ТЕСТИРАЊЕ СОФТВЕРА

Бранко Поповић

Преглед

- Шта је тестирање?
- Тестирање у фази развоја
- Развој вођен тестовима
- Тестирање издања
- Корисничко тестирање

Шта је тестирање?

Шта је тестирање?

- Тестирањем се показује да ли програм ради оно што би требало и врши се откривање дефеката у софтверу, пре него што он буде пуштен у употребу

- Тестирање се врши тако што се програм покреће користећи унапред припремљене податке

- Посматра се резултат извршеног програма (грешке, аномалије, подаци о нефункционалним атрибутима програма)

Шта је тестирање?

- Циљеви тестирања софтвера

  1. Да ли софтвер испуњава одређене захтеве?

  2. Откривање ситуација у коме је понашање софтвера погрешно, нежељено или не одговара спецификацијама

Тестирање валидности

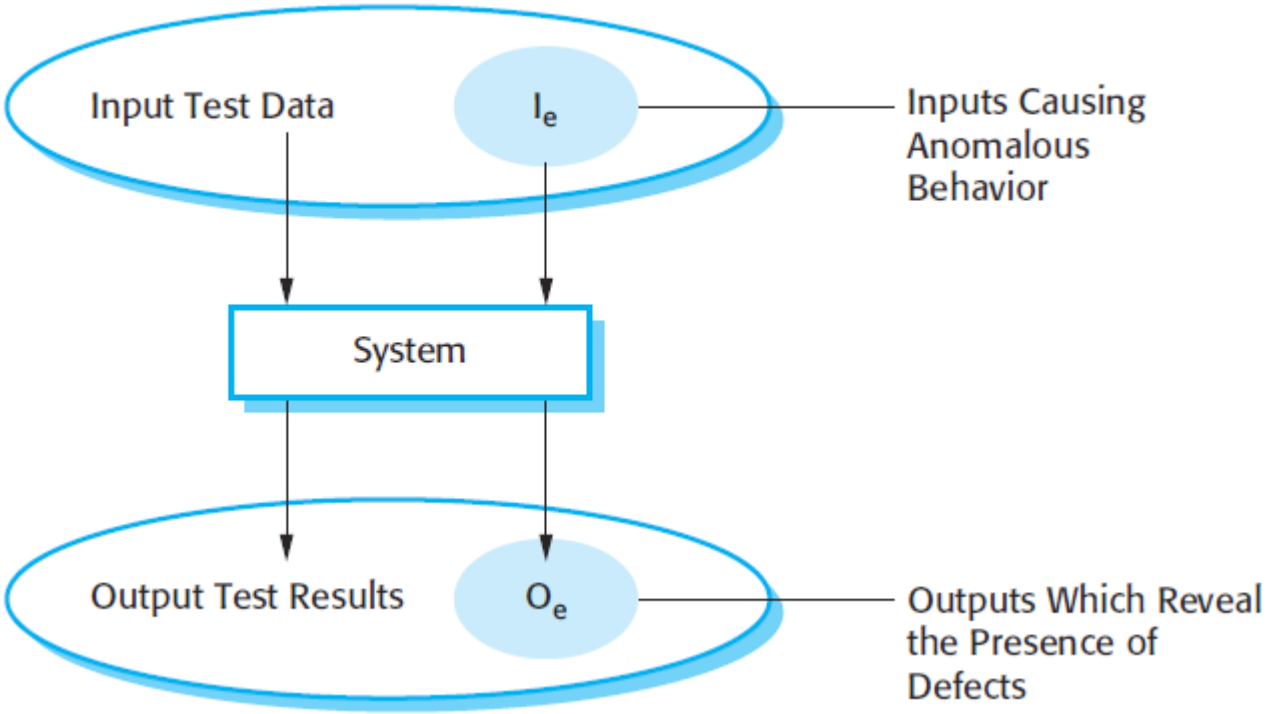
- На основу скупа тест случајева проверава се да ли програм ради оно што би требало

- За специјализовани софтвер би требало да постоји барем један тест за сваки захтев у документацији

- За генерични софтвер би требало да постоји тест за сваку карактеристику система као и за комбинације карактеристика које ће бити укључене у коначном издању

Тестирање дефеката

- Откривање софтверских дефеката
- Тест случајеви су дизајнирани да открију дефекте, па углавном не осликавају праву сврху система
- Откривају се непожељна понашања система као што су нежељене интеракције са другим системима, погрешна израчунавања, неисправност података ...



Слика 1. Улазно-излазни модел тестирања програма

Шта је тестирање?

- Тестирање не може да покаже да софтвер нема дефеката или да ради правилно у свим условима

- ,,*Тестирања може само да укаже на присуство грешака али не и на њихово одсуство* " (Е. В. Дијкстра)

- Тестирање је део доста ширег процеса верификације и валидације софтвера

Верификација и валидација

- Процеси верификације и валидације служе да покажу да ли развијени софтвер задовољава спецификације и да ли обезбеђује функционалности које очекују сами наручиоци софтвера

- Циљ верификације је да провери да ли софтвер задовољава утврђене функционалности

- Валидација је општији процес и служи да провери да ли софтвер задовољава очекивања клијената

Верификација и валидација

• Крајњи циљ верификације и валидације је да пруже уверење да софтверски систем ради оно што би требало

• Ниво уверења зависи од сврхе система, очекивања корисника и тренутног маркетиншог окружења у коме се систем налази :

1. **Сврха софтвера**
   - Што је софтвер "битнији" то мора да буде поузданији
   - На пример, ниво поверења код софтвера који се бави безбедношћу мора да буде виши него код прототипа који служи за презентовање нових идеја
2. **Очекивања корисника**
   - Када нови софтвер почне да се користи, корисници могу да буду толерантни на грешке зато што бенефити коришћења могу да надјачају цену опоравка система. У тим случајевима није неопходно толико посвећивање тестирању.
   - Како софтвер сазрева, тако се и више од њега очекује тако да је неопходно боље тестирање у наредним верзијама
3. **Маркетиншко окружење**
   - Компанија која издаје софтвер може да се одлучи на издавање софтвера пре него што се изврше сва тестирања (због различитих токова тржишта)
   - Нижа цена може да купце учини толерантнијим на грешке

Инспекција и преглед

- Статичке технике које не захтевају извршавање програма
- Проверавају системске захтеве, моделе дизајна, изворни код и предложене системе за тестирање

Инспекција

- Фокусује се на изворни код софтвера али и на било какву читљиву репрезентацију софтвера

- Постоје три предности у односу на тестирање
  1. Код тестирања једна грешка може да замаскира другу. Пошто је инспекција статички процес, до интеракције између грешака неће доћи
  2. Недовршене верзије система се могу инспектовати без додатне цене
  3. Инспекција, осим откривања дефеката, пружа и увид у доста шире атрибуте самог софтвера

- Инспекција ипак не може да замени тестирање зато што не може да пружи откривање дефеката насталих услед интеракције различитих компоненти или проблема са перформансама система

Шта је тестирање?

- Комерцијални софтверски системи обично пролазе кроз три фазе тестирања:
  1. Тестирање у току развоја. Систем се тестира током развоја због откривања багова. Програмери и дизајнери система су укључени у ову фазу
  2. Тестирање издања. Одвојени тимови за тестирање тестирају софтвер пре него што изађе пред купце.
  3. Корисничко тестирање. Корисници или потенцијални корисници тестирају систем у свом окружењу.

- У пракси се користи мешавина мануелног и аутоматског тестирања

Тестирање у фази развоја

Тестирање у фази развоја

- Укључује све активности тестирања које обавља тим који развија систем

- Софтвер обично тестира програмер који га развија

- У тиму развиаоца некада може да постоји група која ради само тестирања и која је задужена за развијање тестова и одржавање података о резултатима тестирања

- У току развоја, тестирање може бити обављено у три нивоа:
  1. Тестирање јединице. Тестирају се јединице одређеног програма као и класе објеката. Фокусује се на тестирање функционалности објеката и метода.
  2. Тестирање компоненти. Неколико јединица су интегрисане у јединствену целину. Фокусује се на тестирање интерфејса компоненти.
  3. Тестирање система. Све компоненте су интегрисане и систем се тестира у целости. Фокусује се на интеракцију компоненти.

Тестирање јединице

- Тестирање програмских компоненти као што су методи или класе објеката

- Тестови позивају ове компоненте користећи различите улазне параметре када се тестирају функције и методе

- Када се тестирају класе објеката, неопходно је направити тест који покрива све особине објеката. Тачније треба:
  1. Тестирати све операције које су повезане са објектом
  2. Поставити и проверити вредности свих атрибута који су повезани са објектом
  3. Ставити објекат у конкретну употребу тј. симулирати ситуације које доводе до мењања стања објекта

Избор тестова за тестирање јединица

- Тестирање је скупо и може да троши доста времена и зато треба бирати ефективне тест случајеве. Ефективност у овом случају значи:
  1. Тест случај би требало да покаже да тестирана компонента ради оно што би и требало
  2. Ако постоји дефект у компоненти, тест случај би требало да га открије

- Увек је добро писати две врсте тест случајева. Први који описује функционалности софтвера и други који би абнормалним улазима тестирао компоненте на грешке

- Ове две стратегије помажу у успешном бирању тест случајева:
  1. Тестирање делова. Препознају се делови система са сличним особинама и тестирају се на исти начин.
  2. Тестирање на основу смерница. На основу претходног искуства се стварају смернице у тестирању и на основу њих се могу препознати евентуалне грешке.

Партиције еквиваленције

- Улазни и излазни подаци се често могу сврстати у различите класе због заједничких карактеристика.

- Један од приступа у дизајну тест случајева је базиран на индентификацији свих улазних и излазних партиција. Тест случајеви су дизајнирани тако да се сви улази и излази налазе унутар дефинисаних партиција

- Тестирање партиција може да буде употребљено за тестирање система али и за тестирање компоненти

- Када се партиције одреде, треба изабрати тест случајеве за сваку од њих. Добро правило за бирање тестова је да се изабере случај који скоро па излази ван граница партиције као и случај који одговара "средини" партиције

Партиције еквиваленције

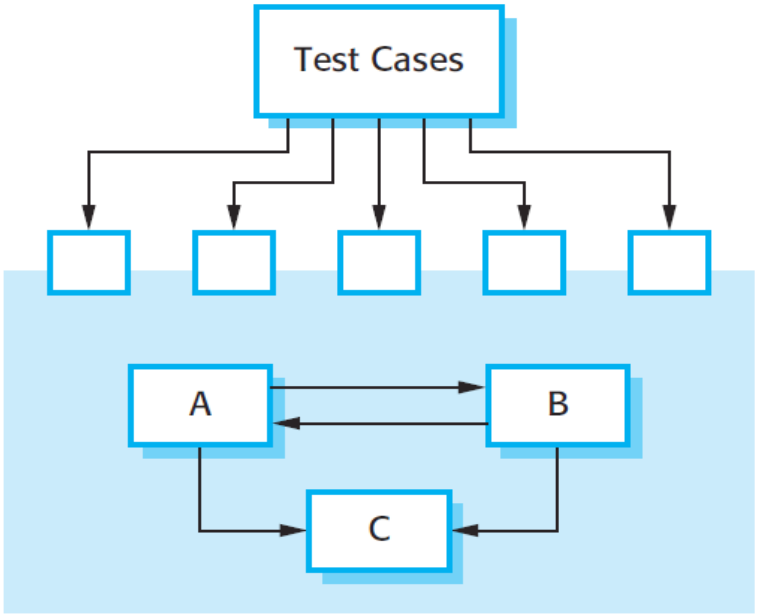
- Саме партиције се одређују на основу документације или искуства

- Када се користи документација за партиционисање онда се то назива "тестирање црне кутије" зато што није неопходно знати како систем ради. Тестирање црне кутије постаје тестирање беле кутије када помоћу анализе програмског кода правимо тестове.

- Основне водиље при дизајну тестова:
  1. Бирати улаз који тера систем на генерисање свих порука о грешкама
  2. Дизајнирати улазе који узрокују прекорачење улазних бафера (енг. Input buffers overflow)
  3. Понављати улазе више пута
  4. Форсирати генерисање неважећих излаза
  5. Форсирати резултате израчунавања који су превелики или премали

Тестирање компоненти

- Софтверске компоненте се обично састоје из више објеката између којих постоји интеракција. Функционалностима тих објеката се приступа путем интерфејса и тестирање компоненти треба да покаже да ти интерфејси правлно раде. Може се предпоставити да су тестови јединица обављени над појединачним објектима.



Слика 2. Тестирање интерфејса

Тестирање компоненти

- Постоје различити типови интерфејса:
  1. Параметарски интерфејси. Ово су интерфејси где подаци или референце на функције бивају прослеђивани са једне компоненте на другу. Методи у објектима имају параметарски интерфејс.
  2. Интерфејси дељене меморије. У овим интерфејсима блокови меморије бивају дељени између компоненти. Подаци су остављени у меморији од стране једног подсистема док их други подсистем прикупља. Овај тип интерфејса се користи у ситуација где, на пример, неки сензори стварају податке које даље прикупљају и обрађују остале компоненте система.
  3. Процедурални интерфејси. Ово су интерфејси где компонента енкапсулира скуп процедура које могу бити позване од стране друге компоненте.
  4. Интерфејси за прослеђивање порука. Ово су интерфејси где једна компонента захтева услугу од друге компоненте помоћу преношења порука. Повратна порука је резултат обављене услуге.

Тестирање компоненти

- Постоје различити типови грешака интерфејса:
  1. Погрешно коришћење интерфејса. Компонента позива неку другу компоненту и настаје грешка у коришћењу. Ова грешка се јавља у параметарским интерфејсима где су параметри погрешног типа или се прослеђују погрешним редом.
  2. Неразумевање интерфејса. Компонента не разуме интерфејс компоненте коју позива и прави претпоставке о њеном понашању. Позвана компонента се не понаша како је претпостављено и настаје грешка.
  3. Грешке у тајмингу. Ове грешке се јављају код интерфејса дељене меморије или интерфејса за прослеђивање порука. Произвођач података и прималац података могу да имају различите брзине и ту настаје грешка.

- Неке смернице у тестирању интерфејса:
  1. Истражи код који треба да буде тестиран. Дизајнирају се тестови са екстремним улазима.
  2. Ако су показивачи преношени преко интерфејса увек тестирати нулти показивач.
  3. Ако се компонента позива кроз процедурални итерфејс дизајнирати тест тако да она намерно пријави грешку.
  4. Користити стрес тестирање у системима за прослеђивање порука. Ефикасно откривање проблема у тајмингу.
  5. Ако више компоненти врши интеракцију кроз дељену меморију дизајнирати тестове који активирају компоненте у различитим редоследима.

Тестирање система

- Тестирање система у фази развоја укључује интеграцију компоненти у циљу креирања верзије система и потом се тестира тај интегрисани систем. Тестирање система проверава да ли су компоненте компатибилне, да ли правилно ступају у интеракцију и да ли се одговарајући подаци у право време преносе преко интерфејса.

- Очигледо постоји сличност са тестирањем компоненти али ту су две битне разлике:
  1. Током тестирања система, компоненте, које се могу више пута употребити, се независно развијају и могу да буду интегрисане са новим компонентама. Онда се комплетан систем тестира
  2. Компоненте које су различити тимови развили могу да буду интегрисане у овом кораку. Тестирање система је колективан процес.

- Нека понашања система постају очигледна тек када се компоненте интегришу. Та понашања такође треба тестирати. Зато се тестирање система треба фокусовати на интеракцију између објеката који чине систем.

- Због тих интеракција, тестирање базирано на случајевима употребе је веома корисно

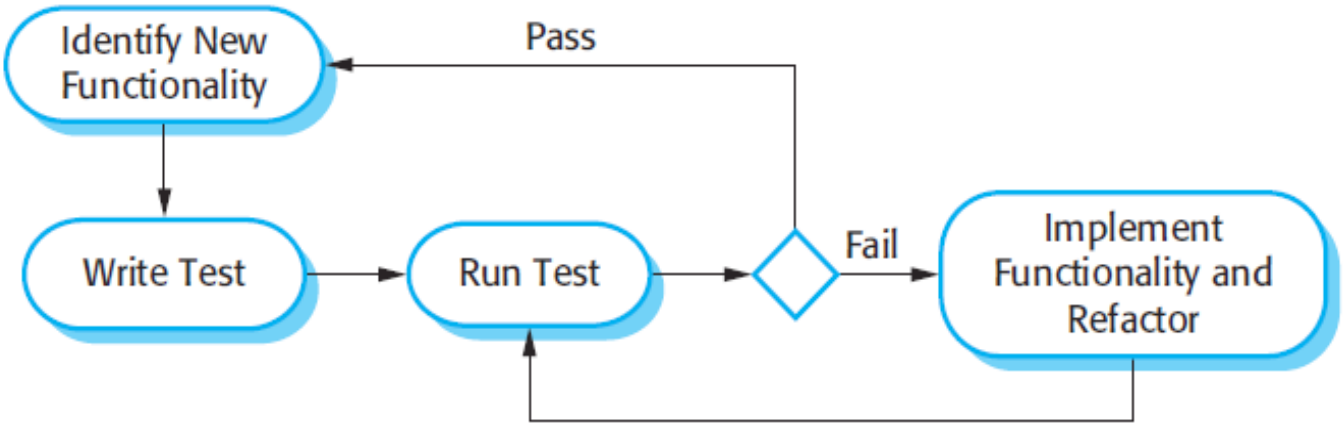
Развој вођен тестовима

Развој вођен тестовима

- Развој вођен тестовима (РВТ) је приступ где се мешају развој кода и самих тестова. У суштини, код се развија инкрементално а са њим и тестови за сваки инкремент. Не започиње се следећи инкремент док тренутни не прође развијене тестове.

- Основни кораци у РВТ:
  1. Идентификује се одређена функционалност.
  2. Пише се тест за функционалност и имплементира се као аутоматски тест. То значи да ће тест да се изврши и да ће да нас обавести о успеху извршавања.
  3. Онда се покреће тест, заједно са свим тестовима који су већ имплементирани. Сама функционалност не мора бити развијена, тако да у том случају нови тестови пријављују грешку.
  4. Имплементира се функционалност и тестови се поново покрећу.
  5. Када сви тестови прођу онда се прелази на следећу функционалност.

- Предност ове врсте тестова је да помажу програмеру да разјасни шта који део кода треба да ради.



Слика 3. Развој вођен тестовима

Тестирање издања

Тестирање издања

- Тестирање издања (ТИ) је процес тестирања одређеног издања система које је предвиђено за употребу ван развојног тима.

- Издања су углавном за кориснике, али је такође могуће да буду и за друге тимове који развијају систем (на пример за менаџере који припремају систем за продају)

- Постоје две основне разлике између тестирања издања и тестирања система:
  1. Тим који није био укључен и тестирање система треба да буде одговоран за тестирање издања.
  2. Тестирање система треба да се фокусује на проналажење багова и дефеката, док тестирање издања проверава да ли систем договара захтевима и да ли је довољно добар за употребу.

- Основни циљ ТИ-а је да пружи уверење да је систем довољно добар.

- ТИ ради обично по принципу црне кутије где су тестови изведени из спецификација система. Други назив за ово је функционални тест пошто је у фокусу функционалност а не имплементација.

ТИ: Тестирање захтева

- Добри захтеви су они за које могу тестови да се напишу

- Тестирање захтева је систематичан приступ креирању тестова зато што се сви тестови изводе из одређених захтева

- Тестирање захтева је пре валидације него тестирање дефеката зато што се демонстрира да је систем имплементиран на основу одговарајућих захтева

- Тестирање захтева се обично своди на писање скупа тестова (а не једног теста) да би се осигурала покривеност целог система

ТИ: Тестирање сценарија

- Тестирање сценарија је приступ где се смишљају типични сценарији употребе који се користе за прављење тест случајева

- Сценарио је прича која описује један начин коришћења система

- Ако постоје проблеми са системом ови тестови треба то да препознају

- У овом приступу, један сценарио укључује више захтева тако да се тестира и различита комбинација захтева

ТИ: Тестирање перформанси

- Тестирање перформанси показује да ли систем може да поднесе предвиђено оптерећење. Обично се реализује као серија тестова, где сваки нови тест доноси и веће оптређење. Тестови се извршавају док систем не "пукне".

- Операциони профили се праве уколико хоћемо да покажемо да ли систем задовољава захтеве преформанси. Операциони профил је скуп тестова који описују стварну мешавину послова коју ће систем да обавља.

- Најефикаснији начин за откривање дефеката је дизајнирање тестова који раде на ивици система. Ово се назива стрес тестирање и има две функције:
  1. Тестира систем на неуспех. Околности могу да проузрокују неочекиване комбинације догађаја где оптерећење система постаје превелико. Тада треба обезбедити да систем очува своје податке.
  2. Стресирање система може да открије дефекте који у нормалним условима не би били откривени.

Корисничко тестирање

Корисничко тестирање

- У овом делу тестирања корисници обезбеђују улазе за тестове и дају савете у тестирању система.

- Корисничко тестирање је кључно чак и када су извршена обимна тестирања система и захтева. Представља праву употребу система која не може бити опонашана од стране тестера.

- Три типа корисничког тетирања:
  1. Алфа тестирање. Корисници раде заједно са развојним тимом на тестирању софтвера
  2. Бета тестирање. Издање софтвера је доступно корисницима којима је дозвољено да експериментишу са системом.
  3. Тестови прихватања. Купци тестирају систем и одлучују да ли је спреман да буде прихваћен и пуштен у употребу.

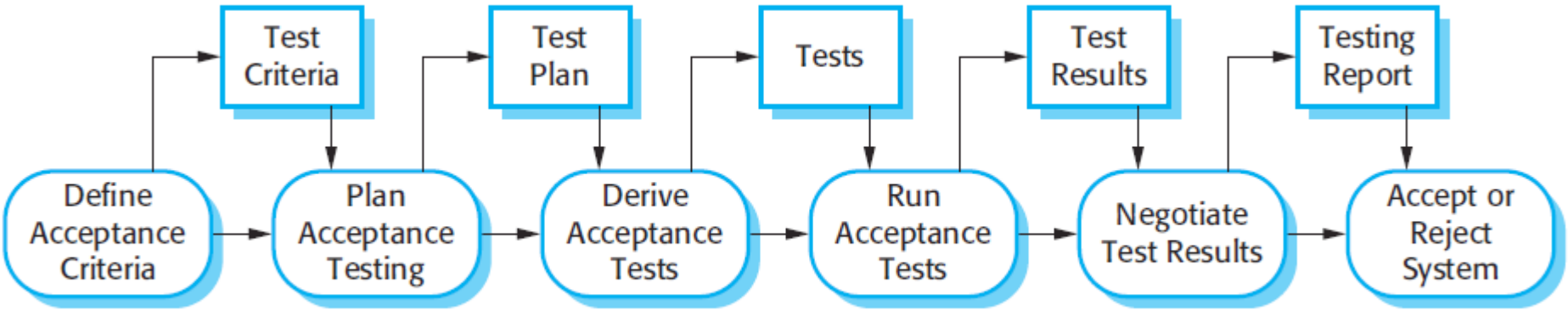
- У алфа тестирању корисници и развијаоци раде заједно на систему у току његове изградње. То значи да корисници препознају пороблеме који нису очигледни тестерима.

- Бета тестирање се обавља када рано, некада и не довршено, издање софтвера буде доступно купцима и корисницима на употребу. Бета тестери могу да буду селектована група људи

- Тестови прихватања се обављају након тестова издања. Купац тестира систем и одлучује да ли ће да га прихвати. Прихваћање подразумева плаћање за систем.

Корисничко тестирање

- Постоји шест фаза у тестирању прихватања:
  1. **Дефинисање критеријума прихватања.** Било би идеално када би ова фаза била рано у процесу, пре него што је потписан уговор за систем. Критеријум прихватљивости би требало да будео део уговора усаглашеног између купца и развиаоца. У пракси је то тешко изводиво у раној фази.
  2. **Планирање тестова прихватљивости.** Ово укључује одређивање ресурса, времена и буџета за тестове прихватљивости као и одређивање распореда тестирања. Ова фаза такође треба да дефинише ризике тестирања и да одреди како ти ризици могу бити ублажени.
  3. **Извођење тестова прихватљивости**. Када је критеријум прихватљивости одређен тестови треба да буду дизајнирани тако да провере да ли је систем прихватљив. Тестови прихватљивости треба да тестирају функционалне и не функционалне аспекте система.
  4. **Извршавање тестова.** Изведени тестови се сада извршавају. У идеалној ситуацији ово се обавља у реалној средини коришћења система. Тешко је аутоматизовати ову фазу зато што се често заснива на интеракцији корисник-систем.
  5. **Преговарање резултата.** Очекивано је да не прођу сви тестови и да постоје проблеми са системом. У том случају купац и развиаоц преговарају да ли да пусте систем у употребу или је потребан даљи развој.
  6. **Прихватање/одбијање система.** Ова фаза укључује сусрет развиаоца и купаца и коначно одлучивање о прихватању. Ако систем није довољно добар шаље се на дораду и понавља се комплетан процес тестирања.



Слика 4. Процес тестирања прихватања

Корисничко тестирање

- У агилним методологијама попут екстремног програмирања тестови прихватљивости имају мало другачије значење.

- Основна разлика је да је корисник део тима за тестирање.

- Он је задужен за развијање тестова који проверавају корисничку страну.

- Тестови су аутоматизовани и не прелази се у следећу фазу док коринсички тест не прође.

- Такав начин тестирања може да представља свакодневан посао за корисника

?

БранкоПоповић

popovich.branko@gmail.com