2. Definicije i temeljna pitanja programskog inženjerstva

Programska potpora (engl. software) ili računalni program je neopipljiva komponenta računala koja uključuje sve podkomponente nužne za uspješno izvođenje računalnih instrukcija. Programska potpora može uključivati izvršne datoteke, skripte, knjižnice i druge podkomponente.

Programski proizvod (engl. application) ili programski sustav (engl. software system) sastoji se od jednog ili više računalnih programa. On se može izraditi oblikovanjem novih računalnih programa, (re)konfiguracijom postojećih programa, ili ponovnom uporabom postojećih komponenata programa.

Dokumentacija je pisani tekst u obliku jednog ili više dokumenata koji objašanjava kako radi programska potpora, koji su zahtjevi na rad programske potpore ili kako se ona treba pravilno koristiti. Dokumentacija programske potpore ima za cilj pružiti razne informacije o programskoj potpori raznim ljudima, u ovisnosti o ulozi koju oni imaju u odnosu na programski sustav.

Programsko inženjerstvo (engl. software engineering) je tehnička disciplina koja se bavi metodama i alatima za profesionalno oblikovanje i proizvodnju programske potpore uzimajući u obzir cjenovnu efikasnost.

Potrebno je naglasiti da programsko inženjerstvo nije računarska znanost (engl. computer science). **Računarska znanost** bavi se temeljima koji su nužni da bi se suštinski razumjeli problemi s kojima se programski inženjeri susreću u svakodnevnom radu, dok je programsko inženjerstvo orijentirano na praksu i rješavanje problema klijenata.

Područje blisko području programskog inženjerstva je inženjerstvo računalnih sustava (engl. computer system engineering) ili kraće, računalno inženjerstvo (engl. computer engineering).

Računalno inženjerstvo je širi pojam od programskog inženjerstva budući da se to područje bavi svim aspektima sustava zasnovanima na računalima (inženjerstvo sklopovlja, programsko inženjerstvo i inženjerstvo procesa). Programsko inženjerstvo, kao dio inženjerstva računalnih sustava, usredotočeno je na razvoj programske infrastrukture i na upravljanje, primjenu i rukovanjem podacima u računalnom sustavu.

Proces programskog inženjerstva je skup aktivnosti čiji cilj je razvoj i evolucija programskog proizvoda. U tom procesu bitnu komponentu čini timski rad pomoću kojeg se učinkovito izgrađuje programski proizvod. Proces programskog inženjerstva je stvarni tijek aktivnosti koji se odvija tijekom izrade programskog proizvoda.
Svaki proces programskog inženjerstva sastoji se od četiri temeljne generičke aktivnosti:

- specifikacije,
- oblikovanja i implementacije,
- validacije i verifikacije,
- evolucije programske potpore.

Model procesa programskog inženjerstva je pojednostavljeno predstavljanje procesa programskog inženjerstva iz određene perspektive (pogleda na proces). Postoji više predloženih modela koji nastoje jasnije predstaviti stvarni proces programskog inženjerstva: vodopadni, evolucijski, komponentni, modelno-usmjereni razvoj, agilni razvoj.

CASE su programski proizvodi ili alati veće ili manje složenosti namijenjeni automatiziranoj podršci generičkim aktivnostima u procesu programskog inženjerstva.
CASE proizvodi često podupiru samo jednu, određenu aktivnost u razvoju programskog proizvoda. Pri tome razlikujemo više CASE proizvode (engl. upper CASE), koji podupiru rane aktivnosti kao što su analiza zahtjeva i oblikovanje arhitekture i niže CASE (engl. lower CASE) proizvode, koji podupiru kasnije aktivnosti kao što su kodiranje i ispitivanje.

Temeljna značajka dobrog programskog proizvoda je da proizvod mora osigurati traženu funkcionalnost i performanse. Osim toga, ostale značajke su: prihvatljivost za korisnika, pouzdanost i mogućnost laganog održavanja.

3. Životni ciklus programske potpore

Svaka programska potpora ima svoj životni ciklus.

Prema Sommervilleu, to su

aktivnosti: 1) specifikacije, 2) oblikovanja i implementacije (ili kraće: razvoja), 3) validacije i verifikacije i 4) evolucije programske potpore.

Validacija odgovara na pitanje: "Gradimo li pravi sustav?", dok **verifikacija** odgovara na pitanje: "Gradimo li sustav na ispravan način?".

Često se ispitivanje prihvatljivosti naziva i **alfa ispitivanje**.

Često se ispitivanje instalacije naziva i **beta ispitivanje**.

Linije proizvoda (ili porodica proizvoda) su skupovi svih proizvoda izrađenih na zajedničkoj osnovnoj tehnologiji. Različiti proizvodi u liniji proizvoda imaju različite ostvarene značajke kako bi zadovoljili različite segmente tržišta. Tako, naprimjer, proizvodne linije nekog programskog proizvoda mogu nositi oznaku “demo”, “pro”, “lite”, “enterprise” i sl. Inačice su vezane uz svaku od tih proizvodnih linija pri čemu nije nužno da inačice budu jednake između različitih proizvodnih linija.

4. Inženjerstvo zahtjeva

Inženjerstvo zahtjeva je proces izrade specifikacije programske potpore. To je prva generička aktivnost tijekom svakog procesa programskog inženjerstva. Zahtjevi opisuju što programski sustav treba raditi kao i ograničenja u njegovom radu. U procesu izrade specifikacije programske potpore postoje postupci pronalaženja, analiziranja, dokumentiranja i provjere funkcija i ograničenja u uporabi. Zahtjevi sami za sebe su kraći ili duži dokumenti koji specificiraju usluge sustava i ograničenja u uporabi.

Mnogi projekti zapadnu u probleme kad se razine zahtjeva međusobno miješaju. Upravo stoga je vrlo bitno **klasificirati zahtjeve** prema razini detalja i prema sadržaju. Tako se prema razini detalja razlikuju:

- **Korisničke zahtjeve (engl. user requirements)**
- **Zahtjeve sustava (engl. system requirements)**
- **Specifikaciju programske potpore (engl. software specification)**

Korisnički zahtjevi su oni zahtjevi najviše razine apstrakcije.

Tako se oni pišu u prirodnom jeziku i crtaju jednostavnim grafičkim dijagramima. Često iz njih nije jasno kako će točno sustav

funkcionirati niti koji će biti njegovi dijelovi.

Zahtjevi sustava su vrlo detaljna specifikacija o funkcionalnosti i ograničenjima programske potpore. Pišu se strukturiranim prirodnim jezikom, posebnim jezicima za oblikovanje sustava, dijagramima i matematičkom notacijom.

Specifikacija programske potpore je najdetaljniji opis i objedinjuje korisničke zahtjeve i zahtjeve sustava. Ona također može dodatno obuhvaćati tehničku specifikaciju koja opisuje detaljne zahtjeve na arhitekturu i programske dijelove.

Prema sadržaju, zahtjeve se može podijeliti na:

• funkcionalne zahtjeve (engl. functional requirements),

Funkcionalni zahtjevi su izjave o uslugama koje programski proizvod mora pružati, kako će sustav reagirati na određeni ulazni poticaj, te kako bi se trebao ponašati u određenim situacijama. U nekim slučajevima, funkcionalni zahtjevi trebaju eksplicitno definirati i što sustav ne treba raditi. Funkcionalni zahjevi su kompletni ako sadrže opise svih zahtijevanih mogućnosti, dok su konzistentni ako ne sadržavaju konflikte ili proturječne tvrdnje. U praksi je nemoguće postići kompletan i konzistentan dokument o funkcionalnim zahtjevima.

• Nefunkcionalne ili ostale zahtjeve (engl. non-functional, other requirements),

Nefunkcionalni zahtjevi su ograničenja u uslugama i funkcijama programskog proizvoda.

Najgrublja podjela je na tri tipa:

1) zahtjeve programskog proizvoda, koji specificiraju na koji se osobiti način treba ponašati isporučeni proizvod, npr. da ima odgovarajuće vrijeme odziva, da radi na operacijskom sustavu Windows 7, da koristi HTTPS protokol;

2) organizacijske zahtjeve, koji su rezultat organizacijskih pravila i procedura, npr. uporaba propisanog normiranog procesa razvoja, korištenje uvijek točno određenog programskog jezika pri razvoju;

3) vanjske zahtjeve, koji proizlaze izvan sustava i razvojnog procesa, npr. postizanje međusobne operabilnosti s drugim sustavima, zakonske zahtjeve i drugo.

• Zahtjeve domene primjene (engl. domain requirements).

Studija izvedivosti je kratka, fokusirana studija na početku procesa inženjerstva zahtjeva kojom se utvrđuje isplati li se predloženi sustav (tj. je li vrijedan uloženih sredstava)

Pogledi (engl. viewpoint) su način strukturiranja zahtjeva tako da oslikavaju perspektivu i fokus različitih grupa dionika. Svaki pogled uključuje skup zahtjeva sustava.

Razlikuju se:

• **pogledi interakcije** - ljudi i drugi dionici koji izravno komuniciraju sa sustavom,

• **neizravni pogledi** - dionici koji utječu na zahtjeve, ali ne koriste sustav izravno,

• **pogledi domene primjene** - karakteristike domene i ograničenja na sustav.

Etnografija je tehnika opažanja koja se koristi kako bi se bolje razumjeli procesi kod klijenta i kako bi se pomoglo otkriti što je moguće više ispravnih i korisnih zahtjeva. Etnografija podrazumijeva dolazak jednog ili više ljudi iz razvojnog tima u tvrtku gdje će se sustav primjenjivati i uključivanje tih inženjera (tzv. etnografa) u svakodnevne aktivnosti u tom okruženju

Validacija zahtjeva je proces provjere da li zahtjevi koje su dobiveni od klijenta zaista definiraju sustav koji korisnik želi.

Tehnike validacije uključuju:

• **Recenziju zahtjeva** - detaljna, ručna analiza zahtjeva od strane zajedničkog tima.

• **Izradu prototipa** - provjera zahtjeva na izvedenom sustavu.

• **Generiranje ispitnih slučaja** - razvoj ispitnih sekvenci za provjeru zahtjeva.

Moguće **je klasificirati vrste promjena zahtjeva** u sljedeće četiri kategorije:

• **Okolinom promijenjeni zahtjevi** - promjena zahtjeva zbog promjene okoline u kojoj organizacija posluje (npr. bolnica mijenja financijski model pokrivanja usluga).

• **Novonastali zahtjevi** - zahtjevi koji se pojavljuju kako kupac sve bolje razumije sustav koji se oblikuje.

• **Posljedični zahtjevi** - zahtjevi koji nastaju nakon uvođenja sustava u eksploataciju, a rezultat su promjena procesa rada u organizaciji nastalih upravo uvođenjem novoga sustava.

• **Zahtjevi kompatibilnosti** - zahtjevi koji ovise o procesima drugih sustava u organizaciji; ako se ti sustavi mijenjaju to traži promjenu zahtjeva i na novo uvedenom sustavu.

5. Procesi i modeli procesa programskog inženjerstva

Modeli procesa programskog inženjerstva predstavljaju uočene pristupe organizaciji projekta izrade programske potpore.

To su:

• **Vodopadni model** (engl. waterfall model) koji temeljne aktivnosti procesa izrade programske potpore gleda kao nezavisne faze razvoja.

• **Evolucijski model** (engl. evolutionary model) kod kojeg se sustav razvija kroz niz inačica ili inkremenata kod kojih svaka sljedeća dodaje neku novu funkcionalnost u onu na koju se nastavlja.

Dva su osnovna tipa evolucijskog modela razvoja programske potpore:

• Istraživački razvoj i oblikovanje (engl. exploratory development). Kod ovog pristupa prisutan je stalan rad s naručiteljem radi identifikacije pravih zahtjeva. Razvoj započinje u dijelovima sustava za koje su zahtjevi dovoljno razumljivi, a nakon toga se dodaju nove funkcionalnosti prema prijedlozima naručitelja.

• Metoda odbacivanja prototipa (engl. throwaway prototyping). U ovom pristupu je cilj bolje razumijevanje zahtjeva naručitelja i na osnovi toga bolja specifikacija zahtjeva. Prototipovi koji se pritom razvijaju na temelju slabo definiranih zahtjeva služe samo kao pomoć pri razjašnjavanju stvarnih potreba korisnika.

• **Komponentni model** (engl. component-based model) čija je motivacija u pretpostavci ponovne iskoristivosti postojećih komponenata.

Osim ta tri generička modela, u novije vrijeme naglašavaju se kao zasebni modeli i modelno-usmjereni razvoj programske potpore (engl. Rational Unified Process, RUP) kao i agilni razvoj (engl. agile development). U stvarnosti, oni čine samo podmodele tri osnovna generička modela razvoja.

-Uz komponentno-usmjereni model veže se jedna čitava grana programskog inženjerstva, odnosno komponentno-usmjereno programsko inženjerstvo (engl. component-based software engineering - CBSE), koja se kao pristup razvoju programske potpore pojavila krajem 90-ih godina prošlog stoljeća s motivacijom postizanja bolje i šire primjene principa ponovne uporabljivosti (engl. reuse) programskih komponenata. Kao najjednostavnija definicija pojma komponente može se reći da je to nezavisna jedinica programske potpore koja se može kombinirati s drugim nezavisnim jedinicama u svrhu izrade cjelovitog sustava programske potpore.

S obzirom na neodvojivost pojma komponentno-usmjerenog programskog inženjerstva od ideje ponovne uporabljivosti komponente, na procese te grane programskog inženjerstva možemo gledati s dva različita stajališta, a to su:

- **Razvoj za ponovnu iskoristivost** (engl. development for reuse), gdje se grade komponente ili usluge predviđene za iscrpno korištenje u drugim razvojnim procesima. U ovom tipu razvoja, programski kôd komponente je u potpunosti otvoren, a često se proces svodi na generalizaciju postojećih komponenti za koje postoje naznake višestruke uporabljivosti.
- **Razvoj s korištenjem postojećih komponenti** (engl. development with reuse) gdje se grade nove aplikacije korištenjem postojećih komponenti. Prikladne komponente se pritom moraju najprije pronaći i identificirati kao one koje zadovoljavaju ili se mogu prilagoditi potrebama aplikacije koja se gradi. Pritom za te komponente ne mora biti dostupan programski kôd.

Modelno-usmjereni razvoj (engl. Rational Unified Process, RUP) je metodologija razvoja programske potpore koja se temelji na oblikovanju pomoću modela (engl. Model Based Design, MBD), odnosno iterativnom razvoju, obrascima uporabe i usmjerenjem na arhitekturu sustava.

Kao i svaka druga projektna metodologija, **RUP određuje faze životnog ciklusa** (engl. lifecycle) procesa i dokumente koji se moraju izraditi završetkom izvođenja svake faze. Faze životnog ciklusa RUP-a su:

1**) početak** (engl. inception), u kojemu se definira doseg projekta, razvoj modela poslovnog procesa, specifikacija početnih zahtjeva,

2) elaboracija (engl. elaboration), kada se definiraju plan projekta, specifikacija značajki i temelji arhitekture sustava,

3) izgradnja (izrada, engl. construction), koja se sastoji od oblikovanja, programiranja, i ispitivanja, te

4) prijenos proizvoda korisnicima (engl. transition), pri čemu se završni proizvod postavlja u radnu okolinu.

Završne, odgovarajuće ključne točke ovih faza su, redom:

1) vizija ili ciljevi životnog ciklusa (engl. lifecycle objectives),

2) temeljna arhitektura (engl. lifecycle arhitecture),

3) početna sposobnost (engl. initial operational capability) i

4) izdanje izvršne inačice programa ili proizvoda (engl. release).

Svaka faza ima barem jednu iteraciju. Stoga, u kontekstu RUP-a, iteracija je niz ili sekvenca aktivnosti u okviru prihvaćenog plana i kriterija evaluacije. Ishod svake iteracije je dokument, a na kraju i jedna izvršna inačica (tj. izdanje) programa ili sustava.

Osim faza životnog ciklusa, RUP definira i niz aktivnosti koje se mogu odvijati u svim fazama, ali obično s različitim intenzitetom.

Jezgrene aktivnosti (engl. core workflows) RUP-a su:

1) modeliranje poslovnog procesa (engl. business modelling),

2) zahtjevi (engl. requirements),

3) analiza i oblikovanje (engl. analysis and design),

4) implementacija (engl. implementation),

Arhitektura programske potpore je struktura ili strukture sustava koje sadrži elemente, njihova izvana vidljiva obilježja i odnose između njih.

Agilni pristup razvoju programske potpore podrazumijeva skupinu
metoda za razvoj programske potpore kojima je zajednički iterativni razvoj uz male inkremente i brz odziv na korisničke zahtjeve. Ovaj model razvoja programske potpore koristi se za razvoj manjih i srednjih projekata u stalnoj interakciji s klijentima putem stalnog predočavanja novih poboljšanja, uz relativno slabo dokumentiranje

Često se može čuti kako određeni dokument ili izvršna datoteka prolazi kroz **iteracije**. Time se želi reći da neki dokument ima više inačica i da mu se u svakom koraku dodaje novi sadržaj, ili otklanjaju pogreške.

Postoje dva međusobno ovisna pristupa iteracijama:

1. Inkrementalni

2. Spiralni

U inkrementalnom pristupu iteracijama (slika 5.2), sustav se ne isporučuje korisniku u cjelini. Aktivnosti razvoja, oblikovanja i isporuke razlažu se u inkrementalne dijelove koji predstavljaju djelomične funkcionalnosti proizvoda. Zahtjevi korisnika se analiziraju i organiziraju u prioritetne cjeline. Dijelovi višega prioriteta isporučuju se u ranim fazama razvoja sustava (i početnim inkrementima). S početkom razvoja pojedinog inkrementa njegovi zahtjevi se fiksiraju („zamrzavaju“). Zahtjevi na ostale kasnije inkremente nastavljaju evoluirati i prilagođavati se željama korisnika i mogućnostima implementacije.

Kod spiralnog pristupa (slika 5.3) iteracijama, proces oblikovanja predstavljan je oblikom spirale umjesto nizom aktivnosti. Svaka petlja u spirali predstavlja jednu fazu procesa. U svakoj fazi razvoja, eksplicitno se određuju i razrješavaju rizici razvoja programskog proizvoda.

6. Alati i okruženja za potporu razvoja programa

U procesu razvoja programske potpore inženjeri koriste posebne alate kako bi automatizirali dio aktivnosti i samim time ubrzali izradu, pribavili korisne informacije o proizvodu koji se razvija te olakšali kasnije održavanje proizvoda. To su tzv. CASE-alati. CASE-alati su programski proizvodi koji podupiru proces programskog inženjerstva, a posebice aktivnosti specifikacije, oblikovanja, implementacije i evolucije.

Pri klasifikaciji CASE-alata koriste se tri različite perspektive:

1. Funkcionalna perspektiva – alati se klasificiraju prema specifičnoj funkciji koju obavljaju.
2. Procesna perspektiva – alati se klasificiraju prema aktivnostima koje podupiru u procesu.
3. Integracijska perspektiva – alati se klasificiraju prema njihovoj organizaciji u integrirane cjeline.

Integracijska perspektiva promatra alate s obzirom na stupanj integracije alata u cjelinu. Alati (u užem smislu) podupiru individualne zadatke u procesu (npr. oblikovanje, provjeru konzistencije zahtjeva, uređivanje teksta, …).

Često se spominju dvije kategorije alata:

- **Upper-CASE alati (front-end),** koji se koriste u ranijim fazama kao što su izlučivanje zahtjeva, analize i modeliranja
- **Lower-CASE alati (back-end),** koji se koriste u kasnijim fazama implementacije, ispitivanja i održavanja.

Radne klupe (engl. workbenches) podupiru pojedine aktivnosti (faze) procesa (npr. specifikaciju). One objedinjavaju više različitih alata za potporu u nekoj fazi procesa programskog inženjerstva. Najčešće podržavaju jednu od tri aktivnosti: analizu i oblikovanje, programiranje te ispitivanje

U programskom inženjerstvu, kontrola inačica programske potpore je
svaki postupak koji prati i omogućava upravljanje promjenama nastalima u datotekama s izvornim kodom ili dokumentacijom

Apache Subversion (SVN) je sustav za kontrolu inačica programske potpore distribuiran pod Apache licencom. Ovaj sustav nasljednik je ranijeg CVS sustava, a najznačajnija unaprjeđenja su u vidu potpune atomarnosti operacije commit, zaključavanja datoteka u slučaju da više korisnika pokuša istovremeno raditi izmjene, dohvaćanje read-only inačice središnjeg repozitorija i dr. Na slici 6.4 prikazana je arhitektura SVN-sustava, a dijagram na slici 6.5 prikazuje shemu rada sa SVN-sustavom.
Prilikom korištenja SVN-sustava, dvije najčešće korištene operacije su Update koja mijenja lokalnu radnu kopiju kako bi bila identična stanju središnjeg repozitorija te operacija Commit koja promjene nastale u lokalnoj radnoj kopiji izvozi u središnji repozitorij. Prilikom stvaranja lokalne radne kopije (prvo dohvaćanje sadržaja središnjeg repozitorija) koristi se operacija Checkout.

Git je danas vjerojatno najrašireniji sustav otvorenog koda za kontrolu inačica programske potpore. U odnosu na SVN znatno je složeniji za savladavanje i korištenje, no zato nudi dodatne funkcionalnosti, a njegova raspodijeljena arhitektura omogućava istovremeno postojanje više smjerova razvoja programske potpore koji se mogu, ali i ne moraju objediniti.