REPUBLIKA SLOVENIJA
VRHOVNO SODIŠČE
CENTER ZA INFORMATIKO

Smernice pri razvoju aplikacij 2023-2027

Vrhovno sodišče Republike Slovenije, Miklošičeva 4, 1000 Ljubljana
Center za informatiko, T: 01 366 42 50, EP: cif.vsrs@sodisce.si

Kontrola verzij

Zadnja verzija:

Verzija	1.0
Datum	8. 5. 2023
Avtor	Andrej Gogala, CIF VSRS
Odgovornost	Bojan Muršec, CIF VSRS
Zaupnost	INTERNO
Datoteka	Z:\Moji dokumenti\1 - JN Učinkovito pravosodje\JN 2023\ nadgradnja spletišča sodnapraksa.si\Smernice pri razvoju aplikacij 2023-2027.odt

Zgodovina:

Verzija	Datum	Avtor	Opis
1.0	2. 9. 2019	Andrej Gogala	Prvi osnutek.
1.1	8. 5. 2023	Andrej Gogala	Posodobitev v skladu s celostno grafično podobo.

Revizije:

Revizija	Datum	Avtor	Opis

Zaščita dokumenta

Kazalo vsebine

1 Uvod

Dosedanje izkušnje pri uvedbi informacijskih sistemov v sodstvu so pokazale, da je nadaljnje vzdrževanje zaradi raznovrstnosti uporabljenih tehničnih rešitev oteženo. Tako npr. so debeli klienti implementirani v treh možnih JAVA tehnologijah – Swing, RPC in JavaFX.

Smiselno je, da se število različnih tehnologij omeji, saj je omejeno tudi število zaposlenih na CIFu, ki morajo v končni fazi vzdrževati vse te aplikacije.

Pričujoči dokument zato opredeljuje smernice poenotenja tehničnih rešitev in definira navodila za implementacijo novih tehničnih rešitev v sistemu sodstva.

Smernice veljajo za obdobje 2023-2027, saj se tehnologije hitro spreminjajo. Po preteku dveh let, bo narejena revizija smernic.

1.1 Kriteriji za izbiro posameznih tehnologij

Pri izbiri tehnologij smo upoštevali naslednje kriterije:

- odprto kodna rešitev;
- razširjena uporabniška baza, t.j. rešitev z veliko aktivnimi uporabniki, ki se redno dopolnjuje / razvija;
- lokalno znanje – v Sloveniji je razširjena uporaba;
- poznavanje tehnologije na CIFu;
- ustrezna uporabniška izkušnja;
- dokaj enostavna izdelava avtomatiziranih testov;
- ustreza načelom izgradnje informacijske podpore CIF.

2 Načela pri izgradnji informacijske podpore

Informacijska podpora naj bo grajena v skladu z dobro informacijsko prakso, upoštevaje naslednja načela razvoja informacijskih sistemov:

- **Fleksibilnost**

  Izvajalec mora stremeti k čim večji fleksibilnosti, kar je še posebej pomembno pri podpori za izpise, večji podpori pri uporabnosti programa ter pripravljenosti, podpreti manjše nove zahteve ali spremembe zahtev v kratkem času.

- **Povezljivost**

  Celoten sistem mora biti načrtovan tako, da povezovanje z dodatnim novim sistemom ne povzroča nesorazmerno velikega vložka dela.

- **Stabilnost in zanesljivost**

  Poudarek mora biti na stabilnosti sistema ter omogočanju hitre diagnostike in reševanja težav.

- **Sledljivost**

Sistem mora omogočati sledenje spremembam v podatkih – to je pomembno z vidika obravnave osebnih podatkov in z vidika sposobnosti spremljanja sprememb.

- **Prijaznost do uporabnika**

  Informacijska podpora mora biti čim enostavnejša za uporabo, tako da bo delo čim manj zamudno.

- **Varnost:**

  Sistem mora biti zasnovan tako, da se dostopne pravice do posameznih delov informacijske podpore omogočijo le na podlagi avtentikacije uporabnika in ustreznih uporabniških vlog.

- **Optimalnost delovanja in skalabilnost**

  Poudarek je na tistih delih sistema, ki bodo pod največjimi obremenitvami.

3 Zahteve pri izgradnji informacijske podpore

Pri izgradnji sistema mora izvajalec upoštevati naslednje osnovne usmeritve:

1. Tri-nivojska arhitektura
2. Storitveno naravnan sistem (SOA)
3. Neodvisnost posameznih nivojev
4. Uporaba odprtih standardov
5. Skalabilnost in atomarnost
6. Večkratna uporaba modulov
7. BPMN za podporo poslovnim procesom
8. Uporaba javanske tehnologije na srednjem nivoju
9. Uporabniški vmesnik kot spletna aplikacija
10. Razvojna orodja pri naročniku
11. Produkcijsko okolje
12. Odzivnost in obremenljivost sistema
13. Omejitev velikosti in oblike prilog ter števila in oblike slik
14. Povezovanje z obstoječimi servisi in podatkovnimi tabelami naročnika
15. Nadzor nad delovanjem sistema
16. Povezovanje z zunanjimi servisi
17. Generiranje dokumentov in poročil
18. Režim delovanja
19. Revizijske sledi
20. Avtentikacija in avtorizacija uporabnikov

3.1 Tri-nivojska arhitektura

Pri načrtovanju tri nivojske arhitekture je potrebno upoštevati strogo ločevanje posameznih nivojev.

3.2 Storitveno naravnan sistem (SOA)

Pri izgradnji sistema se osredotočimo na implementacijo določenih storitev (funkcionalnosti poslovne logike). Storitve so jasno definirane atomarne funkcionalnosti sistema, ki niso odvisne od določenih stanj ali vsebine drugih funkcij. To pomeni, da za uporabnika sistema (odjemalska aplikacija ali drugi sistem) podatkovni nivo ni viden, oziroma dosledneje, ni vidna njegova fizična realizacija. Odjemalec tako uporablja le osnovne funkcije, ki izhajajo iz logičnega poslovnega modela, poslovno logiko in transformacijo poslovnih funkcij na podatkovni model pa izvaja aplikativni nivo. Odjemalec torej pozna le poslovne funkcije srednjega nivoja, ki so natančno definirane in dostopne po vnaprej dogovorjenem standardu. Vse aplikacije na srednjem nivoju so dostopne po enotnem standardu ne glede na tehnologijo, na kateri se izvajajo.

3.3 Neodvisnost posameznih nivojev

Na posameznih nivojih se izvajajo funkcionalnosti, kot narekuje sistem tri-nivojske arhitekture, in sicer:

- podatkovni nivo (database layer),
- srednji – aplikativni nivo (middletier) ter
- odjemalec (client tier).

Podatkovni nivo je namenjen izključno shranjevanju, posodabljanju in zajemanju podatkov. Uporaba programske kode za samostojno izvajanje določenih funkcij na tem nivoju ni dovoljena, razen v izrednih primerih, ko ni druge rešitve, in z dovoljenjem naročnika. Podatkovni nivo lahko komunicira izključno le s srednjim nivojem, torej z aplikacijskim strežnikom. Za povezovanje naj se uporabljajo odprti standardi.

Srednji nivo implementira vso poslovno logiko in kontrole, ki so potrebne za zagotavljanje želenih funkcionalnosti.

3.4 Uporaba odprtih standardov

Povsod, kjer je mogoče, naj se uporabljajo odprti standardi (OASIS – http://www.oasis-open.org, W3C http://www.w3c.org). Enako velja za komunikacijske protokole na vseh nivojih, programske jezike in interpreterje (SQL, XPATH, XQUERY...), povezovalne protokole in ostalo.

3.5 Skalabilnost in atomarnost

Sistem naj bo zgrajen iz obvladljivih funkcionalnih gradnikov, ki naj ne presegajo razumne meje obsega. Funkcionalnosti srednjega nivoja se delijo na dva nivoja, in sicer:

- na najmanjše logične enote – atomarne funkcionalnosti ter
- poslovna pravila – storitve, ki so sestavljene iz ene ali več atomarnih funkcij oziroma njihovega zaporedja.

3.6 Večkratna uporaba modulov

V največji možni meri naj se pri implementaciji posameznih modulov uporabijo že obstoječe funkcionalnosti znotraj projekta in znotraj organizacije. Uporabljena rešitev mora uporabiti obstoječe skupne servise, ki so standardni gradniki rešitev za vsa področja.

3.7 BPMN za podporo poslovnim procesom

Za podporo poslovnim procesom se uporabi BPMN ogrodje, ker olajša komunikacijo med vsebinskim razumevanjem poslovnega procesa in tehnično izvedbo. Hkrati so možne relativno enostavne naknadne spremembe poslovnega procesa.

3.8 Uporaba javanske tehnologije na srednjem nivoju

Izbrana tehnologija za implementacijo srednjega nivoja je Spring Boot. Servisi srednjega nivoja naj bodo dosegljivi preko REST vmesnika. Definicija, kaj REST je, je tule: https://www.restapitutorial.com/. Minimalna zahteva stopnje zrelosti za implementacijo REST je 2 (glej https://martinfowler.com/articles/richardsonMaturityModel.html).

3.9 Uporabniški vmesnik kot spletna aplikacija

Izbrana tehnologija za implementacijo spletnih vmesnikov je Vue.js in/ali Angular. Skupne komponente se definirajo kot spletne komponente (WebComponents), tako da so uporabne v obeh tehnologijah.

3.10 Razvojna orodja pri naročniku

- Za vodenje in kontrolo nad verzijami izvorne programske kode uporablja naročnik sistem Subversion (SVN), nameščen na Centru za informatiko Vrhovnega sodišča Republike Slovenije. Ob predhodnem dogovoru, se lahko uporabi tudi sistem Gitbucket, prav tako nameščen na Centru za informatiko Vrhovnega sodišča Republike Slovenije. Izvajalec mora ob predajah programskih paketov izvorno kodo odložiti v sistem SVN/Git. Naročnik za prevajanje paketa uporablja orodje Maven. Izvajalec mora predati kodo na način, da se bo paket prevedel z orodjem Maven. To velja tudi za uporabniški vmesnik.
- Prevajanje izdelkov se izvaja pri naročniku avtomatično po sistemu tekoče integracije (continuous integration). Za tekočo integracijo se uporablja sistem »Jenkins« (https://jenkins.io/), ki teče na operacijskem sistemu Linux in je nameščen pri naročniku. Za hrambo knjižnic se uporablja aplikacija Sonatype Nexus.
- Za izvedbo GUI testov se uporablja orodje Selenium.

- Za test REST vmesnikov se uporablja Postman.
- Za statično analizo programske kode se uporablja orodje SonarQube.

3.11 Produkcijsko okolje

- Predpisana tehnologija prezentacijskega (odjemalčevega) nivoja je Vue.js in/ali Angular, srednjega nivoja pa Java.
- Servisi srednjega nivoja so narejeni v ogrodju Spring Boot. Uporabljena mora biti verzija Jave 17 ali višja. Aplikacija se zaganja v samostojnem procesu z integriranim Tomcat strežnikom.
- Podatkovni strežnik je Oracle 12.1.
- Za sporočilno vrsto se uporablja RabbitMQ.
- Operacijski sistem za srednji in za podatkovni nivo je 64 bitni Linux – Oracle Linux verzije 9.
- Razvoj in testiranje mora potekati na bitno kompatibilni arhitekturi ter na testnih podatkovnih bazah.
- Odjemalske aplikacije tečejo na raznih verzijah MS Windows (10/11). V primeru debelih odjemalcev je obvezna implementacija v Javi, na način, ki zagotavlja neodvisnost od platforme (brez specifičnih rešitev odvisnih od MS Windows).
- Za notranjo komunikacijo med sklopi sistema se uporablja državno omrežje HKOM s prenosnimi hitrostmi najmanj 10 Mb/s.
- Spletna aplikacija mora delovati vsaj na brskalniku Firefox verzije ESR ter na brskalniku Iridium (https://iridiumbrowser.de/) za notranje uporabnike ter zadnje verzije popularnih brskalnikov za zunanje uporabnike (Edge, Chrome, Firefox in Safari). Zunanje aplikacije morajo biti prilagojen tudi uporabi z mobilnimi napravami.
- Za sistem elektronskega učenja (e-learning) se uporablja sistem Moodle.

3.12 Odzivnost in obremenljivost sistema

Za vsak sistem mora naročnik predpisati metrike, katerim mora zadoščati naročen sistem. Npr.:

- odzivnost sistema – trajanje nalaganja posamezne strani;
- pretočnost sistema oziroma število klicev na časovno enoto.
- ...

3.13 Omejitev velikosti in oblike prilog ter števila in oblike slik

Za vsak sistem morajo biti definirane omejitve velikosti datotek ter njihov format, ki jih sistem še podpira, tako za vhodne kot izhodne datoteke. Ob prekoračitvi omejitev, mora

sistem imeti vgrajene ustrezne kontrole, ter uporabnika obvestiti o prekoračitvi omejitev.

3.14 Povezovanje z obstoječimi servisi in podatkovnimi tabelami naročnika

Pri razvoju sistema mora izvajalec uporabiti obstoječe skupne servise in podatke (na primer imenik sodnikov). Dostop do teh virov je standardiziran preko spletnih servisov.

3.15 Nadzor nad delovanjem sistema

Sistem naj omogoča nadzor nad delovanjem ključnih komponent sistema (delovanje posameznih ključnih servisov, produkcijskih parametrov in povezav z zunanjimi sistemi) z uporabo obstoječih sistemov Nagios in Cacti.

Za vsak servis morajo biti na voljo trije podatki:

- število klicev na časovno enoto;
- število napak na časovno enoto;
- trajanje posameznega klica servisa.

Ti podatki naj bodo na voljo preko JMX storitev, ki naj bodo dostopne preko HTTP vmesnika Jolokia (glej poglavje 4.2 Namestitvena pravila).

Klici servisov se prav tako beležijo v dnevnike izvajanja. Dnevniki morajo omogočiti kasnejšo analizo težav v delovanju sistema. Izvajalec zagotovi pravila za strojno branje dnevnikov z orodjem Logstash ali pa zagotovi beleženje dnevnikov v JSON obliki, tako da jih lahko orodje Filebeat samodejno prebere in razstavi na sestavne dele.

Vsi dostopi do aplikacije potekajo preko enotne dostopne točke (posrednika - apache proxy), na strežniku app.sodisce.si. Za vsak zahtevek posrednik generira enoličen ID zahtevka, ki ga posreduje srednjemu nivoju. Ob zapisovanju dnevnikov, se v dnevnik zapiše ta enoličen ID. Ob klicih naslednjih servisov, se mora uporabiti enoličen ID zahtevka. Le to omogoča spremljanje izvajanja vseh servisov, povezanih s posameznim zahtevkom na strežniku.

3.16 Povezovanje z zunanjimi servisi

Predvidene povezave na zunanje sisteme se izvedejo s klicem spletnih storitev. Pri tem mora izvajalec upoštevati dobre prakse, in sicer:

- za vsak klic mora biti definirana časovna omejitev, v kateri se le-ta mora zaključiti;
- pri klicih zunanjih servisov mora biti uporabljen vzorec varovalke (Circuit Breaker pattern), ki prepreči obremenitev strežnika v težavah; informacija o tem, da se je varovalka vklopila, mora biti na voljo nadzornemu modulu preko JMX vmesnika (glej poglavje 3.15 Nadzor nad delovanjem sistema). Preko JMX vmesnika mora biti na voljo tudi možnost vklopa oz. izklopa varovalke.

Pri pridobivanju podatkov iz zunanjih evidenc naj se uporabi predpomnilnik za tiste zunanje evidence, za katere se dogovorita izvajalec in naročnik.

3.17 Tvorjenje dokumentov in poročil

Za izdelavo dokumentov (obrazcev in vzorcev) in poročil (izpisov in pregledov) se uporablja Oracle Business Intelligence Publisher (BIP). Pri generiranju pisanj (vzorcev), pri katerih uporabniki lahko posegajo v vsebino, se za urejanje dokumentov uporablja OpenOffice.org.

3.18 Režim delovanja

Za sistem mora biti definiran režim delovanja. Za doseganje tega režima delovanja mora sistem omogočati podvajanje komponent za zagotavljanje razpoložljivosti.

V primeru odpovedi zunanjih servisov mora izdelek nuditi tako delovanje, da uporabniška izkušnja ni zmanjšana v tistih delih, ki niso odvisni od nedelujočih zunanjih servisov. Deli, ki so odvisni od zunanjih servisov, pa morajo bodisi ponuditi okrnjeno ali nadomestno funkcionalnost oziroma na uporabniku razumljiv in prijazen način sporočiti, da ta del funkcionalnosti trenutno ni na voljo.

Prenos podatkov mora biti zanesljiv – v primeru odpovedi povezave med povezanimi sistemi se nobeno sporočilo ne sme izgubiti. Po ponovni vzpostavitvi povezave se prenesejo vsa čakajoča sporočila.

3.19 Revizijske sledi

Za vsako podatkovno tabelo mora obstajati tabela z enakimi atributi, v kateri se spremlja zgodovina sprememb. CIF razpolaga z orodjem, s katerim se generirajo DDL stavki za tovrstne tabele, hkrati z prožilci (triggerji) na originalnih tabelah za beleženje sprememb.

Vsak sistem mora definirati način beleženja revizijske sledi v primeru vpogledovanja preko uporabniških vmesnikov. Zabeležiti je potrebno naslednje podatke:

- predmet vpogleda
- uporabnik
- IP računalnika
- čas vpogleda
- opcijsko tudi razlog vpogleda, če tako izhaja iz predpisov

3.20 Avtentikacija in avtorizacija uporabnikov

Za avtentikacijo uporabnikov se uporablja protokol OpenID Connect/OAuth2, implementiran s sistemom KeyCloak.

Avtorizacija uporabnikov se izvaja preko internega sistema vlog – aplikacija Razpored. Informacijo o pripadnosti uporabnika posamezni vlogi dobi sistem bodisi preko žetonov iz sistema KeyCloak, bodisi preko klica skupnih spletnih storitev.

4 Postavitev informacijske podpore na infrastrukturo

Informacijska podpora je nameščena na infrastrukturi na Centru za informatiko Vrhovnega sodišča Republike Slovenije.

4.1 Optimalnost aplikacije in baznih objektov

Glede na to, da bodo vse kategorije aplikacij in pripadajočih baznih objektov delovale na skupni strežniški infrastrukturi ter zaradi optimalne izkoriščenosti skupne infrastrukture je izvajalec dolžan optimizirati programsko kodo in bazne objekte s ciljem zagotavljanja optimalnega delovanja. Vse neoptimalnosti, ki se izkažejo skozi obremenitveni test in skozi generalni test, mora izvajalec odpraviti do trenutka produkcije. Prav tako mora izvajalec vsako spremembo ali nadgradnjo aplikacije predhodno preveriti tudi s performančnega stališča. Lastnik infrastrukture izvaja periodične preglede optimalnega delovanja. Priporočila, ki nastanejo na podlagi takih pregledov, je izvajalec dolžan v najkrajšem razumnem roku upoštevati.

Izdelava statističnih poročil ne sme imeti vpliva na performančno delovanje ostalih delov sistema.

4.2 Namestitvena pravila

Za informacijsko podporo se vzpostavijo naslednja okolja:

- testno okolje služi potrditvenemu testiranju (tj. preverjanju, ali je bil nek popravek izveden v skladu z željami naročnika; testiranje pravilnosti kode se izvaja na strani izvajalca);
- produkcijsko okolje služi polni produkciji; sem se nameščajo samo popravki, katerih prehod iz testa na produkcijo je bil po predpisanem protokolu odobren;
- šolsko okolje je namenjeno izobraževanju uporabnikov in naj bi bilo po verzijah aplikacij izenačeno s produkcijskim;
- okolje za avtomatsko preverjanje delovanja; na tem okolju se izvajajo avtomatizirani testi preverjanja delovanja.
- Nove verzije/popravki aplikacije in baznih objektov se najprej namestijo na testno okolje. Odgovorni predstavnik naročnika opravi najmanj naslednja preverjanja:
- da je bila namestitev opravljena v skladu z zahtevami iz poglavja 4 Postavitev informacijske podpore na infrastrukturo,
- da aplikacija deluje v skladu s funkcionalnimi pričakovanji;
- da je aplikacija tudi performančno ustrezna in deluje v skladu s pričakovanji.

Šele na podlagi pozitivnega izida tega potrditvenega testa in izjave odgovornega, da je bil test pozitivno opravljen, se lahko aplikacija namesti na produkcijo.

Po namestitvi na produkcijo naročnik preveri delovanje po enakem vzoru, kot je bila narejena verifikacija na testu:

- odgovorni predstavnik naročnika izvede potrditveni test na produkciji, ki sestoji najmanj iz naslednjega preverjanja:
  - da je bila namestitev opravljena v skladu z z zahtevami iz poglavja 4 Postavitev informacijske podpore na infrastrukturo,
  - da aplikacija deluje v skladu s funkcionalnimi pričakovanji,
  - da je aplikacija tudi performančno ustrezna in deluje v skladu s pričakovanji.

Potrditveni test mora obsegati poleg delovanja same aplikacije tudi delovanje podatkovne zbirke in ustreznost baznih objektov.

Po pričetku projekta bo naročnik izvajalcu kot osnovo razvojnega okolja predal odtis navidezne naprave (virtual machine), na katerem bodo v docker okolju tekli vsi predvideni zunanji servisi. Le ta služi tudi kot okolje za izvajanje avtomatskih testov. Alternativno bo naročnik izvajalcu predal samo docker podobe z razvojnim okoljem.

Razvojno okolje in testno okolje za razvojno testiranje si vzpostavi izvajalec sam v svojem okolju za potrebe razvoja informacijske podpore. Postavitev tega okolja ni del aktivnosti projekta, ki so predmet dogovora z naročnikom.

4.3 Obvladovanje sprememb

Od izvajalca se pričakuje, da ima izdelane in uveljavljene postopke obvladovanja sprememb (repozitorij, številčenje različic). Izvajalec je dolžan voditi evidenco in vse spremembe ustrezno označevati. Pravilo velja tako za spremembe aplikacije kot za spremembe baznih objektov. Za vse spremembe, ki jih izvajalec načrtuje, mora izvesti postopke obveščanja tako naročnika kot upravljavca infrastrukture.

Vsaka sprememba v sistemu SVN/Git mora biti opremljena s komentarjem, ki vsebuje številko zahtevka v naročnikovem sistemu Redmine in opis zahtevka. Več o dobrem pisanju komentarjev je tule: https://github.com/RomuloOliveira/commit-messages-guide.

4.4 Obremenitveni test

Za kritične aplikacije ali celotni sistem lahko naročnik zahteva izvedbo obremenitvenega testa. Obremenitveni test se običajno izvaja skupaj s strokovnjaki Centra za informatiko Vrhovnega sodišča Republike Slovenije. Izvajalec je dolžan pripraviti ustrezne scenarije in podatke. Naročnik si pridržuje pravico določitve orodja za izvajanje obremenitvenih testov.

4.5 Generalni test

Za kritične aplikacije lahko določi naročnik izvedbo generalnega testa. Generalni test pomeni hkratno vajo vseh uporabnikov. Za generalni test je izvajalec dolžan pripraviti vsebinske in postopkovne scenarije.

4.6 Testi enot

Predana koda mora biti opremljena s testi enot, ki se izvedejo med prevajanjem. Ti morajo pokriti vsaj 60% vse kode. Pokritost kode se meri z orodjem JaCoCo. Za posnemanje odvisnih komponent se uporabi knjižnica Mockito. S testi enot skušamo med drugim zagotoviti čisto kodo, tj. se izogniti globalnemu stanju, statičnim metodam in kreiranju novih objektov v objektih, kjer se uporabljajo. Prav tako testi enot demonstrirajo uporabo posameznih funkcij. Testi enot morajo biti narejeni v skladu s smernicami, opisanimi v predavanju Miška Heveryja „How to Write Clean, Testable Code“.

4.7 Integracijski testi

Izvajalec mora za integracije posameznih modulov oziroma komponent izdelati tudi avtomatsko preverjanje integracij.

Tovrstni testi morajo preveriti večino poslovne logike srednjega nivoja. Na ta način preverjamo, ali je vsa poslovna logika z vsemi kontrolami implementirana na srednjem nivoju (glej poglavje 3.3 Neodvisnost posameznih nivojev). Pred zagonom avtomatskega integracijskega testa mora test poskrbeti za pravilno stanje podatkov v podatkovni bazi.

4.8 Avtomatizirano testiranje uporabniškega vmesnika

Za preverjanje delovanja celotne aplikacije izvajalec izdela avtomatsko preverjanje s pomočjo orodja Selenium. Za poganjanje teh testov bo naročnik uporabil okolje za avtomatsko testiranje, navedeno v poglavju 4.2 Namestitvena pravila. Avtomatski test mora sam vzpostaviti pravilno stanje podatkov v podatkovni bazi pred izvedbo testa.

Vsako novo izdelano funkcionalnost mora izvajalec opremiti z novimi avtomatskimi testi. Avtomatski testi so izvedeni na podlagi scenarijev, ki jih predhodno potrdi naročnik. Uspešnost samodejne izvedbe testov je eden izmed pogojev pri prevzemu aplikacije.