Analiza uticaja izmena i selekcija regresionih testova

Teorijske osnove, metrike kvaliteta i praktična primena alata STARTS, Ekstazi i JaCoCo RTS

STARTS
Statička selekcija (nivo klase)

Ekstazi
Dinamička selekcija (nivo klase)

JaCoCo RTS
Dinamička selekcija (nivo linije/metode)

1. Osnovne definicije i značaj u automatizovanom testiranju

Zašto ne pokrećemo sve testove (Retest-All) nakon svake izmene u CI/CD okruženju?

Ključni pojmovi

• Regresiono testiranje: Ponovno izvršavanje postojećih testova nakon izmene koda radi provere da nije narušeno postojeće ponašanje sistema.

• Analiza uticaja izmena (Change Impact Analysis): Identifikacija svih delova softvera (klasa, metoda, resursa) na koje direktno ili indirektno utiče izmena.

• Selekcija regresionih testova (Regression Test Selection): Izbor isključivo onog podskupa testova koji je relevantan za napravljenu izmenu, dok se ostali preskaču.

Značaj u CI/CD automatizovanom testiranju

• Brži Feedback Loop: Programeri znatno brže dobijaju informaciju da li je njihov commit uneo grešku.

• Ušteda resursa: Manje opterećenje CI/CD servera, niži troškovi infrastrukture u oblaku i manja potrošnja energije.

• Očuvana pouzdanost: Za razliku od nasumičnog preskakanja, formalni RTS algoritmi garantuju da relevantni testovi neće biti izostavljeni.

• Problem Retest-All pristupa: Kod velikih sistema pokretanje svih testova traje satima i usporava isporuku.

2. Klasifikacija pristupa selekciji regresionih testova

Podela RTS tehnika prema načinu prikupljanja zavisnosti i nivou granulacije

Prema načinu analize zavisnosti

Statički RTS (npr. STARTS)
Analizira izvorni kod ili kompajlirani bajtkod bez pokretanja programa. Nema runtime usporenja, ali daje širi skup testova.

Dinamički RTS (npr. Ekstazi, JaCoCo RTS)
Beleži zavisnosti tokom stvarnog izvršavanja testova (instrumentacijom). Visoka preciznost, rukuje refleksijom i fajlovima.

Hibridni RTS (npr. HyRTS)
Kombinuje brzu statičku analizu za sužavanje prostora sa dinamičkim podacima za veću preciznost.

Prema nivou granulacije

Nivo fajla / modula (Coarse-grained)
Najbrža analiza, ali selektuje veliki broj nepotrebnih testova pri malim izmenama.

Nivo klase — Class-level (STARTS, Ekstazi)
Optimalan balans brzine analize i veličine selektovanog skupa u praksi.

Nivo metode / linije (JaCoCo RTS)
Najveća preciznost (ignoriše izmene u nepozvanim metodama iste klase), ali najviša cena snimanja tragova.

3. Parametri ocenjivanja kvaliteta RTS tehnika

Standardni kriterijumi (Rothermel & Harrold) za vrednovanje algoritama selekcije

Inkluzivnost (Inclusiveness / Safety)

• Mera u kojoj tehnika uspeva da izabere sve testove koji mogu da otkriju grešku izazvanu izmenom. Tehnika je bezbedna (Safe) ako garantuje 100% inkluzivnosti.

Preciznost (Precision)

• Sposobnost tehnike da izostavi testove koji nisu pogođeni izmenom. Veća preciznost znači manji broj nepotrebno pokrenutih (lažno pozitivnih) testova.

Efikasnost (Efficiency)

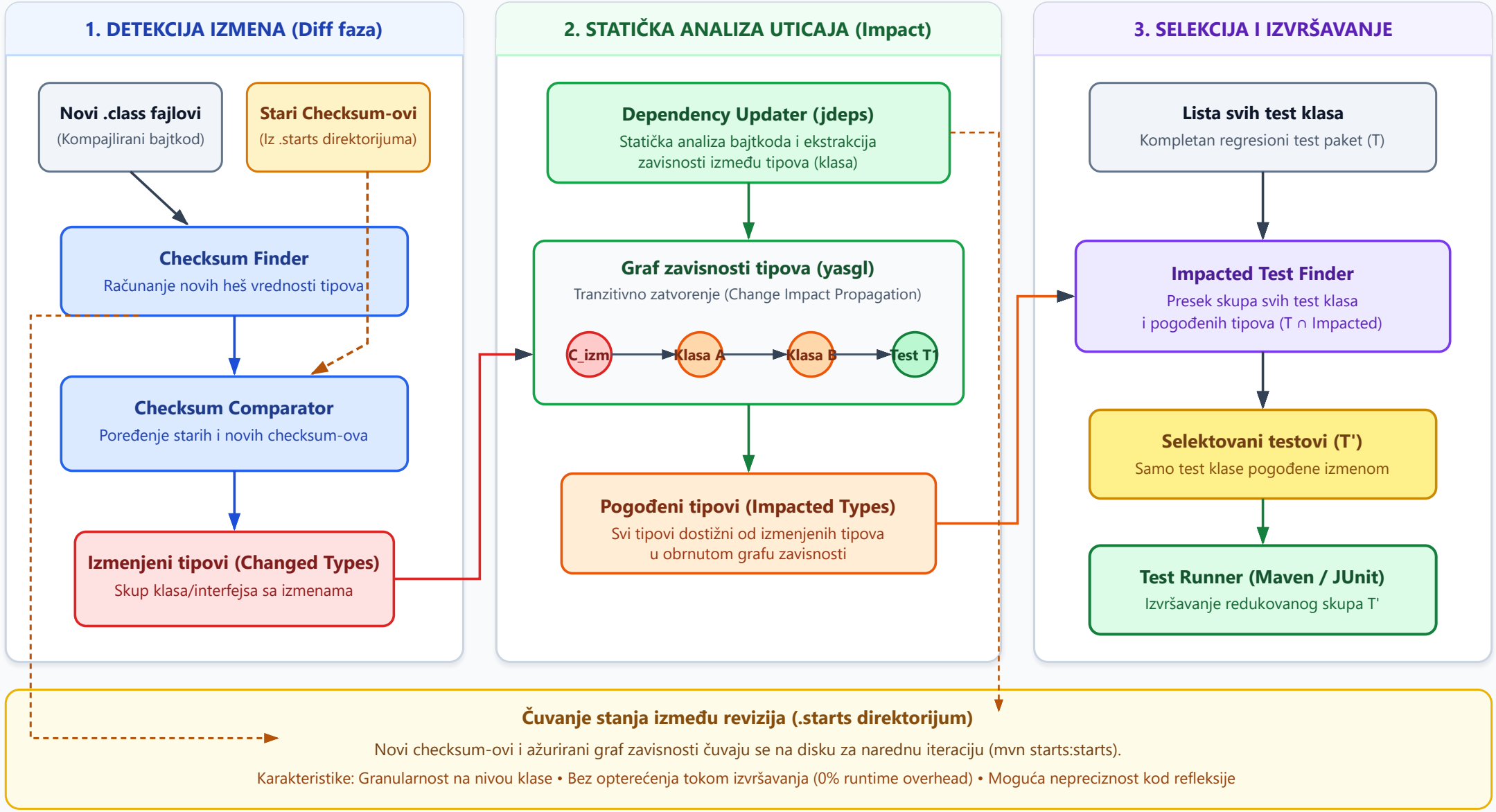
• Ukupna vremenska i prostorna cena analize. RTS je isplativ samo ako važi:
T_analiza + T_selektovani_testovi < T_svi_testovi

Opštost (Generality)

• Sposobnost pristupa da radi sa realnim programskim konstrukcijama: nasleđivanjem, polimorfizmom, refleksijom, višestrukim nitima i eksternim fajlovima.

STARTS – Statička selekcija regresionih testova (analiza uticaja izmena)

STAtic Regression Test Selection • Analiza uticaja izmena na nivou tipova (klasa) bez instrumentacije izvršavanja

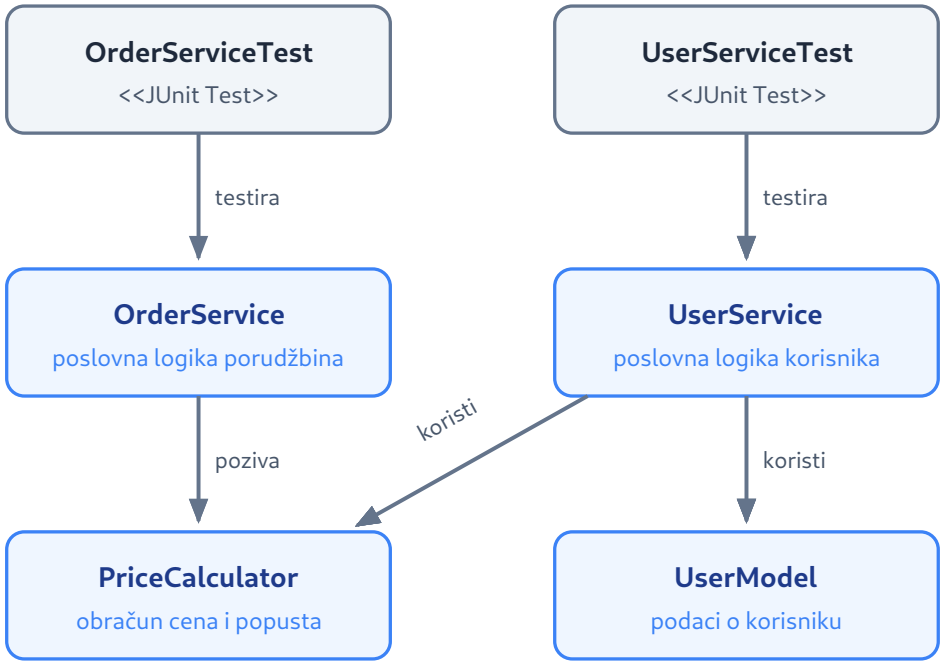


STARTS Primer: Graf zavisnosti i tranzitivno propagiranje uticaja izmena

Ilustracija na primeru tipičnog Java projekta (Klase → Zavisnosti → Selekcija testova)

1. Izgradnja grafa zavisnosti (jdeps + yasgl)

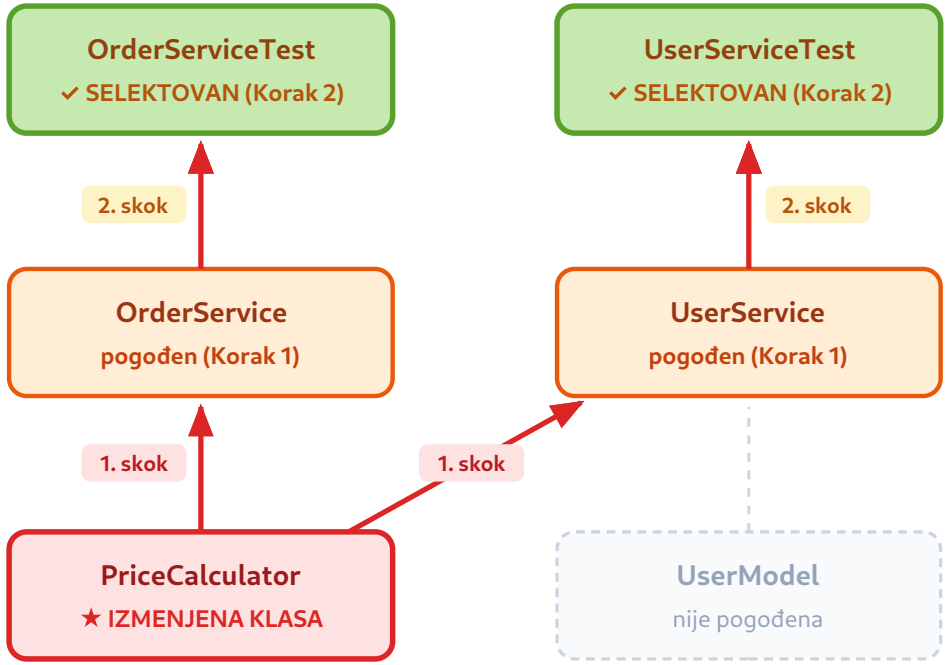
Strelica **A → B** znači: „Klasa A zavisi od klase B“ (koristi/poziva B)



Graf se pravi statički iz bajtkoda (.class), bez pokretanja testova.

2. Tranzitivno propagiranje uticaja izmena (BFS/DFS unazad)

Izmenjena je samo klasa **PriceCalculator** → uticaj se širi suprotno od zavisnosti



Oba testa su pogođena indirektno (tranzitivno) i oba se izvršavaju.

Direktno izmenjena klasa | Tranzitivno pogođena klasa | Selektovani testovi (izvršavaju se) | Smer širenja uticaja (obrnuto od zavisnosti)

STARTS: Analiza uticaja izmena (change impact analysis)

Statička analiza uticaja izmena

1. Izgradnja grafa zavisnosti (Dependency Graph)

STARTS ne pokreće program da bi video šta se dešava, već analizira kompajlirane .class fajlove (bajtkod) i pravi usmereni graf celog projekta.

- **Čvorovi grafa** su sve klase i interfejsi u projektu (uključujući i test klase).
- **Usmerene grane** (strelice) pokazuju ko od koga zavisi na osnovu poziva metoda, pristupa poljima, nasleđivanja ili tipova promenljivih.
- **Značenje strelice A → B**: Klasa A direktno koristi ili poziva klasu B (tj. A zavisi od B).

2. Tranzitivno propagiranje uticaja izmena kroz graf

- Tranzitivnost znači da se uticaj prenosi lančano — ako A zavisi od B, a B zavisi od C, onda izmena u C utiče i na A, iako A direktno ne pominje C.
- **Polazna tačka**: STARTS upoređuje checksum-ove (otiske) .class fajlova sa prethodnom verzijom i pronalazi direktno izmenjene klase.
- **Kretanje unazad** (Reverse Reachability): Od svake izmenjene klase, algoritam pretrage (BFS/DFS) kreće se suprotno od smera strelica zavisnosti kako bi pronašao sve klase koje direktno ili indirektno zavise od nje.
- **Rezultat**: Sve klase do kojih se stigne ovim putem označavaju se kao pogođene, a sve test klase među njima ulaze u skup selektovanih testova koji moraju ponovo da se pokrenu.

STARTS: Šta prikazuje priloženi primer?

Ponašanje pri izmeni PriceCalculator klase

Levi deo (Graf zavisnosti)

Prikazuje odnose u kodu

- OrderServiceTest testira OrderService, koji poziva PriceCalculator.
- Takođe, UserService koristi i UserModel i PriceCalculator

Desni deo (Propagiranje uticaja):

- Pretpostavimo da ste izmenili samo klasu PriceCalculator.
- Uticaj se u 1. skoku širi na OrderService i UserService (jer oba koriste PriceCalculator), a
- u 2. skoku stiže do OrderServiceTest i UserServiceTest.
- Pošto nijedan test direktno ne poziva PriceCalculator, bez tranzitivnog propagiranja nijedan test ne bi bio pokrenut.

5. Praktični primer: starts-demo (Maven)

Konfiguracija starts-maven-plugin-a i ponašanje pri izmeni PriceCalculator klase

Konfiguracija u pom.xml

```
<plugin>
  <groupId>edu.illinois</groupId>
  <artifactId>starts-maven-plugin</artifactId>
  <version>1.4</version>
</plugin>
```

Pokretanje u terminalu

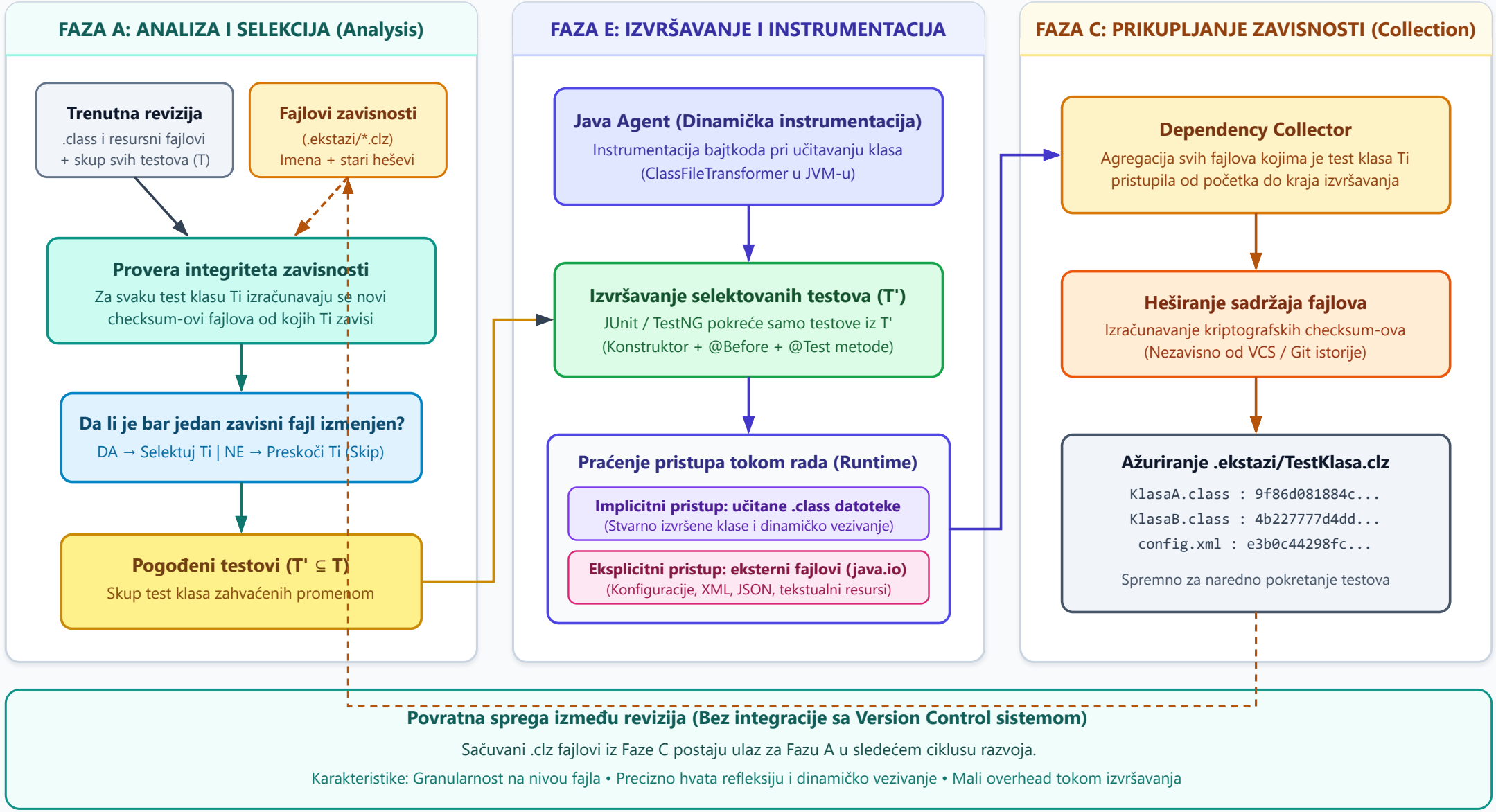
- mvn starts:starts (1. put): Pokreće oba testa i kreira .starts/ direktorijum sa grafom.
- mvn starts:starts (bez izmena): Tests run: 0.
- Izmena PriceCalculator.java → mvn starts:starts: Pokreće oba testa (OrderServiceTest i UserServiceTest).

Zašto STARTS pokreće i UserServiceTest?

- U klasi UserService postoji metoda getMembershipFee() koja poziva PriceCalculator.
- Iako UserServiceTest testira samo getUserSummary() i nikada ne izvršava obračun cene, u kompajliranom bajtkodu UserService.class postoji referenca na PriceCalculator.
- Pošto STARTS ne pokreće kod da bi video koje se grane izvršavaju, on konzervativno pretpostavlja da izmena u PriceCalculator može uticati na ceo UserService.
- Rezultat: 100% bezbednost (Safety), ali niža preciznost.

Ekstazi – Dinamička selekcija regresionih testova (Change Impact Analysis)

Dynamic File-Level RTS • Praćenje stvarno pristupljenih fajlova tokom izvršavanja testova (Analysis → Execution → Collection)



STARTS - Statička selekcija regresionih testova

EKSTAZI

- Realizuje dinamički pristup podeljen u tri faze (**Analysis → Execution → Collection**),
- gde se pomoću JVM bajtkod instrumentacije prate svi
  - implicitno učitani .class fajlovi i
  - eksplicitno otvoreni resursi tokom izvršavanja testova,
- a zavisnosti se čuvaju u .clz fajlovima po test klasi za narednu reviziju.

6. Ekstazi: Dinamička selekcija regresionih testova

Primer dinamičke selekcije testova kroz dve uzastopne revizije koda (Revizija N → Revizija N+1)

1. REVIZIJA N: Dinamičko praćenje u JVM-u i čuvanje .clz fajlova

JVM + Ekstazi Java Agent (Bajtkod instrumentacija u letu)

OrderServiceTest
Pokrenut test (Thread 1)

UserServiceTest
Pokrenut test (Thread 2)

[Sonda] Presreće učitane klase, pozive metoda i čitanje fajlova (I/O)

OrderService
CRC32: 0xA1B2

PriceCalculator
CRC32: 0x99F1

UserService
CRC32: 0x44C0

Disk: Direktorijum .ekstazi/ (Sačuvane zavisnosti po testu)

OrderServiceTest.clz

```
OrderServiceTest : 0x11D4
OrderService : 0xA1B2
PriceCalculator : 0x99F1
tax-rules.json : 0x77E2
```

UserServiceTest.clz

```
UserServiceTest : 0x88C1
UserService : 0x44C0
UserModel : 0x52B9
```

(Nije koristio PriceCalculator)

2. REVIZIJA N+1: Izmena koda i poređenje CRC32 otisaka

Korak A: Programer menja samo klasu PriceCalculator

PriceCalculator.class → Novi CRC32: 0x3F8E (bilo je 0x99F1)
Sve ostale klase i fajlovi imaju nepromenjen CRC32 heš!

Korak B: Ekstazi čita .clz fajlove i proverava svaku stavku

Provera OrderServiceTest.clz

```
OrderService : OK
PriceCalculator : RAZLIKA
(0x99F1 ≠ 0x3F8E)
tax-rules.json : OK
```

POKREĆE SE PONOVO
Ažurira svoj .clz fajl

Provera UserServiceTest.clz

```
UserServiceTest : OK
```

PRESKAČE SE.
Svi heševi u .clz identični.

Kako Ekstazi funkcioniše na konkretnom primeru?

- **Bajtkod instrumentacija (Java Agent)**: Prilikom pokretanja JVM-a, Ekstazi ubacuje lagani kod (sonde) u učitane klase i sistem za učitavanje fajlova. Kad god test pozove neku metodu, pristupi statičkom polju ili otvori konfiguracioni fajl, sonda beleži taj događaj.

- **Kreiranje .clz fajlova (Collection faza)**: Za svaku pokrenutu test klasu, Ekstazi u direktorijumu `.ekstazi/` kreira poseban fajl sa ekstenzijom `.clz` (npr. `OrderServiceTest.clz`). U njemu su upisani tačni nazivi klasa/resursa koji su zaista korišćeni u izvršavanju i njihovi **CRC32 heševi** (otisci).

- **Poređenje otisaka u sledećoj reviziji (Analysis faza)**: Kada programer izmeni kod i ponovo pokrene testove, Ekstazi prvo čita postojeće `.clz` fajlove i upoređuje stare CRC32 heševe sa trenutnim stanjem `.class` fajlova na disku.

- **Preciznija selekcija (manje lažnih alarma)**: Ako je izmenjena metoda u klasi `PriceCalculator` koju poziva samo `OrderServiceTest`, a `UserServiceTest` tokom svog izvršavanja uopšte nije ušao u taj deo koda (ili je koristio samo `UserModel`), Ekstazi će pokrenuti samo `OrderServiceTest`.

7. Praktični primer: ekstazi-demo (Maven)

Konfiguracija ekstazi-maven-plugin-a i praćenje eksternih resursa

Konfiguracija u pom.xml

```
<plugin>
  <groupId>org.ekstazi</groupId>
  <artifactId>ekstazi-maven-plugin</artifactId>
  <version>5.3.0</version>
  <executions><execution>
    <goals><goal>select</goal></goals>
  </execution></executions>
</plugin>
```

Eksperimenti sa mvn test

- Izmena PriceCalculator.java → Pokreće samo OrderServiceTest (Tests run: 1).
- Izmena tax-rules.json (PDV 0.20 → 0.25) → Pokreće samo OrderServiceTest i hvata grešku!

Ključne prednosti Ekstazi pristupa

- Veća preciznost od STARTS-a: Ignoriše statičke veze u kodu koje se tokom testa nikada nisu izvršile.
- Praćenje I/O resursa: Instrumentacijom klasa za rad sa fajlovima beleži i konfiguracione fajlove (JSON, XML, properties).
- Podrška za refleksiju: Beleži klase učitane preko Class.forName(...) jer ih JVM učitava u toku izvršavanja testa.
- Cena: Blago usporenje izvršavanja selektovanih testova zbog rada JVM agenta.

8. JaCoCo RTS: Dinamička selekcija na nivou linija / metoda

Kombinacija pokrivenosti koda za test(JaCoCo Agent) i analize razlika (git diff)

1. Faza snimanja: Pokrivenost po svakom testu

JUnit RunListener + JaCoCo Runtime API
Pre svakog testa radi agent.reset(), a nakon testa agent.getExecutionData() snima izvršene linije.

dump

dump

OrderServiceTest.cov
OrderService: linije 10-12
PriceCalculator: linije 8-12
(Linije 15-18 nisu izvršene)

UserServiceTest.cov
UserService: linije 9-11
UserModel: linije 14-21
(PriceCalculator: 0 linija)

• Granulacija na nivou linije ili metode — najfinija među svim prikazanim pristupima.

2. Faza selekcije: Presek izmena i matrice

Presek: git diff ∩ Per-Test pokrivenost
Izmenjena lin 10 u PriceCalculator.java (popust 0.90 → 0.85).
(Da je izmenjena nepokrivena linija 16, pokrenulo bi se 0 testova!)

pokriva lin. 10

ne pokriva

OrderServiceTest
SELEKTOVAN
Pokriva liniju 10.
Ponovo se izvršava i osvežava .cov trag.

UserServiceTest
PRESKOČEN
Nije izvršavao izmenjeni kod.

• Mana: Visoko opterećenje (overhead) pri snimanju jer se sonde resetuju za svaki test.

JaCoCo RTS: Dinamička selekcija na nivou linija / metoda

Kombinacija pokrivenosti koda za test(JaCoCo Agent) i analize razlika (git diff)

Faza snimanja: Pokrivenost na nivou pojedinačnog testa

- **Izolovano praćenje svakog testa**: Umesto jednog zajedničkog izveštaja za ceo projekat, koristi se `JUnit RunListener` u kombinaciji sa JaCoCo Runtime API-jem. Pre pokretanja svakog testa sonde se resetuju (`agent.reset()`), a odmah po završetku testa preuzimaju se podaci o izvršavanju (`agent.getExecutionData()`).

- **Generisanje .cov mapa**: Za svaki test se na disku čuva tačan spisak klasa i brojeva linija koje su se tokom tog testa izvršile. Na primer, `OrderServiceTest.cov` beleži linije 8–12 klase `PriceCalculator`, dok `UserServiceTest.cov` uopšte nema zabeleženu tu klasu.

Faza selekcije: Presek izmena i matrice (Desni deo slajda)

- **Pronalaženje izmenjenih linija**: Nakon izmene u kodu, pomoću alata kao što je git diff identifikuje se tačna lokacija promene (ovde je to **linija 10** u klasi `PriceCalculator.java`).

- **Presek sa bazom pokrivenosti**: Sistem upoređuje izmenjene linije sa svim snimljenim `.cov` fajlovima i pokreće samo one testove u čijem se tragu nalazi ta linija.

  - **OrderServiceTest** se pokreće jer njegov `.cov` zapis obuhvata liniju `10`.

  - **UserServiceTest** se preskače jer nije prolazio kroz izmenjeni kod.

9. Praktični primer: jacoco-rts-demo (Maven + JUnit Listener)

Implementacija izolovanog prikupljanja pokrivenosti pomoću JaCoCo Runtime API-ja

JaCoCoPerTestListener.java (Ključni deo)

```
public void testStarted(Description desc) {
  RT.getAgent().reset(); // Reset sondi
}
public void testFinished(Description desc) {
  byte[] data = RT.getAgent().getExecutionData(false);
  // Analiza pokrivenih linija pomoću
  // CoverageBuilder i Analyzer klasa
  // Upis u .jacoco-rts/<Test>.cov
}
```

Primer sa dve metode u istoj klasi

- PriceCalculator ima dve metode: calculateFinalPrice() (linije 6–11) i calculateShipping() (linije 14–16).
- OrderServiceTest poziva samo prvu metodu, pa njegov .cov fajl beleži isključivo linije 6–11.
- Ako programer izmeni liniju 15 (calculateShipping), menja se CRC32 cele klase PriceCalculator.class.
- STARTS i Ekstazi bi pokrenuli OrderServiceTest, dok JaCoCo RTS vidi da linija 15 nije pokrivena i pokreće 0 testova!

10. Uporedni pregled alata i zaključak

Izbor optimalnog RTS pristupa u zavisnosti od arhitekture i potreba projekta

Kriterijum	STARTS	Ekstazi	JaCoCo RTS
Tip analize	Statička (jdeps + yasgl)	Dinamička (JVM agent)	Dinamička (Per-Test Coverage)
Nivo granulacije	Klasa (Class-level)	Klasa (Class-level)	Linija / Metoda
Runtime opterećenje	Nema (0% overhead)	Nisko do umereno	Visoko (reset po testu)
Preciznost selekcije	Umerena (konzervativna)	Visoka	Najviša
Eksterni fajlovi i refleksija	Ograničena podrška	Odlična podrška (.clz)	Zahteva dodatnu obradu

Ključne napomene iz prakse

- Selekcija (RTS) vs. Prioritizacija (TCP): RTS odbacuje nepogođene testove radi uštede vremena, dok TCP menja samo redosled izvršavanja svih testova (korisno pred veliko izdanje).
- Flaky testovi: Nijedan RTS algoritam ne može garantovati bezbednost ako testovi zavise od slučajnih brojeva, sistemskog vremena ili spoljnih mreža van kontrole JVM-a.