1 Sarrera

Programazioaren hastapenetan makina-lengoaian idazten ziren programak. Ez zegoen beste aukerarik, konputagailuak ezin baitzezakeen kode bitarrean idatzikoaz besterik ulertu. Sasoi hartan nola-halako notazio sinbolikoa erabiltzen zen makina eragiketak eta, hauek kateatuz, programak idazteko.

Pixkanaka, mihiztadura-lengoaiak sortu ahala, kode sinboliko hori makina-kodetik urruntzen hasi zen. Lengoaia hauetan idatzitako kodea programa itzultzaileen bidez (mihiztatzaileak) makina-kodera pasatzen zen. Makinatik urrundu bai, zertxobait urruntzen zen mihiztadura-kodea, baina hala ere menpekotasun edo lotura estua zeukan makinarekiko; hain estua ezen lengoaiako primitiboak makinaren oinarrizko hardware-eragiketak baitziren. Hitz gutxitan esateko, programagintza astuna zen, metodorik gabea, errore-arrisku oso altukoa eta hauen detekzio eta arazketa benetan nekosoa.

Gauzak honela, alegiazko makina erosoago baten premia nabarmendu zen, edo, nahi bada, erabilgarriago gertatuko zen programazio-lengoaia batena.

Diseinu-ahaleginak 1945ean hasi ziren, eta lan horien fruitu gisara 50. hamarkadaren bukaera aldera agertu ziren estreinako programazio-lengoaiak. FORTRAN izan zen lehenengoa, 1954 eta 1957aren artean John Backus eta bere IBM taldeak garatua, eta benetan eragin handikoa. Harrera eszeptikoa egin zitzaion, ez baitzen uste hain goi-mailako lengoaia eraginkorra izan zitekeenik, eta horregatik ez zen inguru akademikoetatik atera harik eta Backus-en taldeak konpiladore egokia sortu eta behe-mailako kodeekin bezain emaitza onak lortu zituen arte. Orduantxe eman zuten ameto eszeptikoek. Egia esatera, FORTRAN horrek mihiztadura-lengoaien ezaugarri askotxo zituen.

Garaitsu hartan (50eko hamarkadaren bukaera eta 60koaren hasieran) lengoaia ugari jaio zen: ALGOL, COBOL, APL, BASIC, LISP. Guztiek, LISP-ek izan ezik, bazuten FORTRAN lengoaiaren eraginik eta, modu batera edo bestera, hura hobetzeko asmotan sortu ziren: programen irakurgarritasuna zela, datuen erabilpena zela, e. a.

Ugalketa horrek eztabaidari eta, bidenabar, lengoaien arteko alderaketari eman zion bidea. Ingurune honetan sortu zen programazio-lengoaia unibertsalaren ideia eta halaxe jaio zen 1965ean PLI lengoaia. Esperientzia honek ederki erakutsi zuen benetan zaila zela "denetarako" balio zuen lengoaia ikasi eta ondo erabiltzea. Nabaria zen aplikazio-arlo desberdinetan lengoaia berezituak behar zirela.

Garai hartako literatura ("Programming in Basic", "Programming in Fortran IV") ikusi besterik ez dago konturatzeko benetan garrantzizkoak lengoaiak zirela. Programatzen jakitea programazio-lengoaiak jakitea zen, eta ez besterik. Programatzaileak bere kaxa ikasten zituen oinarrizko arauak, askotan aritzeak ematen duen eskarmentura mugatzen zirenak.

60ko hamarkadan aplikazioen konplexutasuna, tamaina eta garrantziak gora egin zuten arren, ez zen gauza bera gertatu programatzaileen iaiotasun eta tresneriarekin. Programa gutxi eta eskasak egiten ziren, eta denbora gehiago behar izaten zen errore-arazketan programak sortzen baino. Egoera penagarri horrek bultzatuta, 1968ko urrian "Software Engineering" izeneko konferentzia ospatu zen Garmisch-en (Alemanian) NATOren babespean. Bilera hartan "Softwarearen krisiaz" hitz egin zen, hau da, beharren eta metodoen arteko desorekak sortutako krisiaz. Bildutakoak, akademiko nahiz industri gizonak, adostasun batera ailegatu ziren: "Softwaregintza krisian badago programazioa orain artean gaizki ulertua izan delako da". Planteamendu horrek aurrekoarekin apurtzea eta bide berrietatik abiatzea ekarri zuen. Urtebete barru, 1969an, IFIP elkarteak Working Group 2.3 lan-taldea sortzea erabaki zuen, *programazioaren metodologia* lantzera dedikatuko zena, hain zuzen ere.

Talde horrek erro-errotiko aldaketak proposatu zituen: utikan artisau lana, egin dezagun programazio zientifikoagoa! Ikusmolde horrek bi oinarri ditu:

1. Programa baten ezaugarri nagusiak honako hauek izan behar dute: zuzentasuna, argitasuna, aldagarritasuna eta eraginkortasuna.

2. Programatzaileak behar-beharrezkoa du bere lana erraztuko duen tresneria eta oinarri teorikoa.

Pentsamolde honen haritik eratorritako programazio-estiloari *programazio egituratu* deitu zitzaion. Bazirudien, hasieran izandako zenbait eztabaidaren arabera behinik behin, programazio egituratua *goto* agindurik gabe programatzea baizik ez zela. Luzagabe argitu zen, ordea, hori baino zerbait gehixeago zela programazio egituratuaz ulertu behar zena.

Hortik aurrera ez ziren falta izan era guztietako ekarpenak, hala nola, programazio-lengoaiak, programen diseinurako metodologiak, e. a.

1970ean PASCAL programazio-lengoaia jaio zen, N. Wirth-ek diseinatua. Lengoaia honetan notazio hobeak, kontrol-egitura argiagoak eta datuen definizio eta maneiurako erraztasunak bildu nahi ziren. Gerora, PASCAL lengoaia hainbat programazio-lengoaia agintzaileren aurrekaria gertatu da, esate baterako, MODULA-2 eta ADArena.

PASCAL-arekin batera abstrakzioaren kontzeptua programazioaren mundura ailegatu zen. Abstrakzio funtzionalak ebatzi beharreko problemaren konplexutasuna zatikatzeko aukera ematen zuen. Problemari bere osotasunean aurre egin beharrean, zatika, aldiro problemaren aspektu batzuk ebaztean, eta gainerakoei itzuri egitean, datza abstrakzioa. Honelaxe plazaratu zuen N. Wirth-ek ondoz-ondoko finketen metodoa 1971n. Ordutik aurrera programazio-metodologiaren funtsa hauxe izan da: programatzea ez da soilik programak idaztea. Programen idazketa, izatekotan, analisi eta diseinu prozesuaren azken urratsa da, eta prozesu hori metodo ordenatu batez burutu behar da, ez nola edo hala. Ondoz-ondoko finketen metodoak berebiziko eragina izan du geroztik formulatu diren metodoetan, esate baterako, gestiorako programen diseinuan ere beheranzko metodoa erabiltzen da, Warnier metodoan kasu.

Programen zuzentasuna egiaztatzeko metodoek ere, ez soilik probetan oinarritutakoek, piztualdi polita ezagutu zuten testuinguru honetan. Lehendik ere, 1949an, A. Turing-ek plazaratua zuen behar hori, bereziki bere metodoa aurkeztu zuenean. Bada bai ildo honetan lan egin duen aitzindaririk, besteak beste, McCarthy, Naur eta Floyd aipa daitezke. Floyd-ek, 1967an, Turing-en ideian oinarritutako metodoa aurkeztu zuen, baina asertzio gisa aldagaien balioak erlazionatzen zituzten formulak

erabili zituen. Guzti horren buruan inor gutxik jartzen zuen zalantzan programen fidagarritasuna hobetzearen beharra eta, halaber, testak soil geratu eta ez zirela nahikoa. Dijkstra-k zioen bezala, probek programak erroreak badituela froga dezakete, baina ez ordea errore gabea denik. Horretan dago, hain zuzen, koska; horregatik da hain interesgarria programak egiaztatzea.

1969an, Hoare-k, Floyd-en ideiak berreskuratuz, axioma eta inferentzi erregelatako sistema formala eratu zuen. Sistema honek programen zuzentasun formala frogatzeko balio du eta asertzio inbarianteen metodoaren euskarria da.

Geroztik programen zuzentasuna formalki frogatzeko zenbait metodo azaldu da, hala nola, "aldizkako asertzioena" edo semantika denotazionalean oinarritutakoa. Baina guztietan erabiliena Hoare-ren kalkulua izan da ezbairik gabe.

Handik lasterrera, ondoz-ondoko birfinketen diseinu-metodoa eta asertzio inbarianteen egiaztapen-metodoa ezkontzeko asmoa sortu zen. Asmoa sinplea zen: arrazoizkoa da, eta eraginkorragoa gainera, programa egin ahala zuzena dena frogatzea, bi eginkizunak, diseinua eta egiaztapena, ideia bertsuen garapenak dira-eta. Horrez gain, programagintza-metodo horrek aukera aparta eskaintzen du diseinuaren propietateak eta erabilitako ideiak ondo dokumentatuta uzteko, eta hori bai dela eskertzekoa programen mantenimendu-fasean.

Dijkstra-k programen eratorpen formala izeneko metodoa planteatu zuen 1975ean, hau ere Hoare-ren kalkulutik eratorritakoa. Metodo horren arabera, programaren helburua espezifikazio jakin bat betetzea da eta horretara zuzendu behar da diseinua. Espezifikazioaren ondoko baldintzak, batetik, lortu nahi den emaitza nolakoa den adieraziko digu eta aurreko baldintzak, bestetik, datuek bete behar dituzten murriztapenak finkatuko ditu. Programazioa hala planteatuz gero ondoko baldintza bihurtzen da diseinuaren gidari, baina, hori bai, diseinuak eskatzen dituen gutxienezko betebeharrak aurreko baldintzak biltzen dituela aurreikusiz. Funtsezko ideia eta formalismoa Dijkstra-k azaldu bazuen ere, Gries izan zen espezifikaziotik abiatuta programak eratortzeko metodoa gehixeago landu eta ezagutzera eman zuena.

Programen eraldakuntzak ere badu zeresanik aipatzen ari garen arrazonamendu eta zuzentasunaren egiaztapenean oinarritutako metodologian. Normalean, eraldatu, programa sinpleak eraldatzen dira, espezifikazioa betetzen dutenak baina eraginkorrak ez direnak. Programa horien semantika babestuz (zuzentasunari eutsiz) baliokide eraginkorragoak lortu nahi izaten dira. Ez da, beraz, beheranzko metodoa, horizontala baizik. Gehienetan, programa eraginkorrak ulergaitzak dira, eta zaila izaten da zuzentasunari buruz arrazonatzea. Horrelakoetan, askozaz erosoagoa gertatzen da jatorrizko programa zuzena zela eta eraldaketak zuzentasunari eusten diola egiaztatzea.

Egia da programa-eraldakuntzak programa batetik besterako pausoa ematen duela eta ez espezifikaziotik programarakoa. Eraldakuntzan ere badira, ordea, espezifikazioaren eta egiaztapenaren beharra duten problema batzuk, txukun ebatziko badira behintzat. Eraldaketak programen zuzentasuna kontserbatzen duela frogatu nahi bada, esate batera, beharrezkoa da programaren espezifikazioa ezagutzea eta egiaztapenteknikak erabiltzea. Gainera, zenbait eraldaketak ez du, murriztapenik ezarri ezean, zuzentasuna beti kontserbatzen.

Programa errekurtsiboak gehientsuenetan sinpleak eta ulerterrazak dira, baina baita ez-eraginkorrak ere. Ez da harritzekoa, beraz, errekurtsibo-iteratibo ereduko eraldakuntzak izatea gaur egun erabilienak eta landuenak.

Programen sintesia edo programazio automatikoa ere espezifikaziotik hasi eta programa sortzera jotzen duen metodoa da, baina kasu honetan adimen artifizialaren inguruko teknikak erabiltzen dira. Batzuetan, espezifikazioa lengoaia funtzionalean

idazten denean, sintesia ez da programa-eraldakuntza mekanizatua baizik. Beste batzuetan, espezifikaziotik programa bat lortzen da zuzenean eta gero programa hori hobetu egiten da programa-eraldakuntzaz.

Programen egiaztapen-prozesuak dituen aspektu mekanikoenak automatizatzeko joerak, ikerkuntza adar berri bati eman zion sorrera egiaztapenaren alorrean. Beste helburu batzuen artena, programazio-inguruneetan tresna berri bat integratu nahi zen: egiaztapen-sistemak; eta hartara programen zuzentasun semantikoa egiaztatzeko erraztasunak eskaini. Hamaika ahalegin egin da 70.etik aurrera egiaztapen-metodo mekanizagarriak deskribatzeko eta egiaztapen-sistema batzuk eraiki ere egin dira, baina elkarrekintzazkoak eta esperimentalak dira oraingoz. Etorkizunean erabilgarria izango litzateke egiaztapen-sistemak programazio-ingurune integratuetako osagaiak izatea, editoreak, konpiladoreak eta liburutegiak bezalaxe.

Abstrakzioarekin zerikusia duen beste programazio-metodo inportantea programa-eskematan oinarritzen dena da. Programa-eskema algoritmo generikoa da, ekintza, funtzio eta objektu abstraktuak (interpretatu gabeak) dauzkana.

Programa-eskemek ideia zahar batean dute jatorria: ebatzi nahi dugun problemak beste batekin erlazio estua badu, eta azken honen ebazpena ezaguna bada, ebatzi dezagun jatorrizkoa bigarrenean oinarrituta.

Azken batean, algoritmo askok portaera bertsua dutela aipatzen duten liburuak programa-eskemen erabileraz ari dira, nahiz eta ez duten diseinu-metodo hau esplizituki aipatzen.

Dijkstra-k "programa-familiak" aipatzen ditu egoki baino egokiago, hau da, algoritmo abstraktu batetik (oraindik birfintzeke dagoena) erator daitezkeen programak jotzen ditu senide eta hauek, bere ustetan, problema-mota (edo familia) bati aplika dakizkioke. Eskema bera, jatorrizko algoritmo abstraktua alegia, dokumentazio ezin aproposagoa da mantentze-faserako.

Eskemen bidezko programa-diseinurako problemak klaseka bildu behar dira, ebazpidean erabiltzen diren ezaugarri komunen arabera. Honela, behin problema klaseren batean sailkatuta, programa-eskema zehatz bat aukeratzen da ebazpenerako, eta eskema horretatik ekintzak eta objektu abstraktuak birfindu beste lanik gabe, programa lortzen da.

Orain artean azaldutako programazio-metodologiaren aspektu horietan oinarritu gara irakasgai honen edukia mamitzean. Esan behar da, gainera, programazioaren inguruko ikerkuntz lerro nagusiak ingurune horretan ari direla garatzen gaur egun.

Hemen jasotzen den materiala programazioko hasierako mailara zuzenduta dago eta bertan programazio agintzailearen oinarrizko ezagutza duen irakurleari programazio metodikoaren bidea erakutsi nahi zaio. Garrantzizkoa deritzogu bide hau urratzeari, izan ere, Dijkstra-k esan zuen bezala, zaila baita artisau-programazioan ohituta dagoenari programazio metodikoaren komenientzia ikustaraztea.

Gure iritziz, programatzea ez da "funtzionatzen" duten programak idaztea soilik. Hori baino areago, programei argitasuna, moldagarritasuna, dokumentazio aproposa, eraginkortasuna eta estilo horretako ezaugarriak eskatu behar zaizkie.

Programatzea ez da jarduera nabaria, ezta sinplea ere. Ezin da edozein moduz programatu, baizik eta lagungarri gertatuko diren tresneria eta ezagutza egokiez baliatuta. Programen diseinu-prozesuak ideien antolaketa eta zorroztasunez egindako azterketa eskatzen du derrigor. Hala egiteak, zenbaitetan bederen, programazioa zaildu egiten duela eman lezake, baina egiaz diseinu-erroreak egiteko arriskua txikitu egiten du (hauek dira gainera detektatzeko eta zuzentzeko gaitzenak) eta ondorioz programaziolana erraztu.

Horrez gain, txukun arrazonatu eta eratutako diseinua da izan daitekeen dokumentaziorik egokiena, eta aspektu inportanterik bada programazioan, horietako bat dokumentazioa da.

Behin problemaren ezaugarriak aztertuta, komeni da programazio-metodo egokiena aukeratzeko gai izatea. Baina horretarako beharrezkoa da metodo eta kontzeptu desberdinak noiz eta nola erabili behar diren jakitea.

Sarrera gisako honetan ukitu ditugun aspektu hauetako batzuk lantzen dira irakasgai honetan zehar, beti ere programa zuzen, argi, eta ulergarriak idatzi nahi dituenari laguntza eskaini nahian.