3. DIZAJNIRANJE PROGRAMA

Ispravno dizajniranje programa je presudan element za uspješnost programa. Osnova za to je praksa analiziranja problema. Na osnovu te analize slijedi dizajn programa. Važno je pokazati kroz šta mora da prođe programer prije pisanja programa.

3.1 Obrazloženje potrebe za dizajnom programa

Program treba dizajnirati prije nego što se isti počne pisati. Ne bi trebalo odmah sjedati za tastaturu i početi ukucavati instrukcije koda u editor, baš kao što ni građevinar ne uzima odmah alat u ruke prije nego što su planovi za kuću gotovi.

Savjet: Nikad nije malo vremena koje treba utrošiti u fazi dizajna programa. Izmjene i dodavanje u programu tokom izrade programa, znaju prilično zakomplikovati i odužiti cijeli posao.

3.2 Definisanje zahtjeva za program

Detaljan dogovor između programera i korisnika neophodan je i posebno je važan za sve oblasti programiranja. Potrebno je napraviti detaljnu specifikaciju zahtjeva za program, odnosno svih elemenata koje program treba da ima, i istu ovjeriti i potpisima svih onih koji su ispostavili zahtjeve i onih koji će se koristiti realizovanim programu.

3.3 Održavanje programa

Nakon što je program napisan, testiran, i distribuiran korisnicima, održavanje programa postaje stalnom i posebno važnom obavezom koja iziskuje dosta vremena. Programi se stalno nadograđuju, kako bi ispunili nove potrebe korisnika.

3.4 Koraci dizajna programa

Postoje tri fundamentalna koraka koji treba da se izvrše kada se kreira program:

1. definiranje izlaza i toka podataka,
2. razvijanje logike da bi se došlo do izlaza i
3. pisanje programa.

1. korak: Definiranje izlaza i toka podataka

Potrebno je imati jasnu ideju o tome šta program treba da uradi i koji podaci su potrebni da bi se to realizovalo. Programer treba da zna kakav će biti izlaz prije pisanja programa, kako treba da izgleda svaki ekran i svaka stranica svih štampanih izvještaja. Potrebno je definisati sve korisničke ekrane za unos podataka, sa detaljnom listom svih polja (tip, dužina) koje program za unos treba da prikaže na ekranu. Tu su i komandna dugmad prozora i linije za skrolovanje koji su također izlaz, jer ih program prikazuje. Sve ove elemente treba do u detalje uskladiti sa budućim korisnicima programa i od njih dobiti odgovarajuću saglasnost za dogovoreno.

Koristeći se programom kao što je Visual Basic, može se prikazati model ekrana, koji korisnici tako mogu i da vide, te potvrde da programi za unose podataka sadrže sve ono što korisnik treba i želi.

Programi za unos su dio definicije izlaza. Definiranje izlaza također pomaže u postavljanju ulaza koji je potreban da bi se proizveo izlaz. Treba sagledati sve podatke koji idu u program.

Potrebno je prvo razjasniti sve detalje problema koji program rješava, da bi se tačno znalo kakav nam izlaz treba.

Zaključno, svi izlazni ekrani, štampani izvještaji i ekrani za unos podataka moraju biti unaprijed definisani. Također se mora odrediti koji podaci će se čuvati u datotekama i u kom formatu.

Primarni problem kod današnjih programera je to što uopće ne ulažu vrijeme u posao dizajna. Zbog toga dolazi do različitih problema.

Iako je stvarni dizajn izlaza, podataka, pa čak i logike u tijelu programa danas mnogo jednostavniji za rad s dostupnim programskim alatima, ipak se mora velika pažnja pokloniti inicijalnom dizajnu izlaza o kome je postignut dogovor s budućim korisnicima. Moraju se također poznavati svi podaci koje naš program treba da sakupi, prije nego što se počne s kodiranjem. Ako se ovo ne uradi, programer će imati velikih problema, jer će stalno nadoknađivati ono što su korisnici izostavili, ili propustili da mu kažu.

Izrada prototipa

Jedna od prednosti Windows operativnog sistema je njegova vizuelna priroda. Prije Windowsa, programski alati su bili ograničeni na dizajn baziran samo na primjeni tekstualnih formi. Danas dizajniranje korisničkih ekrana znači počinjanje rada s programskim jezikom kakav je Visual Basic, crtanje ekrana i postavljanje objekata na ekran (kao što je dugme OK), s kojima će korisnik stupati u interakciju. Na ovaj način se može brzo dizajnirati prototip ekrana koji se onda može prezentirati budućim korisnicima. Kada korisnici vide ekrane s kojima bi trebali da rade, oni mogu kazati svoj stav o tome da li je programer dobro shvatio njihove potrebe i zahtjeve.

Liberty Basic nema nikakve alate za prototipove, ali Visual Basic i Visual C++, kao i Visual C#, pružaju mogućnosti kreiranja grafičkog korisničkog interfejsa za aplikacije. Uz pomoć ovih alata, postavljaju se kontrole, kao što su komandna dugmad, i polja za tekst na formu koja služi kao izlazni ekran. Pri tome programer ne mora napisati ni jednu liniju programskog koda.

Kada se postave kontrole na prozor forme, koristeći se programskim alatom kao što je Visual Basic, forma se može kompajlirati na način kako se kompajlira i program, a zatim i pokrenuti, tako da budući korisnici programa mogu probati rad s kreiranim interfejsom i na osnovu toga dati svoje primjedbe i dodatne zahtjeve.

Brzi razvoj aplikacija

Napredniji programski i dizajnerski alati koji se koriste za definisanje izlaza, toka podataka i logike su programi za brzi razvoj aplikacija – RAD (engl. *Rapid Application Development*). Iako su RAD alati.

RAD alati omogućavaju proces brzog postavljanja kontrola na formu, kao i proces spajanja tih kontrola s podacima, te pristupanje prethodno napisanom kodu, da bi se to na kraju sklopilo u cjelinu bez pisanja ijedne linije koda.

2. korak: Razvijanje logike

Kada se programer s korisnikom složi oko ciljeva i izlaza programa, ostatak posla je prepušten programeru. On treba da odluči kako da računar proizvede izlaz. Potrebno je da se razvije logika koja će da proizvede taj izlaz.

Kod postavljanja logike, važan korak je definisanje postupka koji će omogućiti da se od ulaznih podataka dođe do izlaza. Potrebno je utvrditi koji će proračuni biti potrebni da bi se dobio izlaz poslije unosa ulaznih podataka. Programer mora biti u stanju da generiše ispravan tok i proračun podataka, tako da program može da manipuliše tim podacima i da proizvede ispravan izlaz.

Jedna od mogućnosti za razvijanje logike za program je korištenje dijagrama toka, koji pruža slikovni prikaz programske logike. Dijagram toka ne uključuje sve programske detalje, ali predstavlja generalni logički tok programa. Poslije završetka finalnog programa, dijagram toka može da predstavlja dokumentaciju samog programa.

Dijagrami toka su sastavljeni od standardizovanih simbola. Postoje programi koji pomažu u kreiranju dijagrama toka.

Prisutan je stav da su RAD i ostali razvojni alati eleiminisali potrebu za dijagramima toka. Ovdje se ističe da je korištenje dijagrama toka opravdano, ukoliko programer smatra da mu Opprimjena istih može olakšati i ubrzati posao kreiranja programa.

Neki izvori preporučuju i korištenje pseudokoda.

3. korak: Pisanje koda

Učenje pisanja programa zahtijeva najviše vremena. Kada se nauči programirati, stvarni proces programiranja u principu traje kraće nego dizajn, pod uslovom da je dizajn tačan i kompletan

Zaključak

Programer ne bi smio da piše program prije nego što ga dizajnira. Često programeri žure prema tastaturi, prije nego što dovoljno razmisle o logici programa. Loše dizajniran program rezultira brojnim greškama, koje se u dugom periodu moraju otklanjati i mogu dovesti i do napuštanja programa od strane korisnika.