Развој софтвера 2

4. Кључне одлуке у конструкцији

4.1. Избор програмског језика

Ослобађањем мозга од свог непотребног посла, добра нотација омогућава да се концентришете на напредније проблеме, што доводи до повећања менталне снаге.
Пре откривања арапске нотације, множење је било тешко, а за дељење целих бројева је било неопходно активирати највеће математичке способности. Можда ништа у модерном свету не би толико запрепастило грчке математичаре као сазнање да велики део популације западне Европе може да извршава операције дељења јако великих бројева. Ова чињеница би њима изгледала скоро немогућа....
Наша модерна моћ лаког рада са фракцијама децималних бројева је скоро чудесан резултат постепеног откривања перфектне нотације.

- Алфред Норт Вајтхед

4.1. Избор програмског језика

Избор програмског језика у ком ће бити имплементиран систем је веома важна ствар за програмере, јер ће они бити урођени у тај језик од почетка до краја конструкције.

Студије су показале да избор програмског језика утиче на неколико начина на продуктивност програмера и на квалитет кода.

Програмери су много продуктивнији када користе језик који им је познат. Подаци из модела за процену Cocomo II указују да су програмери који су три или више година радили у неком програмском језику за око 30% продуктивнији од програмера који имају еквивалентно искуство а којима је тај програмски језик потпуно нов. Ранија студија коју је направио IBM је открила да су програмери који имају обимно искуство са неким програмским језиком више од три пута продуктивнији у односу на колеге који са тим језиком имају минимално искуство.

4.1. Избор програмског језика

Програмери који раде са програмским језицима високог нивоа постижу бољу продуктивност и квалитет у односу на оне који раде са програмским језицима ниског нивоа.

Језици као што су C++, Java, Smalltalk, Visual Basic, C#, Ruby су хваљени зато што повећавају продуктивност, поузданост, једноставност и разимљивост за фактор 5-15 пута у односу на језике ниског нивоа као што су асемблерски језик и C.

Надаље, језици вишег нивоа су изражајнији од језика нижег нивоа. Свака линија кога више говори. Табела 4-1 (на следећем слајду) приказује типичан однос између изражајности наредби неколико виших програмских језика у односу на еквивалентни код у програмском језику C.

Подаци из компаније IBM указују на још једну карактериситку која утиче на продуктивност: програмери који раде у језицима који се интерпретирају имају тенденцију да буду продуктивнији од оних који раде у језицима који се компајлирају (Jones 1986a).

4.1. Избор програмског језика

Табела 4-1. *Однос између наредби на високом програмском језику и еквивалентних наредби на програмском језику C*

Language	Level relative to C
C	1 to 1
C++	1 to 2.5
Fortran 95	1 to 2
Java	1 to 2.5
Perl	1 to 6
Smalltalk	1 to 6
SQL	1 to 10
Visual Basic	1 to 4.5

Извор: Подаци из књига Estimating Software Costs (Jones 1998) и Software Cost Estimation with Cocomo II *(Boehm 2000).*

4.1. Избор програмског језика

Неки језици су бољи од других у изражавању програмских концепата. Може се повући паралела између природних језика као што је српски језик и програмских језика као што су Јава и C++. У случају природних језика, лингвисти Сапир и Ворф су поставили хипотезу о постојању везе између изражајне моћи језика и способности да се разматрају одређена размишљања. Другим речима, хипотеза тврди да способност појединца да размишља зависи и од познавања речи које му омогућавају да изрази своје мисли.

На сличан начин програмери могу бити под утицајем језика у ком програмирају. Речи програмског језика које су доступне за изражавање програмерских мисли одређују како се изражавају мисли, па чак може одредити које се мисли могу изразити.

Бројни су докази ефекта који програмски језици имају на начин размишљања програмера. Обично прича иде на следећи начин: “Треба да пишемо нови систем у језику C++, а највећи број наших програмера нема много искуства у језику C++. Они су раније много радили у језику Fortran. Они као пишу C++ код, али се ту у ствари ради о замаскираном Fortran-у. У том случају су програмери сузили C++ како би емулирали лоше особине Fortran-а (као што су goto наредба и глобални подаци) и игнорисали богат скуп објектно-орјентисаних особина језика C++”.

4.1. Избор програмског језика

4.1.2. Описи језика

Следе описи најчешћих језика који се данас користе.

Ада

Ада је програмски језик високог нивоа и опште намене. Синтакса језика је заснована на Pascal-у. Настао је под окриљем Министарства одбране УСА, као језик који је веома погодан за уграђене системе који раде у реалном времену (енг. real-time and embedded systems).

Језик Ада истиче апстракцију података и сакривање информација и приморава програмера да раздвоји јеви и приватни део сваке класе и сваког пакета.

Име “Ада” је дато овом језику у знак почасти Ади Ловлес, математичарки која се сматра првим програмером на свету. Данас се Ада примарно користи у војним, свемирским и авио системима.

4.1. Избор програмског језика

Асемблерски језик

Асемблерски језик, или „асемблер“ је језик ниског нивоа у ком свака од наредби тог језика одговара тачно једној машинској инструкцији.

Будући да наредбе користе конкретне машинске инструкције, то је асемблерски језик специфичан за одређени процесор – на пример за конкретне Intel или Motorola процесоре. При томе треба имати у виду да се фамилије процесора дизајнирају тако да подржавају тзв вертикалну компатибилност. Тако, на пример, асемлерски програм писан за Intel 80386 процесор може без проблема и без икакве промене да се извршава на 80486 процесору или на Pentium-у.

Асемблер се сматра језиком друге генерације. Највећи број програмера га избегава, осим у случају када је неопходно померити границу у брзини извршавања или у величини кода.

4.1. Избор програмског језика

C

Програмски језик C је језик опште намене и средњег нивоа. Он је на почетку био придружен оперативном систему UNIX – развој језика је испреплетен са развојем оперативног система.

C има неке од особина језика високог нивоа, као што су структуирање података, структуирани ток контроле, машинска независност и богат скуп оператора. Он је још називан и „портабилни асемблерски језик“, зато што интензивно користи показиваче и адресе и зато што има неке конструкције ниског нивоа, као што је манипулација са битовима и зато што је слабо типизиран (бар прве верзије).

C је развијен у Bell Laboratories 70-тих година прошлог века. Оригинално је дизајниран за и коришћен код DEC PDP-11, где су и оперативни систем и C компаљер и UNIX апликативни програми сви били писани у C-у. Године 1988, креиран је ANSI стандард који кодификује C, а тај стандард је ревидиран 1999. C је био де факто стандард за програмирање микрорачунара, минирачунара и радних станица у 80-тим и 90-тим годинама прошлог века.

4.1. Избор програмског језика

C++

C++, објектно-орјентисани језик који је занован на језику C, је развијен у Bell Laboratories почетком 80-тих година двадесетог века. Сам језик C++, првобитно назван C са класама, се сматра језиком средњег нивоа, зато што комбинује и карактеристике високог нивоа и карактеристике ниског нивоа.

Поред тога што је компатибилан са језиком C, програмски језик C++ обезбеђује класе, полиморфизам, вишеструко наслеђивање, руковање изузетцима, шаблоне, а такође обезбеђује и робуснију проверу типа него што је то случај са програмским језиком C.

C++ је један од најпопуларнијих програмских језика који је имплементиран на огромном броју хардверских и софтверских платформи. Будући да садржи ефикасан компајлер који преводи до нивоа машинског кода, његови домени примене су системски софтвер, апликативни софтвер, погонски програми за уређаје (енг. device drivers), уграђени софтвер (енг. embedded software), серверске и клијентске апликације високих перформанси и офтвер за забаву, као што су видео игре.

4.1. Избор програмског језика

Smalltalk

Smalltalk је објектно-орјентисан, динамички типизиран, рефлективни програмски језик. Smalltalk је креиран као језик који подупире „нови свет“ рачунарства који објашњава појам „симбиоза човека и рачунара“. Он је дизајниран и креиран у оквиру Групе за истраживање учења (енг. Learning Research Group - LRG) у оквиру истраживачког центра Xerox PARC.
Smalltalk је имао утицај на шири свет програмирања у четири главна домена:

- Инспирисао је синтаксу и семантику код других програмских језика.
- Постао је прототип за модел израчунавања који је познат као прослеђивање порука (енг. message passing).
- Кориснички интерфејс WIMP (прозор-икона-миш-показивач) је инспирисао окружење са прозорима код PC рачунара крајем XX и почетком XXI века, исто као што је прво Macintosh радно окружење по изгледу скоро исто као MVC прозори код Smalltalk-80.
- Интегрисано радно окружење је постало модел за генерисање визуелних програмских алата који личе на Smalltalk дибагере.

Језици Python и Ruby су поново имплементирали неке од идеја Smalltalk-а.
Синтакса и понашање језика Objective-C је под великим утицајем Smalltalk-а.

4.1. Избор програмског језика

Cobol

Cobol је програмски језик који помало подсећа на енглески говорни језик. Овај језик је развијен у периоду 1959.-1961, за коришћење у Министарству одбране УСА.

Cobol се првенствено користи за пословне апликације, а по резултатима Феиман-а и Драјвера је до почетка 21-вог века био један од језика који се налазе у најширој употреби, тј. који су веома популарни.

Током година, Cobol је бивао допуњаван и усавршаван тако да укључи математичке функције и објектно-орјентисане могућности.

Реч Cobol је скраћеница која означава „Општи пословно-орјентисани језик“ (енг. Common Business - Oriented Language).

4.1. Избор програмског језика

Fortran

Fortran је први језик високог нивоа, који је увео идеје променљивих и петљи. Сама реч Fortran је изведена из речи „превод формула“ (енг. Formula translation).

Fortran је настао у 50-тим годинама двадесетог века.

Досад је имао већи број значајних ревизија, укључујући Fortran IV из 1961, те Fortran 77 из 1977, који је додао блоковски структуирану if-then-else наредбу, као и наредбе за манипулацију са знацима и стринговима. Године 1990, ревизија Fortran 90 додаје кориснички дефинисане типове података, показиваче, класе и богат скуп оператора над низовима.

Fortran се углавном користи у начним и инжињерским апликацијама.

4.1. Избор програмског језика

Јава

Јава је објектно-орјентисан језик чија је синтакса слична синтакси језика C и C++, а који је развијен од стане Sun Microsystems.

Програмски језик Јава је дизајниран тако да се извршава на свакој платформи, конвертовањем изворног Јава кода у тзв. бајт-код, који се потом извршава на било којој платформи у оквиру окружења познатог под називом Јава виртуелна машина.

Програмски језик Јава је широко прихваћен за коришћење у разним доменима примене, укључујући и веб апликације.

4.1. Избор програмског језика

C#

C# је језик опште намене, објектно-орјентисан, са синтаксом сличном синтакси језика C, C++ и Јава.

Рад у овом језику подржава велики број алата који су развијени на Microsoft платформама.

C# је један од језика у фамилији тзв. Microsoft .NET језика, у коју још спадају: Visual Basic .NET, C++ .NET, F#, Cobol .NET итд. Код њих се програм извршава слично као код Јаве: конвертује се изворни кода у тзв. управљани код (енг. manageable code), који се потом извршава на циљној Microsoft платформи у оквиру окружења познатог под називом заједничко извршавање језика (енг. Common Language Runtime - CLR).

4.1. Избор програмског језика

Basic

Оригинална верзија програмског језика високог нивоа, названог Basic, је развијена на колеџу Дармут 60-тих година прошлог века. Реч Basic представља скраћеницу за „Почетнички свенаменски симболички инструкцијски код“ (енг. Beginner’s All-purpose Symbolic Instruction Code).

Данашња инкарнација Basic-а, Microsoft-ов производ Visual Basic је објектно-орјентисан, визуелни програмски језик високог нивоа. Њега су дизајнирали у Microsoft-у ради креирања Windows апликација. Од тог времена, језик Visual Basic је проширен тако да подржава подешавања и проширења „класичних“ апликација као што је Microsoft Office, креирање апликација за мобилне уређаје, веб апликација и веб сервиса, као и других апликација.

4.1. Избор програмског језика

JavaScript

JavaScript је интерпретирани скрипт језик који је слабо повезан са језиком Јава.

Овај језик се примарно користи за додавање функционалности и он-лајн апликација на веб страну.

JavaScript садржи како елементе функционалног, тако и објектно-орјентисаног програмирања.

У последње време, захваљујући великој популарности који је стекао HTML 5, програмски језик JavaScript стиче велику популарност.

4.1. Избор програмског језика

Perl

Perl је језик за манипулацију стринговима, који је базиран на језику C и на неколико Unix корисничких алата, а који је креиран у Jet Propulsion Laboratories.

Perl се често користи за задатке системске администрације, као што су креирање скриптова, као и генерисање и процесрање извештаја.

Скраћеница Perl означава „Практичан језик за екстракцију и извештаје“ (енг. Practical Extraction and Report Language).

4.1. Избор програмског језика

PHP

PHP је скрипт-језик отвореног кода, са синтаксом која је слична језицима Perl, Bourne Shell, JavaScript и C. PHP се извршава на свим најпопуларнијим оперативним системима и извршава интерактивне функције на серверској страни.

PHP наредбе се обично умећу у веб старне ради приступа подацима из базе података и ради њиховог приказа на веб страни.

Скраћеница PHP је првобитно значила „лична матична веб страна“ (енг. Personal Home Page), али сада означава „ PHP: хипертекст процесор“ (енг. PHP: Hypertext Processor).

4.1. Избор програмског језика

Python

Python је интерпретиран, интерактиван, објектно-орјентисан језик који се фокусира на рад са стринговима.

Он је најчешће коришћен за писање скриптова и малих веб апликација. Надаље, језик Python такође садржи и подршку за креирање већих програма. Овај језик се извршава у већем броју различитих окружења.

4.1. Избор програмског језика

Ruby

Ruby је динамичан и рефлективан програмски језик општег типа, који комбинује синтаксу инспирисану са Perl-ом са карактеристикама које подсећају на Smalltalk. На дизајн језика су утицали и Eiffel и Lisp. Ruby је дизајнирао и развио Јухиро Мацумото средином 90-тих прошлог века у Јапану.
Ruby подржава више програмских парадигми укључујући функционално програмирање, објектно-орјентисано програмирање, императивни и рефлексивни приступ. Он такође саржи динамичке типове и аутоматско управљање меморијом.

У језику Ruby је свака вредност објекат, што укључује класе и примерке које неки други језици означавају примитивним (нпр. integer, boolean и "null"). Променљиве увек чувају референце на објекте. Свака функција је метод и методи су увек позивани над објектом. Ruby подржава наслеђивање са динамичним прослеђивањем, мешавину (енг. mixin) и јединичне (енг. singleton) методе – тј. методе којсу дефинисане за један јединствени примерак тј. објекат а нису дефинисане за класу. Иако Ruby не подржава вишеструко наслеђивање, класе могу да увезу модуле преко мешавина.

4.1. Избор програмског језика

SQL

SQL је де факто стандардни језик за упите, ажурирање и одржавање релациобе базе података. Скраћеница SQL означава „структурисани упитни језик“ (енг. Structured Query Language).

За разлику од других језика који су излистани у овој секцији, SQL је декларативни језик – што значи да он не дефинише на који начин ће се извршити операције, већ шта ће бити резултат који се добија извршењем операција.

4.1. Избор програмског језика

4.1.3. Табела за брзи избор језика

Табела 4-2 представља кратки, скицирани приказ језика који су погодни за различите сврхе. Њу не треба користити као замену за пажљиво одређивање који конкретни језик највише одговара датом пројекту. Као и у другим сличним случајевима, класификацијама се покрива широко поље, па препоруке из табеле која следи треба узети „са зрном соли“, нарочито ако се већ зна да постоје изузетци.

Табела 4-2. *Најбољи и најгори језици за различите врсте програма*

Kind of Program	Best Languages	Worst Languages
Command-line processing	Cobol, Fortran, SQL	-
Cross-platform development	Java, Perl, Python	Assembler, C#, Visual Basic
Database manipulation	SQL, Visual Basic	Assembler, C
Direct memory manipulation	Assembler, C, C++	C#, Java, Visual Basic

4.1. Избор програмског језика

Distributed system	C#, Java	-
Dynamic memory use	C, C++, Java	-
Easy-to-maintain program	C++, Java, Visual Basic	Assembler, Perl
Fast execution	Assembler, C, C++, Visual Basic	JavaScript, Perl, Python
For environments with limited memory	Assembler, C	C#, Java, Visual Basic
Mathematical calculation	Fortran	Assembler
Quick-and-dirty project	Perl, PHP, Python, Visual Basic	Assembler
Real-time program	C, C++, Assembler	C#, Java, Python, Perl, Visual Basic
Report writing	Cobol, Perl, Visual Basic	Assembler, Java
Secure program	C#, Java	C, C++
String manipulation	Perl, Python	C
Web development	C#, Java, JavaScript, PHP, Visual Basic	Assembler, C

Неки од језика једноставно не подржавају одређене врсте програма, па такви нису приказани као „најгори“.

4.2. Конвенције у програмирању

У високо квалитетном софтверу се може уочити релација између концептуалног интегритета архитектуре и њене имплементације на ниском нивоу. Имплементација мора да буде конзистентна са архитектуром која је води, а мора да буде и интерно конзистентна.

То је тачка на којој се појављују конструкционе смернице која се односе на имена променљивих, имена класа, имена рутина, конвенције форматирања и конвенције за коментаре. У сложеном програму, архитектонске смернице дају програму структурни баланс, а конструкционе сменице обезбеђују хармонију на нижем нивоу, тако што артикулишу сваку класу као поуздани део свеукупног дизајна.

Ма који велики програм захтева контролну структуру која унифицира детаље који се односе на програмски језик у оквиру тог програма. Део лепоте велике структуре је начин на који детаљни делови те структуре носе импликације њене архитектуре. Када не би било унифицирајуће исциплине, креација би представљала збрку лоше координисаних класа и аљкавих стилских варијација. Шта би се догодило ако имате предива дизајн за осликавање, али један део је класичан, други импресионистички, а трећи кубистички? Резултат неће имати концептуални интегритет, без обзира колико прецизно пратили тај диван дизајн, већ ће превасходно личити на колаж. И програму, такође, треба интегритет на нижем нивоу.

4.2. Конвенције у програмирању

Кључна ствар:

Пре него што почне конструкција, треба експлицитно (у виду докумената тј. смерница) исказати све конвенције у програмирању које ће се користити. Те конвенције су на веома детаљне и на веома ниском нивоу, тако да је скоро немогуће да се оне укључе у софтвер пошто део софтвера буде написан. У овој и презентацијама које следе су описани детаљи конвенција програмирања.

4.3. Ваша позиција на технолошком таласу

Стив МекКонел: „Током свог рада сам видео како се звезда PC рачунара подиже, док звезда mainframe рачунара залази иза хоризонта. Видео сам како програми са графичким интерфејсом мењају терминалске програме. Видео сам како се Веб уздиже док Windows полако пада. Може се само претпоставити да ће у тренутку читања овог текста нека нова технологија бити на врхунцу, а да ће веб програмирање, онакво каквог ја данас (2004) познајем, бити на одласку“.
Ови технолошки циклуси, или таласи, утичу на различите програмске праксе у зависности од тога која је ваша позиција на том таласу.
У зрелим технолошким окружењима, тј. на крају таласа, као код веб програмирања половином прве деценије 21-вог века, се извлачи корист из богате инфраструктуре софтверског развоја. Окружења на крају таласа нуде избор између више програмских језика, темељну проверу грешака sa код писан у изабраном језику, моћне алате за дебагирање и аутоматизовану и поуздану оптимизацију перформанси. Компајлери скоро па немају багова. Постоји обимна и поуздана документација која олакшава рад са програмерским алатима. Алати су интегрисани, па се и база и креирање корисничког интерфејса и програмирање и извештаји и пословна логика могу реализовати у једном окружењу. Ако дође до проблема, одмах се можу наћи, коришћењем ресурса са веба и FAQ-а, како превазићи каприце коришћених алата.

4.3. Ваша позиција на технолошком таласу

Код окружења на почетку таласа, као што је веб програмирање средином 90-тих година прошлог века, ситуација је управо супротна. Доступан је само мали број избора за програмски језик и ти језици имају тенденцију да буду баговити и слабо документовани. Програмери проводе значајан део времена покушавајући да схвате како све то ради, уместо да пишу нови код. Програмери такође проводе безбројне сате у раду на заобилажењу багова у језицима, у оперативном систему и у другим алатима које користе.

Алати за програмирање у окружењима на почетку таласа имају тенденцију да буду примитивни.
Може се огодити да дебагери уопште не постоје, а оптимизатори компајлера само буде одсјај у оку програмера. Произвђачи често ревидирају своје преводиоце, испоручују нову верзију и изгледа да свака нова верзија неће да ради са значајним деловима већ написаног кода. Алати нису интегрисани, па се ради са различитим алатима за базу, за креирање корисничког интерфејса, за програмирање, за извештаје и за пословну логику. Алати често нису компатиблини, па се може очекивати да програмер треба да уложи значајан напор само да би очувао постојећу функционалност у ситуацији честих промена компајлера, библиотека и алата.

4.3. Ваша позиција на технолошком таласу

Аутоматизовано тестирање је изузетно корисно зато што помаже много бржем откривању грешака до којих долази услед промена развојног окружења. Ако се наиђе на проблем, иако постоји референтна литература о томе (у неком облику), тлитература није увек поуздана и сваки пут када се наиђе на проблем, ситуација изгледа тако тешка као да се нико пре тога није срео са њом. Може изгледати као да претходни ставови сугеришу да треба избегавати програмирање у окружењима на почетку таласа, али то нипошто није случај. Неки од најиновативнијих апликација потичу од програма са почетка технолошког таласа, као што су Turbo Pascal, Lotus 123, Microsoft Word и прегледач Mosaic.

Важно је истаћи да ће начин на који се организује рад програмера зависити од позиције на технолошком таласу. Ако се налазите при крају таласа, можете планирати да највећи део дана проведете у стабилном писању нове функционалности. Ако сет на почетку таласа, можете претпоставири да ћете провести значајан део свог времена покушавајући да схватите недокументоване карактеристике програмског језика који користите, дебагирајући грешке које очито потичу од грешака у библиотеци кода, мењајући код тако да поново проради уз нову верзију испоручене библиотеке и сл.

4.3. Ваша позиција на технолошком таласу

Када се нађете у позицији да радите са примитивним окружењем, схватите да вам у том случају добре програмерске праксе могу помоћи чак и више него што је то случај у зрелим окружењима. Као што је још 1981и истакао Дејвид Грис, ваши програмерски алати не смеју да одређују начин на који мислите о програмирању.

Грис говори о разлици између програмирања *у језику* (енг. programming in a language*)* и програмирања *кроз језик* (енг. programming into a language).

Програмери који програмирају „у“ језику ограничавају своје мисли на конструкције које тај конкретни језик директно подржава. Ако су алати језика примитвини, онда су и програмерова размишљања такође примитивна. Програмери који програмирају „кроз“ језик ће прво да одлуче које мисли желе да искажу, а тек потом ће да одреде како да искажу те мисли коришћењем алата које обезбеђује дати програмски језик.

Илустрације програмирања *у језику* и *програмирања кроз језик* у контексту Visual Basic апликација и веб апликација.

4.4. Избор најважнијих пракси конструкције

Део припрема за конструкцију је одређивање које ће од многобројних добрих пракса бити истакнуте.

Неки пројекти користе програмирање у паровима и разој вођен тестовима (енг. test driven development), док други користе ппјединачно програмирање и формалне инспекције. Свака од техника може добро функционисати, у зависности од конкретних околности датог пројекта.

Листа за проверу која следи сумарно приказује конкретне праксе за које се пажљиво треба одлучити да ли ће бити укључене у конструкцију или искључене из ње.

4.4.1. Листа за проверу: најважније праксе конструкције

Coding

- Have you defined coding conventions for names, comments, and formatting?
- Have you defined specific coding practices that are implied by the architecture, such as how error conditions will be handled, how security will be addressed, and so on?
- Have you identified your location on the technology wave and adjusted your approach to match? If necessary, have you identified how you will program *into the language rather than being limited by programming in it?*

4.4.1. Листа за проверу: најважније праксе конструкције

Coding

- Have you defined coding conventions for names, comments, and formatting?
- Have you defined specific coding practices that are implied by the architecture, such as how error conditions will be handled, how security will be addressed, and so on?
- Have you identified your location on the technology wave and adjusted your approach to match? If necessary, have you identified how you will program *into the language rather than being limited by programming in it?*

Teamwork

- Have you defined an integration procedure, that is, have you defined the specific steps a programmer must go through before checking code into the master sources?
- Will programmers program in pairs, or individually, or some combination of the two?

4.4.1. Листа за проверу: најважније праксе конструкције

Quality Assurance

- Will programmers write test cases for their code before writing the code itself?
- Will programmers write unit tests for the their code regardless of whether they write them first or last?
- Will programmers step through their code in the debugger before they check it in?
- Will programmers integration-test their code before they check it in?
- Will programmers review or inspect each others' code?

Tools

- Have you selected a revision control tool?
- Have you selected a language and language version or compiler version?
- Have you decided whether to allow use of non-standard language features?
- Have you identified and acquired other tools you'll be using—editor, refactoring tool, debugger, test framework, syntax checker, and so on?

Рекапитулација

● Сваки програмски језик има своје предности и мане. Будите упознати са предностима и манама програмског језика који користите.

● Установите конвенције програмирања пре него што почнете са самм програмирањем. Скоро је немогуће накнадно мењати код тако да се уклопи у конвенције.

● Постоји више конструкционих пракси и не могу се све корситити у датом пројекту. Потребно је да се пажљиво изабере које су од пракси најбоље прилагођене датом пројекту.

● Позиција у технолошком таласу одређује који од приступа ће бити ефикасан, или чак уопште могућ. Треба одредити позицију на таласу, и сходно томе прилагодити планове и очекивања.