1. Sarrera

Kapitulu honetan liburuaren edukiak deskribatuko dira, Software Ingeniaritzan duten garrantzia nabarmenduz. Software Ingeniaritzaren arloari eman zaizkion definizio ugariren artean, IEEE erakundearena[1] da honako hau:

> Softwarearen garapena eta mantentzea ikuspegi sistematiko, diziplinatu eta kuantifikagarri batez egitea, eta ikuspegi horren aplikazioaren azterketa [75].

Horretarako, 1.1. atalean *softwarearen garapenerako metodo formal*en bilakaeran erabakigarriak izan diren faktoreak historikoki kokatuko ditugu. Ikuspegi historiko hori baliagarria zaigu liburu honetan lantzen diren gaiak aurkezteko, eta, bidenabar, gai horien eragina nabarmentzeko. 1.1. atala [76]-n oinarrituta dago partzialki. 1.2. atalean, liburuaren egitura eta edukiak era zehatzagoan deskribatuko dira. Eskerrak emateko ataltxoarekin bukatuko da kapitulua.

1.1. SOFTWAREAREN GARAPENERAKO METODO FORMALAK

Aurrez planteatutako zeregin edo problema baten ebazpena da, funtsean, programa informatikoa. Askotan, jatorrizko zeregin hori emaitza batzuk kalkulatzera edo zerbitzuren bat ematera orientatuta egoten da. Programa informatikoak eraikitzeko jarduera aztertzen duen arloari *programazio* esaten zaio. Hain zuzen ere, programazio-eredu baten oinarriak deskribatzera zuzendua dago liburu hau. Eredu horri hainbat izendapen eta kalifikatzaile jarri izan zaizkio: *programazio metodikoa* edo *programazio zientifikoa* izan dira zabalduenak. Berrikiago *kontratu bidezko programazioa* ere erabili izan da. Esan daiteke ikuspegi honen bilakaera 60ko hamarkadan hasi zela, *programazio egituratua* deitutakoaren eta ALGOL [11] eta Pascal [57] programazio-lengoaien sorrerarekin batera. Programazio egituratuaren oinarrizko ideiak oso ondo daude bilduta Dijkstra-ren [32] maisu-lanean. Ikuspegi egituratu horren funtsezko ideietako bat diseinatutako programen zuzentasuna era sendoan eta

1. Institute of Electrical and Electronics Engineers.

sinesgarrian ziurtatu ahal izateko bidea erraztea zen. Horren haritik iritzikorronte berria sortu zen, programazioa ordura arte aurreikusten zena baino jarduera zientifikoagoa (artisau kutsu gutxiagokoa) zela ulertzearen aldekoa hain zuzen.

Programa bat *zuzena* dela esango dugu, haren diseinua eragin zuen problema ondo ebazten badu, beti (edozein direla ere datuak) esperotako emaitzak lortuz eta inolako errorerik edo ustekabeko ondoriorik sortu gabe. Definizioz, programen ezinbesteko propietatea zuzentasuna da, izaki bizidunena bizia den bezalaxe. Jakina, badira programa guztietan desiragarriak diren eta programak ondo diseinatuta daudela edo ez daudela esatera garamatzaten beste propietate batzuk ere. Propietate horietatik, besteak beste, eraginkortasuna, argitasuna, erabilgarritasuna, moldagarritasuna, sendotasuna eta ondo dokumentatuta egotea azpimarra ditzakegu.

Programa zuzen bat diseinatzea, oro har, zailtasunetatik libre ez dagoen lan neketsua da. Programaren zuzentasuna argi erakutsiko duen dokumentazio edo argudiaketa on bat osatzea ere ez da samurra, batez ere balizko irakurlea konbentzitu nahi bada, eta zuzentasuna bermatzen duten propietateak agerian utzi nahi badira. Azken batean, zuzentasuna argumentatzea zenbat eta errazagoa izan, orduan eta aukera gehiago egongo dira programa aise ulertua izateko, eta, era berean, aise dokumentatua, optimizatua, egokitua edo mantendua izateko.

Horiek horrela izanda ere, programazioaren historia labur samarrean ugariak dira pertsonei nahiz ondasunei kalte konponezinak eragin dizkieten software-erroreen adibideak. Hondamendi horietako batzuk oso ezagunak dira, esate baterako, Ariane 5 suziriarena, eta, beste batzuk ez horrenbeste, Therac-25 erradioterapia-makinarena aipa daiteke kasurako. Lehenengoa 1996an gertatu zen, Ariane 5 suziri europarrak abiatu eta minutu-erdira eztanda egin zuenean. 64 bit-eko zenbakiak 16 bit-eko zenbaki bihurtzean egindako akatsak eragin zuen leherketa, zehazki, bihurketa egiten zuen programari onar zezakeena baino zenbaki handiagoa iritsi zitzaionean. Bigarren hondamendia izatez lehenago gertatu zen Kanadan, 1985 eta 1987 artean, Therac-25 erradioterapia-makinan. Makina horrek 4 pertsonaren heriotza eragin zuen, eta beste batzuei lesio larriak eragin zizkien. Ezbeharraren kausa *race condition* izeneko programazio-akats bat izan zen; errore hori gertatzen da programa kanpo-gertaeren ordena jakin baterako diseinatzen denean, baina gertaeren benetako sekuentzia bestelakoa denean. Aski ezagunak dira, halaber, programazio-errore sinpleen eraginez hondatu izan diren NASAren suziriak. Horien artean aipagarria da 1962an espazioan galdu zen Mariner 1 izenekoa. Artizarrera bidalitako lehenengo espazio-zunda zen. Galera hori ibilbidea kontrolatzen zuen programan, formula edo espresioren

baten transkripzioan, eginiko errore sinpleren batek eragin ote zuen espekulatu izan da. Errakuntza posibleen artean, apostrofo bat (') koma batekin (,) nahastu izana ere aipatu izan da. Beste errore sinple batek eragin zuen 1999an Mars Climate Orbiter zundaren suntsiketa, Marte planetaren atmosferarekin izandako marruskaduraren ondorioz. Berez, zunda horrek ez zukeen inoiz Marteren atmosfera ukitu behar, baina nabigazio-programaren akats batek ibilbidea aldarazi zion; hain zuzen ere, ibilbidea kalkulatzeko, neurriak sistema metriko hamartarrean jasotzea espero zuen programak, baina lurreko kontrol-sistemak eredu anglosaxoiaren araberako neurriak bidali zizkion, hots, miliak, hazbeteak, etab.

Era askotako arrazoiak daude programatzean erroreak egiteko arrisku handia dagoela baieztatzeko. Gainera, akats horiek, itxuraz sinpleak eta hutsalak iruditu arren, ondorio lazgarriak eragin ditzakete. Aurreko adibideetan ikusi den bezala, sistema metriko baten ordez beste bat erabiltzeak, eragiketa aritmetiko bat beste batekin nahasteak, edo objektu baten tamainari muga maximo txikiegia jartzeak, zeharo alda dezakete programaren portaera. Hala, programak egin beharko ez zukeen zerbait egingo du, edo ez du ondo egingo egin beharko zukeena, eta zenbaitetan kalte larriak eragin. Ikusitako adibideak egoera orokor baten isla dira. Programa akastunen ugaritasuna eta errore horiek kodean aurkitzeko zailtasuna dira programazioaren arloari oinarri matematikoa emateko interesa eta ekimena piztu zuten arrazoi nagusiak.

Azken hamarkadetan bi teknika mota sortu dira programazio-erroreei aurrea hartzeko. Alde batetik programak arazteko (alegia, erroreak detektatzeko eta zuzentzeko) teknikak eta tresnak garatu dira. Tresna horiek zenbaitetan automatikoak badira ere, arazte-lan hutsak ezin du errorerik eza bermatu[2]; nork ziurta dezake ez dagoela detektatu gabeko errorerik? Erabat ziur egoteko erarik ez digu ematen arazketak, programaren fidagarritasunean dugun konfiantza areagotzeko balio du bakarrik. Hala ere, nork eta nola arazten duen, horren araberakoa da konfiantza, eta, hori, bistan denez, ez da oso zientifikoa. Programazioari izaera zientifikoagoa emateko asmoz, teknika matematikoagoak edo teknika formalak sortu ziren. Programak diseinatzeko eta analizatzeko dauden teknika formal gehienak *matematika diskretu*an, eta, bereziki, *logika matematiko*an oinarritzen dira. Matematika diskretua da software-ingeniaritzaren euskarri matematiko/zientifikoa, bestelako ingeniaritzen euskarria *kalkulu infinitesimala*, *estatistika* edo *zenbakizko analisia* diren bezalaxe. Edozein ingeniaritzak darabiltza metodo aurredefinituak eta estandarrak bere jardueraren funtsezko parte gisa, eta horietako askok osagai

2. Ederki azaldu zuen hori E. Dijkstra-k [32]-n: «Programa-arazketa akatsak badaudela erakusteko erabil daiteke, baina inoiz ez akatsik ez dagoela erakusteko».

matematiko nabarmena dute.

Programatzaileei (software-ingeniaritzako espezialistei) metodo erabilgarriak eta sendoak eskaintzeko beharrak proposamen ugari sorrarazi ditu, eta horren isla da azken berrogeita hamar urteetako literatura. Programa zuzenen diseinu sistematikorako metodo eta teknika horiek, funtsean, bi fasetan bereizten dute programen diseinua. Lehenengoan, ebatzi beharreko problema era zehatz, doi eta osatu batean espezifikatu behar da. Programak espezifikatzean egoki jaso behar dira egoera onargarri guztiak, eta adierazi behar da, halaber, nolako jokabidea izango den egoera horietan guztietan. Fase horretan egindako akatsak hurrengora, programazio-fasera, igarotzen dira, eta horrek zaildu egiten du programatu beharrekoaren ulerkuntza eta erroreen detekzioa, nahastu egiten baitira modelatze-erroreak eta programazio-erroreak.

60ko hamarkadaren erdialdera arte programek ez zuten hainbesteko pisurik gizartean, eta diseinatzen igarotzen zen denborari edo lortutako produktuaren kalitateari ez zitzaion hainbesteko garrantzirik ematen. Garai horretan nabarmendu zen programa, izatez, esanahi zehatza eta doia duen objektu matematikoa dela. Horrela izanda, programa zehaztasun matematikoz aztertua eta landua izan daiteke haren portaerari buruzko propietateak frogatzeko, bereziki, zuzentasuna frogatzeko. Ildo horretan, 60ko hamarkadaren bukaerarako zenbait metodo eta teknika sortuz programazioari izaera zientifikoagoa emateko xedea zuen ikerlerro garrantzitsu bat bazen (E. W. Dijkstra, R. Floyd eta C. A. R. Hoare ikerlerro horretan zeuden). Berehala jaio zen *programen egiaztapena:* programa batek bere espezifikazioak dioena betetzen duela egiaztatzeko balio duen frogapen matematikoa (logikoa). Bide horren hastapeneko lantzat har daiteke A. Turing-en [89], baina programen egiaztapenaren benetako ernamuinak R. Floyden [40] eta C. A. R. Hoareren [52] dira. Ordutik hona, programazioko metodo formalen arloa etengabeko bilakaeran aritu da. Teknika eta metodo ugari sortu dira, eta programazio-rako oinarri matematiko sendoak landu dira. Egindako aurrerapenek, bai programazioan bai erlazionatutako beste arlo batzuetan (bereziki, software-ingeniaritzarako duen aplikazioan eta *arrazoibide matematikoan*), formalismo berri erabilgarriagoak sorrarazi dituzte, eta beste aplikazio-eremuetarako aukerak zabaldu dituzte. Softwarearen garapen formalerako prozesuaren zenbait atal automatizatzea ere ekarpen garrantzitsua da arloaren bilakaeran.

Metodo formalek programatzaileari laguntzeko helburua dute, eta programa zuzenen diseinua errazteko tresna baliagarri izan nahi dute. Programen diseinurako erabil daitekeen metodologia bakoitzaren egokitasuna, egoera eta testuinguru desberdinen araberakoa da. Programa zuzenak diseinatzean, zein aspekturi aurre egin nahi diogun, horren araberako teknikak erabili behar

dira. Horretarako beharrezkoa da programatzaileak metodo horien ezagutza izatea, eta gai izatea egokiro aplikatzeko, esate baterako, problemak espezifikatzean, parte hartzen duten datu-egiturak espezifikatzean, ebazpen algoritmikoa asmatzean eta formalizatzean, exekutagarria lortzean, planteatutako ebazpenaren zuzentasuna justifikatzean edota diseinatutako programa dokumentatzean. Tekniken eta tresnen ezagutza ezak problemen ebazpen egokia eragozten duela adierazteko, karikatura adierazgarria egin zuen A. H. Maslow-ek [69]-n, horretarako atsotitz ezaguna erabiliz[3]: «Mailua baizik ez baduzu, den-dena iltzea balitz bezala landuko duzu». Programatzaileari tresna hobeak eskaintzeko asmoz, azken hamarkadetan aurrerapen nabarmena gertatu da teknika formalei dagokienez, eta emaitza praktiko batzuk harrigarri samarrak ere izan dira.

Jakina da softwarea garatzeko metodo formalen erabilera eztabaidagarria izan dela, baina badirudi gero eta onartuago dagoela. Aldekoek beti esan izan dute formalismoa onuragarria dela, eta industriak garapen modu zorrotzagoak eta laguntza-tresna egokiak bereganatu beharko lituzkeela. Aurkakoek zailtasun serioak eta intrintsekoak ikusten dizkiete metodo formalei, tamaina handiko proiektu industrialetan erabiliak izateko. Jarrera horren inguruko zenbait mito [50] sortu eta puztu dira, baina mito horiek indarra galdu dute arloak heldutasuna lortu ahala eta industria egokitu ahala. Gaur egun ekarpenak askotarikoak dira, metodoak aplikazio-eremuetara egokitzen ari dira, eta bilakaera sendoa da beste arlo batzuekin lankidetzan. Azken urteotan bereziki nabarmendu dira programazio formalerako laguntza-tresna automatikoak (edo erdi-automatikoak), eta esan daiteke arlo horretan oso ekimen aktiboa eta emankorra nabari dela. Metodo formalak eguneroko lanean benetan erabilgarriak izan daitezen, beharrezkoa da garatzaileek prestakuntza espezializatua jasotzea. Eta, noski, prestakuntza behar horrek zaildu egiten du metodo horiek esparru industrialean ezartzea. Zailtasun hori eta beste zenbait oztopo identifikatu eta landu dira mundu akademikoaren eta industrialaren arteko lankidetza estuan. Industriak gero eta interes handiagoa erakutsi du metodo formaletan, eta horretan zerikusi handia izan du aplikazio informatikoak gero eta garrantzitsuagoak eta konplexuagoak izateak[4]. Egun, softwarea garatzeko prozesuetan tresna sendoak integratzea lortu da dagoeneko, eta tresna horiek erabilerrazagoak dira. Testuinguru horretan tamaina handiko aplikazio industrial ugarik atera diote etekina metodo formalen erabilerari.

3. «I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail».

4. Nahiz eta esan behar den hardwarearen industriak urteetako aldea atera diola softwarearenari bilakaera honetan.

Metodo formalen erabilera arrakastatsuaren adibide deigarria da Parisko garraioaren automatizazioa. 1989an jarri zen martxan SACEM sistema, eta geroztik sistema horrek kontrolatu du A (RER) lineako tren guztien abiadura. A (RER) sistema Módula-2n idatzita dago eta 21.000 kode-lerro ditu. Kode horren % 63 (segurtasunaren ikuspegitik kritikotzat hartzen dena) automatikoki lortu zen espezifikazio formal batetik, eta, ondoren, egiaztatu egin zen modu erdi-automatikoan. Hau da, softwarea garatzeko teknika formal osagarriak erabili ziren sistemaren segurtasuna bermatzeko. Izatez, Parisko garraioan bada formalki garatutako software kritiko gehiago ere. Horrela, aipagarriak dira honako bi sistema automatiko hauek: batetik, metroaren 14. linean ezarritako sistema, eta, bestetik, París-Roissy lineako gidaririk gabeko autobusa. Lehen sistema arazorik gabe martxan dago 1998tik, eta bigarrena 2007tik.

Oro har, 90eko hamarkadaz gero, metodo formalak erabiliz garatu diren aplikazio industrial ugari eman dira ezagutzera. Gainera, garapen formal horiek dokumentatu egin dira, eta agerian geratu dira metodoen eta erabilitako laguntza-tresnen abantailak. Esate baterako, aipatu ditugun bi aplikazioetarako (Parisko metroa eta autobusa) formalki garatu den softwarea [18]-n eta [12]-n deskribatu da. Horrela garatutako aplikazio industrialetatik anitz garraiorako (nabigaziorako) kontrol-sistemak dira, hala lurreko nola aireko, ureko edota garraio espazialerako. Maiz samar aplikazio horiek erakunde akademikoekin elkarlanean garatu dira. Adibidez, Nieuwe Waterwegeko (Rotterdam-etik gertu) marea-barreraren kontrolerako sistema [24] CMG Public Sector B.V., Division Advanced Technology erakundeak garatu zuen, Twenteko Unibertsitateko *Formal Methods and Tools* taldearen sostenguarekin.

Metodo formalak erabiliz garatu izan diren tamaina handiko sistemen artean aipagarriak dira, halaber, sistema eragileak. Adibidez, L4.verified [5] proiektukoak nabarmentzekoak dira. Bereziki, seL4 [39] mundu errealeko software kritikoa da, formalki egiaztatu dena frogapen formaletarako sistema automatiko baten laguntzaz. 8.700 kode-lerro ditu seL4-k, eta horiez gain baditu behe-mailako lengoaia batean idatzitako beste atal batzuk ere. seL4 izan zen osorik eta formalki egiaztatu zen lehenengo sistema eragilea. Era berean, badira formalki eta osorik egiaztatu diren bestelako aplikazioak ere, hala nola konpiladoreak, komunikazio-protokoloak eta abar. Frogapen automatikorako tresnen nukleoak (*kernel*) ere formalki egiaztatu izan dira.

Metodo formalek industrian lortutako arrakastan eragin handia izan du zenbait enpresa handik softwarearen garapen formalerako tresnak sortzearen alde egindako apustuak. Tresna horiek erabiliz lortu da aplikazio industrial

5. `http://ertos.nicta.com.au/research/l4.verified/`

fidagarriagoak garatzea. Enpresa horien artean, adibide gisa, honako hauek aipa daitezke: Intel, IBM, Sony, Siemens edo Microsoft. Horietako enpresa batzuk beren software-produktuetan erroreak ez egiteagatik murgildu dira arlo honetan. Nolanahi ere, badira metodo formalen ikerkuntzan ahalegin handia egiten ari diren enpresa garrantzitsuak, eta softwarea zorrotz garatzeko tresna praktiko oso erabilgarriak ere sortu dituzte. Nabarmentzekoa da *software development* arloaren barnean metodo formalei buruzko proiektu ugari dituen Microsoft Research dibisioa. Proiektu horiek aurrera eramateko, Microsoftek ospe handiko profesionalak kontratatu ditu, besteak beste, aitzindari izandako C. A. R. Hoare zientzialaria. Proiektu horien bidez softwarea formalki garatzeko prozesuan lagungarri izango diren tresnen multzo bat sortzen ari da Microsoft. Frogatzaile automatikoak ere sortu dituzte, eta frogatzaile horiek tresna orokorragoen azpisistema gisa erabiltzen dira propietate jakin batzuk betetzen al diren frogatu behar denean. Microsoftek sortutako tresnen artean Dafny[6] izeneko lengoaia eta egiaztatzailea nabarmenduko dugu, liburu honen edukiekin lotura estua du eta. Dafny banaketa askekoa da, eta web nabigatzaile baten bidez ere erabil daiteke. Dafnyren webguneak aukera ematen du sistema online atzitzeko, sistemaren instalatzekoak deskargatzeko, eta dokumentazio nahiz laguntza egokiak jasotzeko.

Programen egiaztapen automatikorako (edo automatizaturako) tresnak ugariak dira, eta softwarearen industrian ohikoak diren hainbat lengoaiatan erabil daitezke. Egiaztatzaile ezagunenen eta erabilienen artean daude Spark, Key, ACL2, Isabelle, Jack, KIV, Spec#, eta egiaztatzaile horiek hainbat lengoaiatan idatzitako programak onartzen dituzte, adibidez, Java, Java Card, C, C++, C#, Ada, Occam eta ML. Egiaztapenaren osagarri gisa, sistema horiek bestelako tresnak ere integratzen dituzte, esate baterako, erroreen arazketarako erabiltzen diren proba-kasuen edo kontra-ereduen sortzaileak, edo baita kode-sortzaileak ere. Dafnyk, adibidez, C# kodea sortzen du. Sistema horiek erabiliz garatu diren aplikazio industrial konplexuen erreferentzia anitz topa daiteke literaturan. [93]-n autoreek metodo formalen aplikazio industriala zertan den sakonki aztertu dute. Artikulu horretan erabilgarri dauden tresnak nahiz garatutako aplikazio industrialak aztertu dira, baita aplikazio horien kodearen tamaina, arloko jarduera zientifikoa konferentzietan, aldizkarietan, biltegietan eta abar ere. [93]-n deskribatutako aplikazio industrialen artean daude hegaldien kontrolerako, finantza elektronikoetarako edo segurtasunerako sistemak, eta mikroprozesadoreak.

Azken urteotan nabarmen aurreratu da arrazoibide logikoa automatizatzeko zereginean, eta horrek ekarpen handia egin die metodo formalen euskarri gisa erabiltzen diren tresna praktikoei. Honela dio C. A. R. Hoarek

6. `http://research.microsoft.com/en-us/projects/dafny/`

2006ko [17] artikuluan:

> Azken 10 urte hauetan, frogak eraikitzeaz arduratzen den softwarearen eraginkortasuna 1.000ko faktoreaz hobetu da. Hori, hardwarearen abiaduran, ahalmenean eta eskuragarritasunean gertatutako hobekuntzetan lortutako 100eko faktoreari gehitu behar zaio. Bai softwareak bai hardwareak hobetzen jarraitzen dute, horrela datozen 10 urteetan programen egiaztapenari bidezkoa den aukera emanez.

Azpimarratzekoa da, era berean, lau urte lehenago (2002) Bill Gates-ek programen egiaztapenak gero eta garrantzi handiagoa duela aitortu izana[7]:

> Hamarkada askotan zehar, egiaztapen formala konputazio-zientzien Grial Santua izan da, baina orain oso garrantzitsuak diren arlo batzuetan —esate baterako, kontrolatzaileen egiaztapenean— softwarearen fidagarritasuna bermatzea ahalbidetzen duten tresnak eraikitzen ari gara.

Frogapen automatikoan egiten ari diren aurrerapenak hainbat eratan aplikatzen ari dira programen analisian eta garapenean. Izatez, bi arloen arteko elkar hartzea oso emankorra gertatzen ari da.

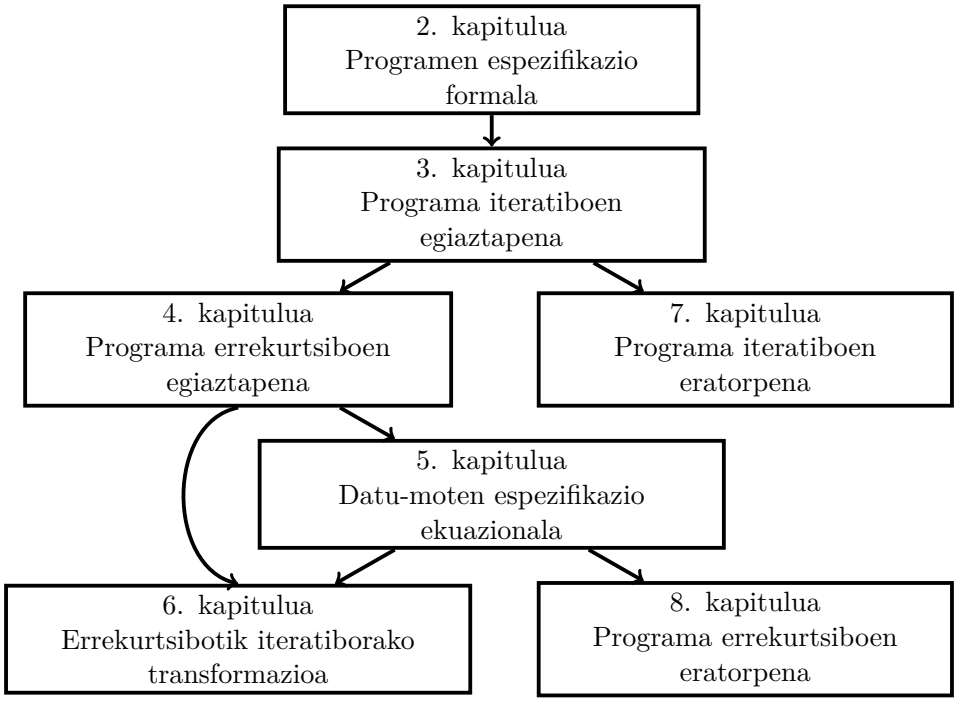
Berriki, 2012an, *DO-178C* dokumentua argitaratu da *Software Considerations in Airborne Systems and Equipment Certification* gaiari buruz. Dokumentu hori argitaratu izanak erakusten du aurrera egin dela metodo formalen gaian, eta, bereziki, softwarearen egiaztapenean. *DO-178C* dokumentuak ezartzen du zein oinarri bete behar dituzten hegazkinetarako software komertzialek, erakunde ziurtatzaileek (Estatu Batuetako Federal Aviation Administration (FAA) eta European Aviation Safety Agency (EASA) esaterako) onar ditzaten. Dokumentu horrek metodo formalei buruzko gehigarria du, eta bertan metodo formalak eta softwarearen egiaztapena honela azpimarratzen dira:

> Metodo formalak sistema digitalen softwarearen espezifikaziorako, garapenerako eta egiaztapenerako erabiltzen diren eta matematikatan oinarrituta dauden teknikak dira. Metodo formalen oinarri matematikoa logika formala, matematika diskretua eta ordenagailuak irakur ditzakeen lengoaiak dira. Ingeniaritzaren beste alorretan bezala, analisi matematiko egokiak egiteak diseinuen zuzentasuna eta sendotasuna bermatzen lagundu dezakeela uste izateak eragin du metodo formalak erabiltzea.

Horrenbestez, badirudi metodo formalak gero eta gehiago erabiliko direla softwarearen garapenean, eta tresna automatiko sendoagoak eta erabilerrazagoak eskainiko direla. Arlo honetan, azkarrago aurreratzeko eta emaitza

7. Windows Hardware Engineering Conference (WinHEC 2002) konferentziako sarrerako hitzaldian.

hobeak lortzeko, ezinbestekoa da software-ingeniaritzako prestakuntzan metodo formalen oinarriak lantzea, bai eta tresna eta aplikazio interesgarrienak ere.



1.1. irudia. Aholkatutako irakurtze-ordena.

1.2. LIBURUAREN DESKRIBAPENA

Liburu honen helburua programazioaren metodologia eta teknologiaren arloan funtsezkoak diren oinarrizko metodo formalez osatutako bilduma bat aurkeztea da, adibide eta ariketen bidez. Liburu honetan, programen eta datu-moten espezifikazioa, programa iteratibo eta errekurtsibo errazen egiaztapena, espezifikazio (edota programa) errekurtsiboak programa iteratibo bihurtzea, eta, aurre-ondoetako espezifikazioetatik abiatuz, bai programa iteratiboen bai errekurtsiboen eratorpena lantzen dira.

Liburu honetako edukiek, ebatzitako eta proposatutako ariketak barne,

Euskal Herriko Unibertsitatean hainbat hamarkadatan zehar eman den Programazioaren Metodologia izeneko irakasgaian dute jatorria. Gaur egun irakasgai hori Informatika Ingeniaritzako Graduan lehenengo mailako bigarren lauhilekoan ematen da. Gradu horretako lehenengo lauhilekoan, programazioko oinarrizko kontzeptuak azaltzen dituen beste irakasgai bat dute ikasleek. Liburu hau, Programazioaren Metodologia irakasgaiko testu-liburu bezala erabil daiteke. Urteetan zehar irakasgaia pixka bat aldatuz joan den arren, muinari eutsi egin zaio. Egun, irakasgaia muin horretara mugatuta dago, eta hori da hain zuzen ere liburu honetan era xehatuan aurkezten eta lantzen dena. Hala eta guztiz, denbora-urritasuna eta zailtasuna direla eta, liburu honetako alderdi edo atal batzuk ez dira aurkezten aipatutako irakasgai horretan.

Liburuko kapitulu bakoitzean, guztiz garatutako adibideak egoteaz gain, ebatzi gabe dauden ariketez osatutako hainbat atal ere badaude. Ebatzi gabeko ariketa horiek beren aurretik garatu diren adibideen zailtasun-maila bera dute gutxi gorabehera. Gainera, kapitulu bakoitzaren bukaeran kapituluan zehar aurkeztutakoa lantzeko aukera ematen duen ariketa-bilduma ere badago. Bestalde, kapitulu bakoitzak interesgarriak diren aipamen historikoak eta erreferentziak jasotzen dituen bibliografia-oharrei buruzko atala ere badu. Sarrerako kapitulu honetan egindako baieztapen batzuk era zehatzagoan dokumentatzeko balio dute kapitulu bakoitzeko erreferentziek.

Liburua irakurtzeko aholkatzen den ordena 1.1. irudian eskematizatu da. Horrela, 2., 3. eta 4. kapituluek programa agintzaileen —bai iteratiboen bai errekurtsiboen— espezifikazioari eta egiaztapenari buruzko funtsezko muina osatzen dute. Bestalde, 5. kapituluan datu-motak espezifikatzeko teknika formal bat jaso da. Hor azaldutakoa 6. kapituluan abiapuntu bezala erabiliko da, datu-moten gaineko eragiketen espezifikazioetatik abiatuz, eragiketa horiek inplementatzen dituzten programak transformazioen bidez nola lor daitezkeen azalduko baita. Bere aldetik, 7. kapitulua 2. eta 3. kapituluetan oinarrituta dago; izan ere, programa bat eta haren zuzentasunaren frogapena era sistematikoan eta zorrotzean (matematikoan) eraikitzen dituen diseinu-metodoaren oinarriak finkatzeko, programen zuzentasuna frogatzeko balio duen metodoa erabiltzen da. Bukatzeko, 8. kapitulua batez ere 4. kapituluan oinarrituta dago, baina 5. kapituluan aurkeztutako ekuazio bidezko espezifikazioak ere erabiltzen dira bertan.

1.3. ESKER ONAK

Liburu honi sorrera eman dion irakasgaia ematen unean-unean aritu diren Euskal Herriko Unibertsitateko Lengoaia eta Sistema Informatikoak Saileko kideei benetako esker ona adierazi nahi diegu. 80ko hamarkadaren hasieratik, ikasketa-plan desberdinak tarteko, irakasgaia bilakatuz eta aldatuz joan da, eta haren edukiak landuz eta hobetuz joan dira. Hainbeste urteetako ibilbidean, asko izan dira beren ekarria utzi duten pertsonak. Eskerrik asko horiei guztiei beren ekarpenengatik. Ikasleen parte-hartzea ere eskertzen dugu; izan ere, haien gogo biziarekin, galderekin eta iruzkin eta iradokizunekin, liburuaren sorrera bultzatzeaz gain, haren edukiak aberasten ere lagundu dute, bereziki adibideei eta ariketei dagokienez.