună pentru realizarea unei sarcini. Aceste segmente de cod trebuie să comunice între ele (fiecare cu fiecare), cu alte componente software sau hardware şi cu operatorul uman pentru a realiza sarcina dată. Se verifică, în acest sens, dacă parametrii care se transmit de-a lungul interfețelor sunt proprii acestora.

Fiecare dintre aceste interfețe este o sursă potențială de probleme. Analiza codului interfețelor se realizează tocmai pentru a verifica că aceste interfețe sunt corect implementate. Interfețele hardware şi ale operatorului uman vor fi incluse în “analiza proiectarii limitelor”.

d.Măsurarea complexității codului

Unul din scopurile măsurării complexității este tentativa de a o minimiza, pentru reducerea probabilității de erori în sistem.Una dintre tehnicile cele mai importante pentru reducerea complexității este *modularizarea*. Complexitatea se va putea măsura, utilizând metricele McCabe sau tehnici similare.

e.Analiza actualizărilor limitelor proiectului

Criteriile pentru analiza proiectării limitelor utilizate în faza de proiectare în detaliu, sunt aplicate într-o noua iterație şi codului final. În această fază, pot fi realizate testări reale care caracterizează comportamentul şi performanțele software, în completarea analizei propriu-zise. Se vor lua în considerare limitările fizice ale platformei hardware pentru implementarea dată.

f.**Listele de control ale inspecțiilor codului (care includ codificarea standardelor)**

Codificarea standardelor este bazată pe ghidul de stil şi subseturile sigure ale limbajelor de programare. Listele de verificare sunt dezvoltate pe parcursul inspecțiilor formale pentru a facilita inspectarea codului în vederea demonstrării conformității acestuia cu codificarea standardelor.

g.Metodele formale

În practică, se întâmplă cazuri frecvente de aplicare a metodelor formale pentru cod

deja existent.În acest caz, tehnicile de analiză pot fi utilizate pentru extragerea logicii din interiorul
codului.

Analiza codului neutilizat

Una din erorile care se regăsesc frecvent, în practicaă, este generarea de cod care este în mod firesc exclus de la execuție, adică precondițiile executării acestui cod nu vor putea fi satisfăcute niciodată. Nu exista tehnici particulare care să identifice codul neutilizat; de obicei, identificarea acestuia se produce pe parcursul altor tipuri de analize ale codului. O atenție deosebită trebuie acordată analizei logică a codului pentru a se verifica faptul că fiecare parte a codului este executată eventual, în acelaşi timp pe parcursul tuturor modurilor de operare ale sistemului.

Analiza testelor

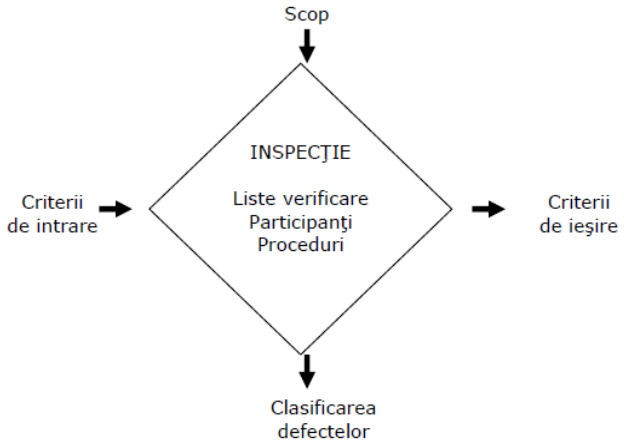
Pe parcursul fazei de testare se realizează doua tipuri de analize :

- ✓ Analiza înaintea testării pentru asigurarea validității testelor;
- ✓ Analiza rezultatelor testelor;

Pe parcursul fazei de testare, testele se împart pentru a verifica toate cerințele de siguranță pentru care testele au fost selectate în conformitate cu metodele de verificare.[3]

2.Inspecții tehnice

Scopul principal al procesului de inspecție software este de a elimina defectele dintr-un anumit produs. Produsul poate fi: specificarea cerințelor, manualul de proiectare sau module program.



-parametrii inspecției tehnice-

Principalii parametrii utilizați în inspecția tehnică sunt:
-*scopul*-descrie ce va realiza inspecția(scopul general al inspecției este verificarea este verificarea produsului software);
-*criteriile de intrare*-se referă la datele de intrare care caracterizează produsul sofware(criteriile minime sunt codul produsului și documentația produsului cu ortografia corectă)
-*listele de verificare*-acestea comțin indicații și recomandări pentru ca cei care fac inspecția să găsească mai ușor defectele produsului analizat;
-*participanții*-persoanele implicate în procesul de inspecție;
-*procedurile*-acestea au rolul să explice cum trebuie abordată inspecția;
-*clasificarea defectelor* astfel încât să fie cunoscute tipul defectului,clasa și gravitatea pentru a ajuta la procesul de identificare a defectelor;

-*criteriile de ieșire* care includ,de obicei,produsul software verficar și corectat,precum și raportul de inspecție.[5]

3. Prezentări(Walkthrough)

Acest tip de verificare este utilă în evaluarea timpurie a documentelor, a modelelor si a codului, în fazele de specificare a cerințelor software și proiectare. Obiectivul unei prezentări este de a evalua un element software specific prin identificarea defectelor și căutarea de soluții posiblie.Spre deosebire de celelate forma de revizie,prezentările au și un obiectiv secundar,acela de a educa.Toate erorile,schimbările și îmbunătățirile sugerate din timpul unei prezentări sunt înregistrate împreună cu acțiunile definite în timpul întrunirii.[3],[10]

4.Recenzii tehnice

Recenziile tehnice constau în verificarea codurilor și inlăturarea defectelor de către membrii echipei proiectului. Un defect este reprezentat de un bloc de cod care nu este implementat conform cerințelor, care nu funționează după intenția programatorului sau care nu este incorect dar poate fi perfecționat. Pe lângă faptul că inlătură bug-urile, această tehnică este de asemenea benefică pentru dezvoltatorii începători care pot învăța în această manieră noi modalități de programare.[11]
Componenta echipei de revizie difera in functie de faza din ciclul de viata in acre are loc revizia.
Astfel, din echipa pot face parte:

- Utilizatori;
- Conducători de proiect;
- Ingineri software;
- Membri ai achipei de asigurare a calității;
- Experți independenți.

5.Audituri

În cazul general auditul informatic reprezintă un ansamblu de procese informatice pentru a răspunde la o cerere precisă a clientului.

Auditarea reprezintă o activitate deosebit de importantă pentru companiile de software întrucât rezultatele procesului de auditare se constituie pentru fundamentarea deciziilor pe termen lung privind politicile de management al calității. Certificarea echipelor, sistemelor de management al calității, companiilor,persoanelor, produselor informatice, evidențiază capacitatea de a derula procese, tranzacții,capacitate care trebuie să se manifeste ori de cate ori se dezvoltă un produs, prin respectarea standardelor, prin utilizarea tehnicilor, modelelor si instrumentelor adecvate. Toate elementele converg spre obținerea de produse și servicii de calitate.

Principalele tipuri de audit informatic sunt :

1.*auditul sistemului operațional de calcul* se referă la revizia controalelor sistemelor operaționale de calcul şi rețelelor: sistem de operare, rețea, software de aplicație, baze de date;

2.*auditul instalațiilor IT* este legat de securitatea fizică, controalele mediului de lucru, sistemele de management şi echipamentele IT;

3.*auditul sistemelor aflate în dezvoltare*, care se ocupă cu controalele managementului proiectului, specificațiile, dezvoltarea, testarea, implementarea şi operarea controalelor tehnice şi procedurale;

5.*auditul procesului IT* cuprinde dezvoltarea aplicației, testarea, implementarea, operațiile, mentenanța, gestionarea incidentelor;

6.*auditul managementului schimbărilor* se ocupă cu verificarea planificării şi controlului schimbărilor la sisteme, rețele, aplicații, procese etc.[5],[8]

4.2.VERIFICĂRI FORMALE

Verificările formale sunt modalitățile prin care se aprobă sau dezaprobă corectitudinea algoritmilor care stau la baza unui sistem pe baza unor specificații sau proprietăți, folosind metode formale, matematice. Deoarece sistemul este modelat matematic, sunt posibile rezultate garantate in limitele posibilităților/presupunerilor de modelare.
Verificările formale sunt foarte folositoare la demonstrarea corectitudinii sistemelor software exprimate prin codul lor sursă. Acest tip de verificare constă în furnizarea unei dovezi formale asupra corectitudinii algoritmului pe baza modelării matematice abstracte a sistemului, corespondența dintre modelul matematic și natura sistemului fiind cunoscute din etapa conceptuală. [8]

Verificarea formală pentru software include mai multe tipuri de verificări:

1.Modelul checking;
2.Inferența logică;
3.Derivarea de program.

1.Model checking

Metoda model checking constă în explorarea sistematică si exhaustivă a modelului matematic. Acest lucru este posibil atât pentru modelele finite cât și pentru cele infinite unde seturile infinite de stări pot fi reprezenate in mod finit prin abstractizare. De obicei această metodă constă in explorararea tuturor stărilor și tranzițiilor din model, utilizând tehnici de abstractizare pe domenii, care consideră grupuri intregi de stări intr-o singura operație, reducând astfel timpul de calcul. Implementarea tehnicilor include enumerarea spațiului de stări, a spațiului

simbolic de stări, interpretarea abstractă, simularea simbolică și rafinarea abstractă.
Proprietățile care vor fi verificate sunt descrise in logica temporală, precum LTL- linear temporal logic și
CTL- computational tree logic.
Această metodă rezolvă urmatoarea problemă: dându-se modelul unui sistem, se verifică dacă acesta indeplinește o anume specificație dată. Pentru a rezolva această problemă dupa un algoritm, modelul sistemului și specificațiile acestuia vor fi formulate intr-un limbaj matematic precis: se verifică dacă o structură dată satisface o formulă logică dată.
Model checking-ul este un automat cu stări finite. Spre deosebire de verificarea prin demonstrare de teoreme,această metodă făcându-se automat pentru toate execuțiile posibile, în caz de eroare generându-se un contraexemplu.

Pentru acest model sunt incluși următorii pași:

- Se specifică proprietățile produslui soft și se traduc în limbaj C;
- Deoarece programul poate fi complex, se folosește abstracția în verificare, adică se dorește concentrarea asupra porțiunii relevante din program. Acest lucru se realizează prin *tehnica program slicing* prin care se determină fragmentul de program care afectează o anumita proprietate a programului;
- Se generează programul boolean pornind de la predicatele din specificație.
- Se analizează programul boolean adică se calculează mulțimea stărilor atinse.

- Dacă se depisteaza erori se vor genera contraexemple și noi predicate;dacă contraexemplul este realizabil, atunci eroarea este reală,altfel, se pleacă din nou din starea de eroare in programul abstract și se parcurge calea de eroare inapoi până când se găsește o inconsistență. După care se generează predicatele corespunzătoare și programul boolean;se reia verificarea. [7]

2.Inferența logică

O altă abordare este reprezentată de inferența logică care constă in folosirea unei versiuni formale a unui raționament matematic. Se folosesc de obicei teoreme precum HOL, ACL2, Isabelle, Coq. Aceste teoreme sunt aplicate parțial automat și sunt conduse de înțelgerea de către utilizator a sistemului de validat. Instrumentele mai noi precum Perfect Developer și Escher C Verifier incearcă să automatizeze complet procesul. Logica liniară și cea temporală poate de asemenea fi folosită la inferența logică și nu numai in cazul model checking-ului. [3]

3. Derivarea de program

O alta metoda este reprezentata de derivarea de program**,** când producerea de cod eficient se face plecând de la specificațiile funcționale și se urmează o serie de pași cu rolul de a conserva corectitudinea. [3]

5.CONCLUZIE

Metodele de validare și verificare sunt importante în dezvoltarea de produse soft,deoarece acestea fac posibile înțelegerea și punerea în discuție a problemei,descoperirea contradicțiilor de orice natură prin notații clare și precise.Totodată trebuie luat în calcul și satisfacerea nevoilor și îndeplinirea cerințelor clientului pentru a-i ușura munca.

De asemenea un proces al validării și verificării neatent supravegheat poate duce la un produs defectuos și chiar la realizarea unui produs neperformant cu costuri mult prea ridicate.De aceea este nevoie de documentare în derularea fiecărei etape pentru a găsi cu ușurință greșelile și pentru avea la dispoziție un set bogat de soluții.

IV Protecția Sistemelelor critice [13]

1. Siguranta in functionalitate si securitate ca elemente de siguranta generala

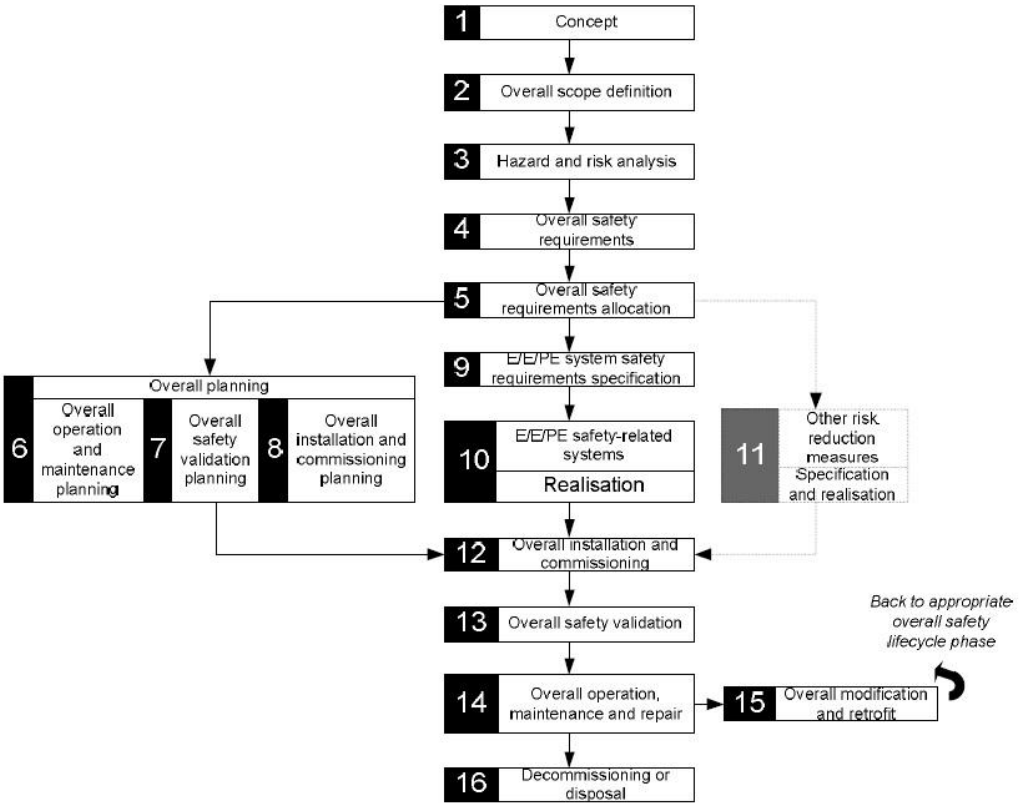
Siguranta si securitatea sunt aspecte generale ale sistemelor de control critice in orice domeniu de aplicatii. Dezvoltarea lor trebuie sa se bazeze pe riscurile si pericole percepute, indentificarea acestora fiind a unei activitati de dezvoltare esentiala. Principalul motiv pentru aceasta abilitate este a viza dezvoltarea actititatii si eforturile partilor corecte si functiilor sistemului, asadar nivelurile dorite de siguranta si securitate pot fi asigurate cu un efort minim.

Ideea de model de dezvoltare bazat pe abordări și conexe este de a utiliza modele ca artefacte de inginerie primare în cursul dezvoltării de aplicații în loc de, de exemplu, documente.Promisiunea cheie al abordărilor este abilitatea de a automatiza o parte din munca de dezvoltare cu un model de transformări care folosesc modele și specificații în producția de modele revizuite și executabile. În plus, modelarea formală sau semi-formala pot permite modelelor să fie analizate in ordine, în scopul de a descoperi probleme sau erori predispuse la aspectele de proiectare. Din punct de vedere al standardelor de siguranță, cu toate acestea, aplicatiile modelelor bazate pe tehnici au fost rare și dificile până de curând. Cu toate acestea, noua editie, a doua IEC 61508 [2010] prevede că generarea automată a software-ului poate fi un ajutor suplimentar, corect. În modelul bazat pe dezvoltarea de aplicatii critice pentru siguranță, aplicatiile ar trebui să fie astfel dezvoltate folosind modele, dar acestea ar trebui să ia în considerare, de asemenea, cerințele de la standardele de siguranță.

Siguranta

IEC 61508 [2010] este un standard internațional care acoperă aspectele care trebuie luate în considerare atunci când electricitatea / electronicele / sisteme electronice programabile (E / E / PSE), sunt utilizate pentru a efectua funcții de siguranță. Standardul este format din șapte părți, care se concentrează pe aspecte diferite ale dezvoltării de siguranță legate de sisteme incluzand,spre exemplu, cerinte generale ale software-ului .Versiunea curentă a standardului a fost publicată în 2010.IEC 61508 [2010] este de o importanță deosebită ca un standard de siguranță funcțională din mai multe motive. În primul rând, standardul a fost recent reînnoit.Prin urmare, de exemplu, lista de acțiuni recomandate și tehnici ar trebui să fie la fel de modere precum si aplicabile.În al doilea rând, unul din scopurile IEC 61508 este de a facilita dezvoltarea industriei standarde specifice, care cresc importanța sa.În cele din urmă, IEC 61508 poate fi standardul de siguranță cel mai dificil utilizat ca bază pentru dezvoltare.

Siguranta ciclului de viata introdus de IEC 61508 este prezentat în figura de mai jos. Ciclul de viață de siguranță constă in 16 faze care acoperă fazele de la conceptul și definiția domeniului de aplicare generală la dezafectarea și eliminarea sistemului. Faza de realizare a sistemelor de E / E / PE legate in siguranta (faza 10-a ciclului de viață al modelului în figură) poate fi descompusă în continuare în activități mai mici și faze. Fazele ciclului de viață ale software-ului de securitate includ un program de securitate în ceea ce privește specificațiile de securitate și de siguranță funcția de integritate, siguranța software-ului, anunturi de planificare, proiectare si dezvoltare software, integrarea PE (hardware / software), exploatarea software-ul precum si procedurile de întreținere, planificare și software-ul de securitate de validarePentru diferite faze de dezvoltare software și de design, standardul recomandă o serie de măsuri și tehnici. În plus față de tehnici, standardul definește proprietăți de integritate sistematice, a cărui realizare este promovat de măsuril recomandate și tehnici. Proprietățile de integritate sistematice pe care standardul le definește pentru faza cerințelor de siguranță includ: 1)caracter complet în ceea ce privește siguranța nevoilor abordate de către software, 2) corectitudine in ceea ce priveste siguranțea nevoilor abordate de către software, 3) libertatea de defecte intrinseci de specificații , inclusiv libertatea de ambiguitate, 4) înțelegere a cerințelor de siguranță, 5) libertatea de interferențe adverse a functiilor ce prezinta nesiguranta, cu nevoile de siguranta abordate de software și 6) capacitatea de a furniza o bază pentru verificare și validare [IEC 61508]. Aceste proprietăți au fost, de asemenea, folosite ca o tinta pentru abordarea modelarii ce urmează să fie dezvoltate în pachetul de lucru.



2.Pericolul si riscul modelarii

Analiza pericolelor și a riscurilor constituie o parte esențială a dezvoltării de sisteme critice pentru siguranță. De exemplu, în ciclul de viață general al IEC 61508 [2010], analiza riscului și pericolului constituie a treia faza a ciclului de viață, intenția fiind de a determina care pericolele și evenimente periculoase ale EUC și sistemul de control EUC.

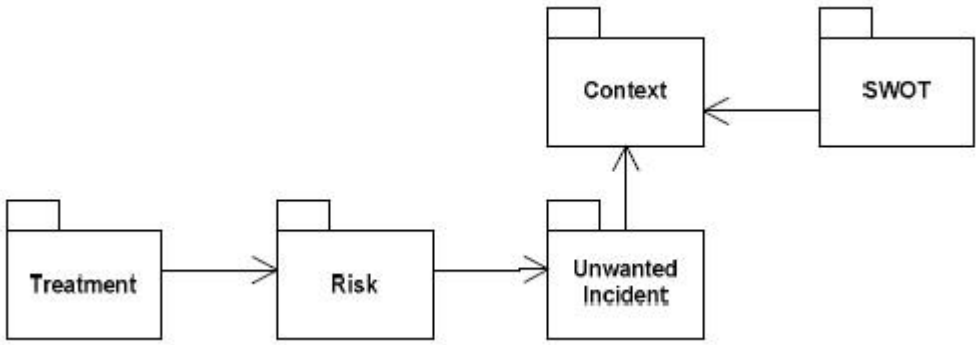
În plus față de utilizarea riscuri și pericole, ca bază pentru cerintele specificațiilor de siguranță, riscurile de modelare pot ajuta în diverse scopuri în dezvoltarea de aplicații legate de siguranță și de securitatea critica. Potrivit OMG(Object Management Group), motivația metamodelului de risc și modelare ar putea fi utilizarea practică a UML pentru a sprijini managementul riscurilor în cadrul evaluării generale și a riscurilor, în special. În evaluarea riscurilor, bazat pe modele, modele UML sunt folosite pentru trei scopuri: pentru a descrie scopul evaluării la nivel potrivit de abstractizare , pentru a facilita comunicarea și interacțiunea dintre diferite grupuri ale părți interesate implicate în evaluarea riscurilor și pentru a documenta rezultatele de evaluare a riscurilor și ipotezele de care rezultatele depind pentru a sprijini reutilizarea și întreținerea.

Mai mult, documentarea pericolelor și riscurilor asociate cu pericolele din aceleași modele cu cerințele si design-ul software-ului ar putea ajuta diverse alte scopuri.Aceste scopuri includ, cel puțin, trasabilitatea între pericolele și riscurile identificate și funcția de siguranță corespunzătoare și siguranta cerințelor de integritate , înțelegerea cerințelor și a mecanismelor de a reduce sau elimina riscurile și de a identifica funcțiile cele mai critice. Dezvoltatorii de sistem ar putea urma întotdeauna cerințele modelului periculos în cazul în care e pus la îndoială și, dezvoltatorii și implementatorii de funcții de siguranță ar putea, de exemplu, să fie instruiți să verifice întotdeauna coerența dintre artefactele înainte de a scrie o singură linie de cod. Procedând astfel, coerența ar fi verificata de mai multe ori in timpul dezvoltarii de diferite persoane, inclusiv de dezvoltatorii de cerințelor de siguranță, implementatorii (programatori) si tester-ii, pentru a numi câtiva.

Din păcate, modelarea pericolelor și riscurilor nu este în prezent acoperita demulte limbi de modelare sau profile utilizate în ingineria software. Cu toate acestea, există abordări pentru a acoperi prezentari structurate de pericole, cum ar fi profilul analizei de siguranță de Douglass [Douglass 2009] și abordarea UML Profile Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms

<u>Profiul UML pentru modelarea calității de servicii și a caracteristicilor de toleranță la erori și specificatii ale mecanismului</u>

Profilul UML pentru modelarea calității de service și caracteristicilor toleranța la erori și mecanisme este un profil UML specificat de OMG intentionand să extindă UML pentru a acoperi conceptele de calitate a serviciilor (QoS) și eroare de toleranță (FT).Extensiile sunt definite în două cadre generale:-cadru de modelare QoS și cadrul de modelare FT. Cadrul de modelare QoS descrie de vocabularul ce trebuie a fi utilizat cu tehnologii de înaltă calitate și oferă posibilitatea de a asocia cerințele și proprietățile cu elemente de model în scopul de a introduce aspecte de extra-funcționalitate în modelele UML. Cadrul pentru toleranța la erori include notații pentru un model din evaluări de risc, acordând o atenție deosebită la descrierea pericolelor, riscurilor și tratamente de risc. Acest cadru sprijină, de asemenea, descrierea erorilor de toleranta a arhitecturilor bazate pe repetiții obiect. Figura de mai jos prezintă submodelele (pachete) de cadru și dependențele lor.



3.Analiza profilului de siguranta

Potrivit Douglass [Douglass 2009], obiectivul de analiza al profilului de siguranță este de a permite captarea cerintei și analiza de siguranță cu Telelogic ® Rhapsody ® pentru modelele UML. Utilizarea UML pot ajuta la dezvoltarea într-un număr de moduri, inclusiv furnizarea de claritate a design-ului, redundanta modelarii arhitecturale și la nivel scăzut, crearea relevanta a siguranței , cu privire la cerințele de proiectare, precum și in sprijinirea analizei de siguranță. UML pot fi, de asemenea, utilizat pentru specificarea trasabilității dintre cerințe si proiectare. Vizualizările, pe de altă parte, pot fi susținute cu diagrame separate concentrându-se pe aspectele înguste ale sistemului în curs de dezvoltare cu modelul de bază, fiind inca în concordanță. Una din analizele de siguranță abordate susținută de profil este analiza defectelor unui arbore(FTA-Fault Tree Analysis).

FTA este o abordare grafica și analitica frecvent utilizata la identificarea riscurilor și a pericolelor asociate cu sistemele. În FTA, condițiile și defecte care duc la pericolele pot fi analizate logic pentru relații cauză-efect utilizând operațiuni cunoscute logice, cum ar fi SI, SAU, XOR și NOT, iar acestea pot fi, de asemenea, utilizate într-un mod cantitativ. Odată ce relațiile au fost identificate, măsurile de siguranță pot fi identificate și se adaugă la modelul de analiză. De exemplu, profilul sugerează construirea funcțiilor de siguranță, astfel încât, pentru a ajunge

la condiții periculoase, greșelile originale ar trebui să fie sfarsite cu defecte ale măsurilor de siguranță [Douglass 2009]. Cu alte cuvinte, pentru a ajunge la starea periculoasa, funcția de siguranță ar trebui, de asemenea, să eșueze.

Cerinte de modelare

Specificarea cerintelor poate fi partea cea mai critică a dezvoltării unor sisteme complexe. Specificațiile cerințelor ar trebui să fie suficient de formale pentru a îndeplini cerințele standardelor încă bazate pe concepte familiare pentru dezvoltatori.Limbile utilizate în prezent în sistemele de modelare si dezvoltare software îndeplinesc cu greu aceste proprietăți . De exemplu, numai în UML definește concepte ,,use case'' pentru care să ateste (cerințe. SysML defineste, de asemenea, un set de concepte textuale ale specificarii cerințelor, deși ele pot fi cu greu caracterizat ca formale. În domeniul de control industrial, există, de asemenea, standarde referitoare la cerințele funcționale, inclusiv IEC 62424 [2008] și IEC 61804 [2003]. Aceste standarde sunt, probabil, mult mai familiare pentru dezvoltatorii din domeniu.

Poate că cele mai bune rezultate de securitate ar putea fi realizate prin integrarea datelor de securitate în procesul de dezvoltare în loc de evaluare și corectare a riscurilor de securitate în sistemele specificate sau chiar puse în aplicare, astfel cum a fost uneori sugerat. Principalul motiv pentru aceasta este faptul că nu poate fi posibil de a porni de la riscuri, dacă sistemul nu există încă pentru a fi analizat. Punctul de plecare, prin urmare, deja diferă de la punctele de plecare ale funcțiilor de securitate (sisteme) în care sistemul conține riscurile deja existente, cel puțin în documentele de proiectare. Nu poate fi posibil să se ocupe de securitate legate de riscuri, fără a lua în considerare riscurile, în toate fazele de dezvoltare și, cel mai important, în timpul proiectării și punerea în aplicare a interfețelor de rețea. Cu toate acestea, riscurile potențiale care trebuie să fie abordate pe parcursul dezvoltării ar trebui să fie notaet în toate fazele de dezvoltare, de exemplu, realizarea de porturi (sau elemente similare în proiectare), care se ocupa de date printr-o rețea publică.

Tehnicile de dezvoltare ar trebui să includă, de asemenea, informații cu privire la securitatea datelor. În cazul sistemelor de safetyrelated, de exemplu, testarea ar putea, de asemenea, include întotdeauna de testare de securitate. În mod similar la niveluri SIL sau PL, componentele și subsistemele pot fi clasificate pe baza nivelului lor de securitate al datelor, astfel încât pentru întregul sistem pentru a atinge un anumit nivel de securitate, toate subsistemele sale ar trebui să fie la acelasi nivel sau un nivel superior. În viitor, astfel de nivele de securitate ar putea fi, de asemenea, adăugate la standardele de siguranță funcționale pentru a finaliza clasificările SIL și PL, deoarece, în general, un compromis privind securitatea poate afecta comportamentul corect al sistemului, inclusiv de siguranță.

4.Unificarea de siguranță și securitate

Scopul IEC 61508 este de a oferi o abordare unificată care este rațională și coerentă pentru toate activitățile ciclului de viață de siguranță ale sistemelor funcționale de siguranță. Standardul se referă la toate fazele ciclului de viață de la conceptelor la dezafectare. Securitatea este un activ la care standardul nu se referă, însă, cu toate că a compromite securitatea ar putea pune, de asemenea, siguranța în pericol în cazul în care, de exemplu, un vierme ar putea bloca comportamentul unui subsistem al unui sistem de siguranță.

Conform cu IEC 61508, faza ciclului de viață, care urmareste definirea conceptului și scopului fazei, este analiza de risc și pericol, al cărei scop este de a determina (toate) pericolele și evenimente periculoase, precum și secvențele care conduc la evenimente periculoase și riscurile asociate pericolelor. Poate că, o parte din pericolele legate de securitate și de risc ar putea fi specificate deja în această fază. De exemplu, riscurile legate de securitate ar putea fi inițial cautate la cerințele utilizatorilor, cum ar fi, nevoia de telecomanda. Acest lucru pune toată greutatea pe efectuarea analizei de risc iterativ, deoarece nu poate fi posibil pentru a identifica toate pericolele legate de securitate înainte de proiectarea sistemului de control EUC. Cu toate acestea nu pot fi abordări alternative pentru a unifica dezvoltarea datelor de siguranta functionala si securitate.

5.Securitatea fazelor ciclilor de viată

Cum IEC 61508 prevede, ciclul de viață de siguranță generală ar trebui să fie utilizat ca bază pentru revendicarea in conformitate cu standardul, dar un ciclu de viață diferit de siguranță global poate fi folosit pentru a dat de faptul că standardul cu condiția ca obiectivele și necesitățile fiecărei clauze standard sunt îndeplinite [1]. În consecință, pentru a răspunde preocupărilor de securitate a datelor, o faza ciclului de viață adecvat (sau faze), referitoare la dezvoltarea de aplicații sigure de date ar putea fi adăugate la modelul ciclului de viață al standardului. În scopul de a aplica tehnici de analiză de pericol și de risc pentru identificarea de date legate de securitate pericolele și riscurile, faza ar necesita, cel puțin, documente de proiectare legate atât de E / E / PE sistem cat și sistemul de control; în consecință, nu a putut fi efectuată mai devreme decât simultan cu faza de realizare a sistemului E / E / PES. Figura 10 schiteaza abordarea fără a se identifica nicio activitate sau sub-faza necesara pentru a asigura securitatea datelor.

Cu toate acestea, fazele ar putea include, de exemplu, identificarea pericolelor datelor de securitate și riscurile for, iar dacă este necesar, restructurare și modificările la design arhitectural al software-ului legat de siguranță.În orice caz, este vital să înțelegem că riscurile trebuie să fie identificate (găsit cu risc și metodele de analiză a riscurilor), indiferent dacă acestea sunt funcționale sau de securitate. Modalitățile de a elimina sau a le atenua ar putea fi parțial diferit. Verificarile de securitate ar trebui să fie, de asemenea, incluse în toate fazele ciclului de viață. Securitatea afecteaza analiza de risc (faza 3), în care întrebările cum ar fi "putem fi afectatati de un virus nou și necunoscut de Internet atunci când se utilizează web-based configurația dispozitivului?" Aceasta afectează realizarea (faza 9) atunci când utilizarea

unor tehnici criptografice pot face cele mai multe conexiuni de încredere. Aceasta afectează întreținerea (faza 14) atunci când orice schimbare în mediul de rețea va avea un efect asupra sistemului de critică. Aceasta afectează faza de eliminare (faza 16) atunci când eliberarea de dispozitive la reciclarea materialului pot expune parolele și numele de utilizatoi în afara societății, fără ștergerea sistemelor în mod corespunzător.

O altă abordare pentru a creste securitatea datelor poate fi de a folosi o abordare similară cu siguranța functionala . De exemplu, securitatea riscurilor ar putea fi cautata de la cerințele utilizatorilor, cum ar fi nevoia de telecomanda, și, eventual, manipulate cu funcții legate de securitate.(Acestea ar putea fi numit funcți de securitate în loc de funcții de siguranță.)Functiile legate de securitate ar putea fi folosite , de exemplu, pentru autentificarea utilizatorilor, de autorizare, precum și detectarea și corectarea erorilor de comunicare, pentru a numi doar câteva. Aceste funcții pot asigura cu greu realizarea securității fără alte activități. Există o linie fină între sisteme sigure și sisteme utilizabile in deplina siguranta, care sunt utile pentru publicul vizat. In cel mai bun de securitat,oamenii inteleg unde este linia, dar nu este întotdeauna ușor implementeze cadre de dezvoltare.

Adaosul de controale de securitate

În plus față de abordarea schițata, securitatea datelor ar putea fi tratata pe baza unei "categorii ușore", abordare care nu ar necesita modificări la modelul ciclului de viață. În această abordare, controalele de securitate ar putea fi adăugate practic latoate activitățile de dezvoltare și fazele ciclului de viață ale aplicațiilor software. În timpul acestor controale, ar putea fi evaluat dacă produsele (de exemplu, caietul de sarcini), din etapa de dezvoltare actuală compromite cumva securitatea. În cazul în care securitatea este compromisă, corecții ar putea fi planificate deja în timpul sesiunii și puse în aplicare înainte de a continua faza de dezvoltare următoare.

6. Securitatea produsului prin intermediul metodelor de proiectare

Cei mai multi consultanți de securitate oferă o soluție de ,,formare'' pentru a face produsele și sisteme mai sigure. Formarea și educația nu sunt suficiente totuși.Scopul lor principal este de a educa utilizatorii programatori, designeri și în problemele legate de securitatea. Singura soluție care functioneaza este de a forța programatori și designeri pentru a lua în considerare securitatea.

Instrumentele actuale de programare sunt bune pentru a ascunde conexiunile de rețea de utilizator. Sisteme industriale de automatizare și control sunt produselor software de retea cu cerințe diferite pentru conexiunea la rețea, în funcție de informațiile transferate. Pentru sistemele de siguranță, acest lucru înseamnă includerea unui factor de fiabilitate pentru toate protocoalele de rețea de cel mai jos nivel.

Conexiunile de siguranta necesita utilizarea de protocoale de siguranta, precum și diverse comunicații în timp real necesita utilizarea de protocoale și soluții care îndeplinesc capacitati

necesare in timp real. Prin urmare, este nu numai faptul că, de exemplu, un protocol de securitate este folosit, ci si faptul că protocoalele de nivel scăzut de transport sau implementările pe care aceste protocoale le utilizeaza sunt testate si au un factor de fiabilitate estimată. Problema de securitate reală, de obicei, constă în protocoalele de rețea fundamentale, care pot paraliza utilizarea unui protocol de nivel superior de siguranță.

Una dintre soluțiile pentru a forța programatori si designeri de a observa și de a aborda aceste cerințe ar fi să pună în aplicare "conexiunea de retea" tip de interfață pentru modelare comuna și instrumente de generare de cod de care UML este un exemplu. Tipul de conexiune de rețea ar avea subcategorii, în funcție de tipul de conexiune la rețea (în comun cu alt trafic, link-ul dedicat, conexiune wireless). Conexiunea fără fir ar avea regula "în cazul în care nu poate fi sigur de o conexiune securizată este utilizată si apoi forteaza criptarea pe interfata" și așa mai departe. Această abordare poate fi vizualizata ca implementand cele mai bune comunicatii practice de retea pentru conexiunile de interfață de programare a aplicațiilor utilizate în cadrul instrument de proiectare.

6.BIBLIOGRAFIE

[1].Ian Sommerville,*Software Engineering*,Ed.Addison-Wesley,2007
[2] http://bigfoot.cs.upt.ro/~dan/curs/fis/Cap7_Testare.pdf
[3].Militon Frențiu,*Verificarea şi validarea sistemelor soft*,Ed.Presa Universitară Clujeană,2010
[4]. http://en.wikipedia.org/wiki/Verification_and_validation_(software)
[5].http://www.ionivan.ro/ANUL-UNIVERSITAR%202010-2011/ZZZZ-cartea%20structuri%20date/F00036000-inspectie.pdf
[6]. http://en.wikipedia.org/wiki/Verification_and_validation
[7]. http://en.wikipedia.org/wiki/Model_checking
[8]. http://en.wikipedia.org/wiki/Formal_verification
[9]. http://alecu.ase.ro/articles/economistul_2005.pdf
[10]. http://en.wikipedia.org/wiki/Software_walkthrough
[11]. http://en.wikipedia.org/wiki/Software_review
[12]. Ian Sommerville ,*Software Engineering* ,Eighth Edition
[13]. Timo Malm, Matti Vuori, Jari Rauhamäki, Timo Vepsäläinen,Johannes Koskinen, Jari Seppälä, Heikki Virtanen, Marita Hietikko & Mika Katara, *Safety-critical software in machinery applications* , 2007