Ján Hanák

Ako sa stať softvérovým vývojárom

Microsoft® Visual Studio® 2010

Microsoft®

Ján Hanák

Ako sa stať softvérovým vývojárom

Microsoft Slovakia
2010

Obsah

Predhovor

Vážení čitatelia,

táto príručka vám poskytne rady, usmernenia a vhodné postupy na to, aby sa z vás mohli stať softvéroví vývojári. Príručku sme sa snažili navrhnúť a následne aj realizovať tak, aby dokázala osloviť čo možno najširšie cieľové publikum záujemcov. Predpokladáme, že publikácia bude nápomocná študentom stredných škôl, poslucháčom vysokých škôl a rovnako aj odborníkom z praxe, ktorí by radi zmenili svoje profesijné zaradenie.

V publikácii sa venujeme nasledujúcim témam:

1. **Ako sa stavia softvér.** V tejto kapitole analyzujeme kompletný životný cyklus softvérového produktu, takže čitatelia sa dozvedia, ako sa v skutočnosti budujú moderné počítačové aplikácie a systémy.

2. **Čo je super na tom, keď ste softvérovým vývojárom.** Tvorba softvéru je jednou z najdynamickejšie sa rozvíjajúcich oblastí ľudských činností na tejto planéte. Vydajte sa s nami na exkurziu po všetkých výborných veciach, ktoré sú typické pre softvérového vývojára.

3. **Čo urobiť pre to, aby ste sa stali softvérovými vývojármi.** Nech už ste študentom strednej školy, vysokoškolákom alebo pracovníkom v IT firme, poradíme vám, ako maximalizovať efektivitu vášho úsilia v záujme dosiahnutia vysnívaného cieľa – práce softvérového vývojára.

4. **Ako efektívne študovať algoritmizáciu a programovanie.** Každý softvérový vývojár má algoritmizáciu a programovanie v malíčku. Aby ste ovládli tieto technické disciplíny s pôsobivou eleganciou, pripravili sme pre vás metodiku, ktorá garantuje váš rýchly progres.

5. **Ako prebieha pracovný pohovor na pozíciu softvérového vývojára.** Prišiel deň D a vy máte absolvovať pracovný pohovor na pozíciu softvérového vývojára. Ako to bude vyzerať a ako sa najlepšie uviesť pred novým zamestnávateľom – to všetko zistíte po preštudovaní tejto kapitoly.

6. **Absolvovanie a vyhodnotenie ukážkového písomného testu algoritmizácie a programovania v jazyku C++, ktorý bol použitý pri pracovnom pohovore na pozíciu softvérového vývojára.** Aby sme budúcim adeptom „počítačovej mágie" ukázali, s čím sa môžu pri pracovnom pohovore stretnúť, uvádzame ukážkový technický test programátorských zručností v jazyku C++. Test podávame aj s riešením a vysvetlením správnych odpovedí. Nakoniec umožňujeme, aby sa čitatelia mohli sami otestovať a zhodnotiť svoj výkon.

Z vlastných skúseností vieme, že dostať sa na špičku moderného informatického sveta nie je vôbec jednoduché – úspešné absolvovanie tejto cesty vyžaduje nesmiernu dávku energie, úsilia a času. Keď ste však pevne presvedčení o tom, že informatika a vývoj softvéru je pre vás to pravé, sme pripravení vám poskytnúť maximálnu možnú podporu, aby ste boli úspešní.

Ján Hanák, Bratislava august 2010

1 Ako sa stavia softvér

Pracovnou náplňou softvérového vývojára je vytváranie kvalitného softvéru, ktorý plní požiadavky svojich finálnych používateľov. Každý adept na remeslo softvérového vývojára preto musí najskôr pochopiť tú najdôležitejšiu vec – ako sa v skutočnosti softvér stavia. Dobrou správou je, že stavba softvéru nie je žiadna mágia. Skôr by sme mohli povedať, že sa jedná o dobre premyslený proces, ktorého štádiá sú vo veľkej miere automatizovateľné prostredníctvom programov na to určených. Moderná informatika definuje množstvo metodík, ktoré sa koncentrujú na to, ako jednotlivé etapy stavby softvéru zvládnuť čo najefektívnejšie a najrýchlejšie. Vo všeobecnosti môžeme proces stavby softvéru diferencovať do nasledujúcich štádií:

1. **Vízia.** Začiatočná fáza celého procesu je rýdzo invenčná – ako softvéroví nadšenci máme nápad, myšlienku či ideu budúceho programu, ktorý by sme chceli vlastnými silami zhotoviť. Pritom existuje viacero aspektov, ktoré môžu našu víziu ovplyvniť. Napríklad môžeme chcieť vytvoriť softvér, ktorý bude účinne automatizovať úlohy, ktoré sme museli dosiaľ vykonávať manuálne. Alebo by sme radi prišli so softvérom, ktorý prinesie zatiaľ nevídané riešenie. Termín „riešenie“ použitý v predchádzajúcej vete, má veľmi široký akčný rádius. Riešením môže byť trebárs nový expertný systém, ktorý dokáže inteligentne diagnostikovať choroby pacientov, a to v kratšom čase, s vyššou spoľahlivosťou a s menšími nákladmi ako súčasné expertné systémy. Ďalším riešením môže byť sofistikovaný grafický subsystém herného stroja počítačovej hry budúcnosti. Takýto grafický subsystém dokáže generovať hladkú grafiku vo fotorealistickej kvalite, čím umožní hráčom prežiť vskutku hlboký vizuálny zážitok z hrania hry. Riešením je aj vynájdenie nového spôsobu, akým môžu ľudia vo virtuálnom svete zdieľať svoje zážitky a komunikovať s ostatnými ľuďmi. Ako riešenie by sme v neposlednom rade označili aj zbrusu novú koncepciu integrovaného operačného systému automobilov, ktorý svojej posádke poskytne v hocakom čase a priestore všetky relevantné služby.

2. **Analýza požiadaviek finálnych používateľov.** Softvérový produkt nikdy neexistuje vo vákuu, ale je tvorený pre vopred vymedzenú cieľovú skupinu osôb. Tieto osoby budeme označovať termínom „finálni používatelia softvéru". Postupujeme pritom takto: najskôr si vytýčime množinu finálnych používateľov, na ktorú bude náš budúci softvérový produkt zacielený. Charakter cieľového segmentu používateľov je variabilný, no vždy medzi ním a budovaným softvérom existujú priame interakčné väzby.

   Ak bude naším cieľom tvorba nového medicínskeho expertného systému na podporu diagnostiky onkologických ochorení, našimi finálnymi používateľmi budú lekári, onkologickí špecialisti a samozrejme ich pacienti. Pri vývoji softvéru, ktorý bude riadiť elektronický stabilizačný systém osobných automobilov, budeme úzko spolupracovať s elektrotechnikmi a strojárskymi pracovníkmi, ktorí konštruujú dopravné zariadenia. Finálnymi používateľmi našej počítačovej hry budú (v závislosti od žánru) zrejme tínedžeri a mladí ľudia, ktorí sa radi ponárajú do tajov virtuálnych počítačových svetov.

   Keď sme určili segment finálnych používateľov pre vytváranú softvérovú aplikáciu, pokračujeme získaním ich ašpirácií. Zisťovanie požiadaviek finálnych používateľov sa môže diať osobne alebo neosobne (sprostredkovane). Osobná forma je vhodná vtedy, keď máme možnosť sa priamo stretnúť s výberovým súborom finálnych používateľov (v tomto smere kladieme dôraz na to, aby náš výberový súbor používateľov zodpovedal optimálnemu štatistickému rozdeleniu – vďaka tomu budeme schopní na preferencie jeho členov aplikovať štandardné techniky štatistickej indukcie). V prípade, keď nie je možné uplatniť metódu osobného kontaktu s vybranou skupinou finálnych používateľov, robíme tak sprostredkovane. Za týchto podmienok využívame najmä elektronické mechanizmy dopytovania a vyhodnocovania požiadaviek kontaktovaných osôb.

   O tom, že analyzovať požiadavky finálnych používateľov nášho softvéru je veľmi dôležité, nie je samozrejme žiadnych pochýb. Otázkou však zostáva, ako veľmi je nutné finálnych používateľov do tohto procesu zainteresovať. Pri softvérových

produktoch, ktoré sú vyvíjané priamym zadaním klientom, je rozhodne potrebná jeho 100% účasť v procese zisťovania požiadaviek. (Ako vieme, skutočnosť, že nás priamo osloví klient, aby sme pre nebo zostavili počítačovú aplikáciu, ešte nemusí znamenať, že má presnú predstavu o aplikácii, ktorú potrebuje.) Za istých okolností vie vývojový tím špecifikovať väčšinu programových rysov a ďalších čŕt budúceho softvérového produktu. V takýchto kontextoch je interakcia s finálnymi používateľmi stále významná, avšak ide skôr o doladenie už známych požiadaviek ako o ich kompletnú kategorizáciu.

3. **Konkurenčná analýza.** Konkurenčná analýza predstavuje analýzu konkurenčného prostredia softvérového produktu. Menej formálne, ide nám o vyhľadanie softvérových aplikácií podobného, respektíve príbuzného zamerania, ktoré sa aktuálne vyskytujú na počítačovou trhu. V praxi sa špecificky orientujeme najmä na analýzu priamych konkurentov – to sú programy, s ktorými bude naša budúca aplikácia súperiť. V trhovej ekonomike sú vzťahy medzi softvérovými firmami, ich produktmi a finálnymi používateľmi dané voľným trhom, ktorý má obvykle formu monopolistickej konkurencie (zjednodušene môžeme konštatovať, že podstatou monopolistickej konkurencie je nekooperatívna hra mnohých subjektov, v ktorej súťaží početná skupina počítačových firiem o záujem zo strany používateľov). Konkurenčná analýza je jednou z metód výskumu trhu. Cieľom výskumnej činnosti je nájdenie a usporiadanie konkurenčných softvérových produktov podľa ich sily vo vzťahu k našej aplikácii. Pojem „sila konkurenčného produktu" je však pre potreby bližšej analýzy vágny, a preto sa odporúča ho čo možno najviac konkretizovať kvantifikáciou.

   Otázkou však je, ako vlastne môžeme merať silu počítačových programov iných výrobcov. Prvá metóda, ktorá nás zrejme napadne, bude empirická analýza – jednoducho si požadovaný konkurenčný produkt vyskúšame a zhodnotíme jeho kvantifikovateľné znaky. K tým patrí napríklad početnosť funkcií a programových rysov, rýchlosť odozvy na vstupy používateľa či ergonomická prívetivosť. Áno, spektrum kvantifikovateľných znakov je veľmi široké, no my sa ho snažíme rozumne zúžiť, a to tak, že sa sústredíme len na signifikantné kvantifikovateľné

znaky softvérovej aplikácie. Každý typ počítačového softvéru má svoje vlastné signifikantné znaky, ktoré možno kvantifikovať. V drvivej väčšine prípadov platí, že finálni používatelia nakupujú také softvérové produkty, ktorých signifikantné kvantifikovateľné znaky im prinášajú maximálnu možnú mieru uspokojenia (ide teda o maximálne naplnenie potrieb, ktoré používatelia od softvéru očakávajú).

Rýdzo ekonomický pohľad na IT svet nám poskytne ďalšiu metódu na meranie konkurenčnej sily počítačových programov – ich trhový podiel. Táto informácia vypovedá o hĺbke penetrácie vybraného softvérového produktu na trhu so všetkými finálnymi používateľmi, prípadne na trhu s dedikovanými finálnymi používateľmi.

Ak dĺžka vývoja plánovanej počítačovej aplikácie neprekročí rok, je analýza súčasného konkurenčného prostredia postačujúca. V prípade softvéru, ktorého výroba si vyžiada viac času, je rozumné spojiť analýzu súčasného konkurenčného prostredia s predikciou pravdepodobného budúceho vývoja tohto prostredia. Takéto inteligentné správanie nám zaručí, že náš produkt bude konkurencieschopný aj po svojom dokončení a uvedení na počítačový trh. Odvetvím informatiky, v ktorom producenti softvéru musia rátať s budúcnosťou, sú napríklad počítačové hry. Ako by nám povedali herní vývojári, je len veľmi málo moderných hier, ktoré možno zhotoviť za čas kratší, ako sú 2 roky. Zvyčajne trvá vývoj hry 3 a viac rokov, čím sa produkcia počítačových hier zaraďuje k projektom so strednodobou životnosťou. Rozumná spoločnosť, ktorá sa chystá vytvoriť závodnú počítačovú hru, nemôže ako bernú mincu brať aktuálne závodné tituly iných herných firiem. Ak by sa tak predsa len stalo, firma by po troch rokoch uviedla na trh hru, ktorá by už bola technologicky zastaraná. O takejto firme by sme asi povedali, že podľahla „informačnej myopii", teda že svoj softvérový produkt plánovala len krátkozrako.

4. **Návrh a modelovanie softvérového produktu.** Vo chvíli, keď je analytická časť životného cyklu softvérového produktu hotová, prichádzajú na rad softvéroví architekti a IT analytici, ktorí majú jeden spoločný cieľ: na základe požiadaviek

finálnych používateľov navrhnúť a vymodelovať celý skelet budúcej počítačovej aplikácie. Moderná informatika definuje nemálo metodík, ktoré zainteresovaným členom softvérového tímu radia, ako zvládnuť túto etapu rýchlo, kvalitne a s dôrazom na dodržanie projektových míľnikov. V súčasnosti je hlavným prúdom objektové modelovanie, na ktoré vzápätí (v implementačnej fáze) nadväzuje objektové programovanie softvérového produktu. Výsledkom práce analytikov sú viacvrstvové modely softvérového produktu. Tieto modely zachytávajú nasledujúce pohľady na vyvíjanú počítačovú aplikáciu:

- **Statický pohľad na aplikáciu.** Statický pohľad prináša informácie o hierarchii komponentov, tried a vzájomných vzťahov medzi týmito entitami. Statický model softvéru je základným dokumentom, podľa ktorého budú programátori konštruovať jednotlivé časti počítačového programu.

- **Dynamický pohľad na aplikáciu.** Dynamický pohľad na aplikáciu nám vraví, ako sa aplikácia správa v procese interakcie s používateľom. Súčasné programy sú vždy implementované ako stavové stroje, takže je užitočné (a samozrejme z hľadiska korektnej funkcionality absolútne nutné) vždy jednoznačne poznať priebehy stavov, v ktorých sa aplikácia môže nachádzať. Používatelia sú rôzni, a preto môžeme oprávnene očakávať, že prechod rôznych používateľov počítačovým programom bude odlišný. Softvéroví architekti neraz definujú „prechod používateľa programom" termínom „aktuálny používateľský profil".

Úlohu softvérových architektov môžeme elegantne prirovnať k pozícii stavebných architektov. Podobne, ako stavební architekti tvoria modely (technické výkresy a potrebnú dokumentáciu) pre svoje budúce stavby a budovy, softvéroví architekti pripravujú modely pre budúce počítačové programy a informačné systémy. Obe skupiny kreatívnych pracovníkov uskutočňujú svoju prácu na vysokej úrovni abstrakcie. Po objektívnom posúdení môžeme vyhlásiť, že miera pracovnej abstrakcie softvérových architektov je predsa len vyššia. Je to

spôsobené tým, že ich diela majú (na rozdiel od diel bežných architektov) nehmotnú povahu.

5. **Implementácia softvérového produktu.** Modely budúceho softvérového produktu, príslušnú dokumentáciu, ako aj iné spriaznené materiály preberajú od softvérových architektov programátori. Úlohou programátorov je navrhnuté modely previesť do formy zdrojového kódu počítačového programu. Na vývoji takmer všetkých moderných počítačových programov participujú variabilne početné programátorské tímy. Podľa typu, robustnosti a zamerania vytváraného softvéru sa môže počet vývojárov pohybovať v desiatkach, stovkách alebo až tisíckach. So vzrastajúcim počtom programátorov dochádza ku kreácii viacerých tímov, ktoré majú delegovanú zodpovednosť za vopred špecifikované množiny úloh. Programátorské tímy pôsobia ako samostatné pracovné jednotky, avšak, prirodzene, existujú medzi nimi úzke kooperačné väzby. Podotknime, že maticová organizačná štruktúra „svedčí" tímom softvérových vývojárov azda najviac (a uľahčuje aj ich súbežný manažment).

   Aby mohli vývojári naprogramovať softvér, musia vedieť nielen programovať, ale aj čítať analytické modely, ktoré im poskytnú softvéroví architekti. Programátori používajú pri svojej práci programovací jazyk, prípadne súpravu viacerých programovacích jazykov, pomocou ktorých syntakticky implementujú entity a algoritmy, ktoré s týmito entitami manipulujú. Programovacích jazykov, s ktorými sa budúci adepti programátorského remesla môžu v praxi stretnúť, nie je rozhodne málo (ich počet sa pohybuje v desiatkach). Pokiaľ sa obmedzíme len na hybridné programovacie jazyky s podporou objektovej paradigmy vývoja počítačového softvéru, tak môžeme našu analýzu zamerať na nasledujúce jazyky:

   - **C++.** Jazyk C++ bol vytvorený v 80-tych rokoch 20. storočia ako hybridný jazyk, v ktorom môžeme programovať štruktúrovane aj objektovo. Jazyk C++ je nasadzovaný predovšetkým do robustných projektov, z ktorých mnohé patria k takzvaným kritickým aplikáciám. C++ má povesť exkluzívneho, ba až

„kráľovského" jazyka. Ako mnohí ostrieľaní programátori radi vravia: „Ak zvládnete C++, tak zvládnete všetko".

- **C#.** Jazyk C# je dielom spoločnosti Microsoft a hoci svetlo sveta uzrel len v roku 2002, najmä v komerčnej sfére sa teší veľkej obľube. Ako hovoria softvéroví inžinieri z Microsoftu: „C# v sebe spája len to najlepšie z jazykov Visual Basic a C++". Pravda je, že väčšina všetkých softvérových projektov na platforme Microsoft .NET je implementovaných v jazyku C#. Tento jazyk patrí k skupine objektových jazykov, disponuje košatou jazykovou špecifikáciou a prvkami funkcionálneho programovania. Rovnako dobre si rozumie s paradigmou paralelného programovania.

- **Visual Basic.** Podobne ako C#, aj jazyk Visual Basic je vynálezom firmy Microsoft. Ako nástupca „starého dobrého" Basicu je Visual Basic považovaný za jednoduchý jazyk s prehľadnou a najmä pre začiatočníkov v IT svete intuitívnou syntaxou. Od roku 2002 existuje „dotnetová" verzia Visual Basicu, takže aj fanúšikovia Visual Basicu môžu ťažiť zo silných stránok platformy Microsoft .NET. Visual Basic je rovnako funkčne vyspelý ako C# a ako vieme, oba jazyky budú v rámci stratégie koevolúcie funkčne ekvivalentné aj vo svojich nastávajúcich verziách.

- **Java.** Programovací jazyk Java priviedla na svet firma Sun v roku 1995. Java sa profilovala ako objektovo orientovaný jazyk, ktorého základy spočívali na rovnomennej platforme. (Pre prehľadnosť dodajme, že termín „Java" je preťažený, pretože reprezentuje jednak programovací jazyk a rovnako aj platformu, na ktorej sú spracúvané aplikácie v tomto jazyku vytvorené.) Vďaka multiplatformovej povahe kódu je možné programy napísané v Jave vykonávať na rôznych operačných systémoch a hardvérových zariadeniach. Java je nasadzovaná aj v oblasti webových aplikácií, či aplikácií (a hier) pre mobilné telefóny.

Ako zvykneme hovoriť, voľba programovacieho jazyka je v dnešných časoch skôr otázkou životného štýlu programátora ako čohokoľvek iného. V prípade firmy Microsoft a jej platformy .NET sú najfrekventovanejšími jazykmi C# a Visual Basic. V komerčnej praxi však majú svoje nezastupiteľné miesto tiež jazyky C++ a Java. Na druhej strane, voľbu programovacieho jazyka do značnej miery ovplyvňuje aj charakter vyvíjaného softvérového produktu. Hoci vieme, že väčšinu programov môžeme zhotoviť s ľubovoľným z uvedených programovacích jazykov, niekedy musíme siahnuť po iných programovacích jazykoch. Ak by sme napríklad chceli rýchlo vyvinúť matematickú knižnicu, ktorá by realizovala bezpečné paralelné matematické výpočty, s výhodou by sme uplatnili funkcionálny programovací jazyk, trebárs F# alebo Haskell.

Len veľmi málo komerčných softvérových produktov má monolitickú internú štruktúru. Ak nie je určené inak, interná štruktúra moderných počítačových aplikácií je komponentová. Na program tak nahliadame ako na inteligentne prepojenú sústavu komponentov, ktoré spočívajú na objektových základoch. Komponenty predstavujú softvérové jednotky, ktoré na vysokej úrovni abstrakcie poskytujú služby svojim klientom. Program obsahuje jeden hlavný (bázový) komponent a konečnú a neprázdnu množinu doplnkových (komplementárnych) komponentov. Každý komponent sprístupňuje dobre navrhnuté verejné rozhranie a je zodpovedný za uskutočňovanie istého spektra softvérových činností. Rozhrania jednotlivých komponentov sú unifikované, čo zvyšuje mieru ich interoperability (vzájomnej spolupráce). S príchodom viacjadrových procesorov je čoraz väčšia pozornosť venovaná paralelnému programovaniu. Cieľom paralelného programovania je vyvíjať taký softvér, ktorý dokáže hladko a bez vzniku nepriaznivých kolíznych stavov využiť všetku dostupnú výpočtovú kapacitu hocako výkonného počítačového systému. Paralelné programovanie „chutí najlepšie“ v súčinnosti s objektovým, komponentovým alebo funkcionálnym programovaním. Ideálne je, keď sú komponenty programu navrhnuté tak, že dokážu delegované výpočtové procesy realizovať súbežne.

Ani samotná implementačná fáza softvérového produktu nie je monolitická. V skutočnosti je naplánovaná s viacerými míľnikmi rôznej dôležitosti a dosiahnuteľnosti. Náležité a dochvíľne plnenie stanovených míľnikov je významné, či už z hľadiska psychologického, alebo ekonomického. Ako je známe, ľudia sú spokojní vtedy, keď vidia jasný progres vo svojej pracovnej činnosti. Podobne, plnenie míľnikov načas a bez zbytočných odkladov garantuje efektívne využitie peňažných prostriedkov alokovaných na realizáciu celého softvérového produktu.

6. **Ladenie a testovanie softvérového produktu.** Pod ladením počítačovej aplikácie máme na mysli iteratívny proces, ktorého cieľom je odhaliť všakovaké softvérové chyby vo vyvíjanom programe. Programátori sú si vedomí faktu, že v ich fragmentoch zdrojových kódov sa môžu tu a tam vyskytovať chyby rôzneho druhu (typicky ide o chyby syntaktické a logické). Ladením zbavujeme program chýb väčšej, strednej a menšej dôležitosti. Snažíme sa eliminovať všetky možné chyby, ktoré boli detegované v režime návrhu (pri preklade kódu) i behu (pri spracovaní kódu) programu. Pochopiteľne, ladenie si vyžaduje zmeny v kódovej základni programu. To znamená, že etapa ladenia softvéru sa do značnej miery prekrýva s implementačnou etapou (spravidla však ide len o modifikáciu existujúceho kódu, len vo výnimočných prípadoch je nutná tvorba nového kódu).

   Testovaním kontrolujeme, či program produkuje správne výstupy a preukazuje validné a očakávané správanie. Rovnako overujeme, či sú výstupy programu za každých okolností deterministické. Kritérium deterministických výstupov je nutné citlivo analyzovať predovšetkým pri softvérových produktoch, ktoré niektoré (alebo všetky) svoje činnosti vykonávajú paralelne. Je známe, že nesprávne zavedenie vybraných paralelných techník môže zapríčiniť vážne chyby či dokonca až nemožnosť spracovania kódu programu.

   Dobrou správou je, že procesy súvisiace s ladením a testovaním softvérového produktu sú úplne automatizovateľné. Vývojári a testovací pracovníci (alebo tiež testovači, ako sú títo ľudia čoraz častejšie nazývaní) majú k dispozícii nástroje na

profilovanie programu, verifikáciu jeho validity a deterministického správania. Nástroje tohto razenia si poradia aj s analýzou výkonnostných metrík programu.

Ladenie, testovanie a úprava kódu je trojfázový proces, ktorý opakovane vykonávame až dovtedy, pokiaľ nemáme k dispozícii bezchybnú verziu softvérového produktu, s ktorou sme úplne spokojní. Zmienený trojfázový proces je jadrom stratégie maximalizácie kvality kódovej bázy softvérového produktu. Pri stavbe veľmi rozsiahlych softvérových aplikácií je nutné implementovať kompletný systém riadenia kvality softvéru.

7. **Tvorba systému elektronickej dokumentácie, príprava inštruktážnych materiálov a marketingových dokumentov.** Každý seriózny softvérový produkt v súčasnej dobe sprevádza moderný systém elektronickej dokumentácie, ktorý vysvetľuje jeho funkčné rysy finálnym používateľom. Okrem súborov rôzne zameraných tematických blokov v sebe systém elektronickej dokumentácie integruje i bohatý a viacúrovňový register kľúčových slov, rovnako ako aj stroj na plnotextové prehľadávanie tém podľa zadanej postupnosti alfanumerických reťazcov. Systém elektronickej dokumentácie je starostlivo prepojený s počítačovou aplikáciou. Stupne prepojenia sú pritom rôzne, no vždy platí, že aplikácia vystavuje rozhranie, prostredníctvom ktorého je jej elektronická dokumentácia prístupná finálnym používateľom. Ako vývojári sa snažíme, aby bolo toto prepojenie logické, intuitívne a ergonomické.

   Za plánovanie, návrh a realizáciu systému elektronickej dokumentácie nie je priamo zodpovedný softvérový vývojár, ale špeciálna osoba známa ako technický spisovateľ. Ak je vytváraný program robustný, môže v organizačnej štruktúre softvérovej firmy existovať celá skupina technických spisovateľov s vlastným manažérom (vedúcim pracovníkom). Tak alebo onak, softvéroví vývojári a technickí spisovatelia sú v priamom kontakte a úzko medzi sebou spolupracujú. To je efektívne, pretože systém elektronickej dokumentácie sa tvorí súbežne s vývojom počítačovej aplikácie. Technickí spisovatelia majú na starosti aj prípravu rôznych inštruktážnych materiálov. K nim sa radia napríklad video sprievodcovia,

ktorí začínajúcich používateľov zoznamujú s užitočnými pracovnými metódami. Cieľom tejto iniciatívy je zvýšenie pracovnej produktivity finálnych používateľov pri používaní softvérového produktu. Na záver nesmieme zabudnúť ani na marketingové materiály, ktoré nesú jasné informačné posolstvo a propagujú silné stránky vytvorenej počítačovej aplikácie. Marketingové materiály sú zacielené na selektovaný segment finálnych používateľov, pričom sú komunikované tak, aby zasiahli maximálny možný počet dedikovaných finálnych používateľov. Na procese tvorby inštruktážnych a marketingových materiálov sa okrem technických spisovateľov podieľajú aj marketingoví kreatívci, grafickí špecialisti a umelci.

8. **Nasadenie softvérového produktu.** Keď je softvérová aplikácia hotová, čaká ju nasadenie u finálnych používateľov. Nasadením sa rozumie distribúcia programu a jeho inštalácia na stanicu každého klienta, ktorý si softvér zakúpi. Zatiaľ čo v uplynulých desaťročiach prevládala fyzická distribúcia softvérových produktov, v súčasnosti sa čoraz väčšej obľube teší ich elektronická distribúcia. Výnimkou nie je ani kombinovaná forma distribúcie programového vybavenia, ktorá spája silné stránky fyzickej a elektronickej formy distribúcie. Keďže chceme, aby finálni používatelia mohli s našou aplikáciou pracovať čo najrýchlejšie po jej obstaraní, dodávame ju so špeciálnym inštalačným programom. Inštalačný program nainštaluje aplikáciu na klientsku počítačovú stanicu. V priebehu inštalácie dochádza ku kopírovaniu programových súborov, realizuje sa zápis informácií do systémových registrov a uskutočňuje sa primárna konfigurácia programu pred jeho prvým spustením. Ak je to nutné, po prvom spustení softvérového produktu dochádza k začatiu aktivačného procesu. Voliteľne môžu finálni používatelia iniciovať aj registračný proces. Po úspešnej registrácii získajú vyššiu úroveň starostlivosti, bonusy či milé darčeky.

9. **Údržba softvérového produktu.** Napriek tomu, že v tomto momente je náš softvérový produkt prítomný na staniciach finálnych používateľov, naša starostlivosť oň sa nekončí. Práve naopak, sofistikovaná IT firma dbá na to, aby nielenže dodala kvalitný softvér, ale pokračovala v udržiavaní maximálnej možnej spokojnosti finálnych používateľov z jeho používania. K obľúbeným technikám,

ktoré vedú k tomuto cieľu, patria aktualizácia softvéru pomocou servisných balíčkov, inkrementálne inovácie softvéru (zdarma dostupné nové črty, ktoré rozšíria spektrum služieb pôvodného produktu) a riadenie zmien (promptné reagovanie na vylepšenia, návrhy či pripomienky finálnych používateľov k softvérovému produktu). V tomto smere platí známa obyčaj: „Čím lepšie sa budeme starať o používateľov našich programov, tým viac ich budeme mať".

2 Čo je super na tom, keď ste softvérovým vývojárom

Jednoduchá a milá odpoveď by mohla znieť: všetko. Ako softvéroví vývojári máme možnosť podieľať sa na tvorbe počítačových programov, ktoré dokážu uľahčiť a spríjemniť životy miliónom ľudí na tejto planéte. Naše programy pomáhajú ľuďom rýchlejšie a lepšie vykonávať ich pracovné úlohy (*rôzne typy aplikačného softvéru*), ochraňujú ľudské zdravie a bezpečnosť ľudských životov (*programy riadiace elektronické systémy dopravných strojov, programy poháňajúce zdravotnícke zariadenia či aplikácie riadiace procesy v elektrárňach a v iných strategických objektoch*), umožňujú ľuďom zostať v kontakte s najbližšími a zdieľať s nimi svoje zážitky (*programy vytvárajúce interaktívne a multimédiami podporované sociálne siete*), vykonávajú bezpečné transakcie s peňažnými prostriedkami (*programy elektronického bankovníctva*) a v neposlednom rade prinášajú ľuďom nové možnosti vzdelávania, zábavy a skvalitnenia životného štýlu (*didaktické programy, počítačové hry a elektronickí asistenti zdravého životného štýlu*). Ľudia na tomto svete majú tisíce potrieb a počítačové programy, ktoré vytvárame, slúžia hlavne na ich uspokojenie. Ako softvéroví vývojári si môžeme vybrať oblasť záujmu, preniknúť do nej, ovládnuť ju a stať sa v nej skutočnými profesionálmi.

Softvérovými vývojármi sa najčastejšie stávajú osoby s výborným analytickým myslením, nadpriemernou schopnosťou práce s abstraktnými symbolmi a záľubou v riešení zložitých problémov modernej informatiky. Vývojári sú ľudia, ktorí dokážu pracovať v stále sa meniacich podmienkach, radi sa zoznamujú s novými technológiami a robí im radosť študovať a používať inovatívne softvérové nástroje. Samozrejme, vývojári nežijú len

vytváraním softvérových produktov. Ak odhliadneme od profesionálnych atribútov typických softvérových vývojárov, môžeme konštatovať, že táto sorta ľudí vyznáva dynamický životný štýl, v ktorom majú zastúpenie rôzne športové aktivity (to je výborné, pretože, ako vieme, šport udržiava čistú myseľ, a tá má pre vývojárov cenu zlata).

Softvéroví vývojári žijú v turbulentnom prostredí, ktorého medze ohraničujú rôzne projekty, iniciatívy, stratégie, výzvy, technológie a nástroje, s ktorými sa vývojári dennodenne stretávajú. Práca softvérových vývojárov je preto dynamická, rozmanitá a prostá akéhokoľvek stereotypu. Na tomto svete neexistuje veľa povolaní, u ktorých je preukázateľne dokázaný ich jednoznačne pozitívny vplyv na rozvoj ľudských intelektuálnych schopností. Vývoj softvéru k nim však určite patrí – a to je fajn, pretože vycibrená schopnosť riešenia abstraktných programových úloh umožňuje vývojárom rýchlejšie a efektívnejšie riešiť aj bežné problémové situácie v „normálnom" živote.

Práca softvérového vývojára je takmer vždy tímovou prácou s inými softvérovými vývojármi. Vývojári sú šťastní, keď môžu zdieľať svoje nadšenie a pracovný entuziazmus s ďalšími, podobne zmýšľajúcimi osobami. V praxi našťastie neplatí téza o tom, že typický tvorca počítačového programového vybavenia je zvláštny introvert, ktorý pracuje v ústraní v akejsi sporo osvetlenej miestnosti. Ľudia vymýšľajúci softvér sú spoločenskí, radi o svojich úlohách a navrhovaných riešeniach diskutujú, pričom sú naklonení konštruktívnej kritike. Mnohí softvéroví vývojári sú aktívnymi členmi dedikovaných vývojárskych komunít a snažia sa pomáhať „služobne mladším" kolegom v rozvoji ich technických, analytických a odborných zručností. Ba čo viac, nemálo softvérových vývojárov sa pohybuje aj v rýdzo akademických a vedeckých kruhoch, čo dokazuje, že vo svojom povolaní objavili vyšší princíp a chápu ho ako svoje poslanie. Veď čo môže byť zmysluplnejšie, ako odovzdávanie vedomostí, schopností a zručností novo nastupujúcim generáciám inteligentných mladých ľudí.

Okrem spoločenskej prestíže je softvérovým vývojárom vlastná aj značná finančná exkluzivita. V pravidelne uskutočňovaných platových monitoroch sa pracovné pozície softvérových vývojárov umiestňujú na čelných priečkach platového rebríčka. Hoci vieme, že priemerné údaje platov v IT branži nie sú vždy presným odrazom skutočnosti, je jasné,

že skúsení softvéroví vývojári sú a ešte dlho budú na pracovnom trhu patriť k tým najžiadanejším.

3 Rád by som sa stal softvérovým vývojárom, čo mám pre to urobiť?

Ak sme vás presvedčili o tom, že profesionálna kariéra softvérového vývojára bude pre vás to pravé, sme na dobrej ceste. Ste študentom strednej školy, vysokoškolákom či riadnym zamestnancom v informačných technológiách, prípadne v inom odvetví? V poriadku, pre všetky spomenuté segmenty osôb máme pripravené odporúčania hodné postupy, ktoré vás spoľahlivo a úspešne dovedú k vytýčenému cieľu – vysnívanej práci softvérového vývojára.

3.1 Scenár 1: Som študentom strednej školy a chcel by som sa stať softvérovým vývojárom

Naše odporúčanie, ako na to: Počas stredoškolského štúdia prejavte záujem o rozšírené vyučovanie matematiky a informatiky. Popri štandardných (normovaných) výučbových jednotkách týchto predmetov sa prihláste aj na doplňujúce formy výučby, akými sú semináre, praktické cvičenia či laboratórne experimenty. V matematike sa sústreďte na zvládnutie klasickej (v zmysle dvojhodnotovej) matematickej logiky (ide najmä o výrokovú a predikátovú logiku), teóriu množín, relácie a prácu s funkciami. Vymedzte si porciu svojho voľného času na riešenie slovných matematických úloh, ktoré pomôžu formovať schopnosti vášho analytického myslenia a vylepšia techniku vašej problémovej dekompozície. V informatike ovládnite základy ako aj pokročilé aspekty operačných systémov a bežného typového aplikačného softvéru (ide predovšetkým o kancelársky softvér a softvér automatizujúci úlohy, s ktorými sa finálni používatelia frekventovane stretávajú). Naštudujte si základy algoritmizácie, aby ste sa

dozvedeli, čo sú to algoritmy, aké majú vlastnosti, na čo slúžia a ako prebieha proces ich zostavovania. Oboznámte sa s elementárnymi algoritmickými riadiacimi štruktúrami, ku ktorým patrí sekvencia (postupné spracovanie krokov algoritmu), selekcia (vetvenie toku algoritmu v závislosti od vyhodnotenia podmienky) a iterácia (opakované vykonanie množiny krokov algoritmu). Naučte sa programovať v jednom programovacom jazyku, najlepšie rýdzo štruktúrovanom. Takýmto programovacím jazykom je napríklad jazyk C, ktorého zvládnutie vám vrelo odporúčame. Vzhľadom na to, že väčšina informaticky orientovaných vysokých škôl univerzitného typu má vo svojich študijných programoch predmety ako „Štruktúrované programovanie v jazyku C" alebo „Procedurálne programovanie v jazyku C", je ovládnutie jazyka C už na strednej škole brané ako výhodná investícia do budúcnosti.

Neváhajte a zúčastňujte sa matematických a informatických olympiád či rôznych stredoškolských programátorských súťaží. Tieto podujatia sú skvelé, pretože v nich môžete testovať aktuálnu úroveň vašich teoretických vedomostí a praktických zručností. Navyše, spoznáte mnohých kamarátov a kamarátky, pre ktorých je matematika a informatika rovnako inšpirujúca a zábavná ako pre vás.

Keď sa budete rozhodovať o skladbe vašich maturitných predmetov (čo sa obvykle deje v prvej polovici tretieho ročníka štúdia), určite medzi ne zaraďte matematiku a informatiku (ak plánujete ďalej študovať na Matematicko-fyzikálnej fakulte Univerzity Komenského, pridajte k nim aj fyziku). Po úspešnom zložení maturitnej skúšky pokračujte v nadväzujúcom vysokoškolskom štúdiu. Výber kvalitnej vysokej školy so zameraním na výučbu informatiky je veľmi dôležitý krok, a preto mu venujte dostatočné množstvo času.

3.2 Scenár 2: Som študentom vysokej školy a chcel by som sa stať softvérovým vývojárom

Naše odporúčanie, ako na to: V ideálnom prípade ste práve poslucháčom prvostupňového (bakalárskeho) študijného odboru „Informatika" (prípadne príbuzného študijného odboru) na vysokej škole univerzitného typu. Študijný

odbor, ktorý študujete, exaktne definuje rozsah študijného programu, ktorý tvoria vysokoškolské predmety rôznych druhov (v tomto smere sa spravidla uplatňuje „triádová" klasifikácia predmetov na povinné predmety, povinne voliteľné predmety a voliteľné predmety). Ako hlavný garant determinuje vybraná fakulta vysokej školy optimálnu skladbu predmetov daného študijného programu v požadovanej časovej výmere (na prvom stupni štúdia sú výučbové jednotky štandardne diferencované medzi prednášky a cvičenia). Študenti sa môžu oboznámiť s curriculom každého jedného predmetu, čím získajú vedomosti o tom, akým smerom sa daný predmet uberá a čo svojim poslucháčom poskytuje (zjednodušene povedané, študenti sa dovtípia, „čo ich čaká a neminie"). Fakulta, ktorá garantuje realizáciu informatických študijných programov, vždy kladie dôraz na optimálne študijné zaťaženie svojich študentov. Znamená to, že predmety, ich náročnosť a vzájomné prepojenie sú konfigurované tak, aby generovali pre študentov maximálny možný pozitívny synergický efekt.

Aj keď sme si vedomí skutočnosti, že študijné programy rôznych informatických vysokých škôl sa môžu v určitých rysoch odlišovať, základné študijné zameranie je jednotné. Na nasledujúcich riadkoch si dovolíme uviesť významné sústavy informatických predmetov, ktoré odporúčame s vysokým nasadením zvládnuť všetkým budúcim softvérovým vývojárom.

Sústava predmetov 1: Teoretická informatika, teória algoritmov a programovanie

Charakteristika: Teoretická informatika je značne široká disciplína, ktorá siaha od teórie informácie, cez algoritmizáciu, až po problematiku dátových štruktúr a programovacích techník. Podľa sofistikovanosti zamerania a stupňa náročnosti vysokej školy môže byť teoretická informatika viac či menej prepletená s vyššou matematikou. Ako nezriedka vravia študenti po absolvovaní vybraných partií teoretickej informatiky: „Až teraz naozaj oceňujeme úlohu matematiky pri budovaní modernej informatiky". Keďže sa chcete stať softvérovým vývojárom, je vynikajúce zvládnutie predmetov teoretickej informatiky nutnou podmienkou vášho ďalšieho akademického napredovania. Prosím, pripravte sa na to, že sa budete potýkať aj s ťažkými chvíľkami, a to najmä pri štúdiu netriviálnych sekcií predkladaného výseku

teoretickej informatiky. Aby ste dokázali študijný nápor zdarne ustáť, odporúčame vám viesť sa radami, ktoré uvádzame v kapitole *4 Ako efektívne študovať algoritmizáciu a programovanie*.

Sústava predmetov 2: Databázové systémy a technológie

Charakteristika: Každá stredne veľká počítačová aplikácia, respektíve každý informačný systém manipuluje s kvantami dát, ktoré sú ukladané v databázach. Je preto vhodné poznať spôsoby archivovania a získavania dát z rôznych dátových zdrojov za prispenia systémov na riadenie bázy dát. Hoci teoretická informatika zostavila viacero prístupov k správe dát a definovala hneď niekoľko dátových modelov a typov databáz, pre budúcich softvérových vývojárov je signifikantné, aby sa zoznámili najmä s relačnými a objektovými databázami. V tomto kontexte je dôležité pochopiť najmä primárne databázové objekty (entity) a vzťahy medzi nimi. Stranou nesmie zostať ani problematika dátového modelovania – študenti by určite mali získať poznatky o tvorbe logických, konceptuálnych a fyzických databázových modelov. Rovnako odporúčame naučiť sa deklaratívny jazyk SQL, ktorý umožňuje formulovať dopyty (respektíve dopytovacie výrazy) v záujme získania vybranej množiny dát z požadovanej databázy (výber je obvykle realizovaný za pomoci vopred špecifikovaných predikátov). Rozhľad poslucháčov rozšíria aj praktické skúsenosti s lokálnymi či distribuovanými databázovými systémami.

Sústava predmetov 3: Operačné systémy

Charakteristika: Operačný systém je viacvrstvový počítačový program, ktorý zavádza medzi finálneho používateľa a technické zariadenie počítača účinnú vrstvu abstrakcie. Vďaka operačnému systému sa môžu používatelia sústrediť na riešenie svojich úloh a nemusia sa starať o uskutočňovanie nízkoúrovňových softvérových služieb. Študenti informatiky musia poznať typológiu a architektonickú štruktúry moderných operačných systémov. Medzi ďalšie významné vedomosti patrí: správa procesov, správa pamäte a správa periférnych zariadení. S rozmachom počítačov, ktoré sú osadené viacjadrovými procesormi, je nutné vedieť, ako

fungujú paralelné a pseudoparalelné modely spracovania počítačových aplikácií bežiacich v operačnom systéme.

Sústava predmetov 4: Počítačové siete a internetové technológie

Charakteristika: Ak máte ambíciu stať sa webovým vývojárom, potom je starostlivé zvládnutie problematiky počítačových sietí a spriaznených internetových technológií pre vás emergentne dôležité. Odhliadnuc od lokálnych a globálnych sietí rôznej topológie, významné je pochopenie distribuovaného spracovania dát a distribuovaného spracovania procesov. Ako si môžeme všimnúť, v týchto aspektoch vidíme interdisciplinárny prienik počítačových sietí a distribuovaného programovania, respektíve počítačových sietí a distribuovaných databázových systémov. V prvostupňovom vysokoškolskom štúdiu však odporúčame zamerať sa len na základné témy počítačových sietí a webových technológií. Zväčšenie rozsahu preberanej látky ako aj jej prehlbovanie môžeme odložiť až na druhostupňové (inžinierske či magisterské) vysokoškolské štúdium, ktoré má špecializačný charakter.

Sústava predmetov 5: Umelá inteligencia a expertné systémy

Charakteristika: Umelá inteligencia sa zaoberá vytváraním rozumných a efektívne konajúcich agentov, ktorých správanie pri riešení problémov simuluje správanie skutočných ľudí. Umelá inteligencia, neurónové siete, genetické a evolučné algoritmy sú jednou z najdynamickejšie sa rozvíjajúcich oblastí modernej počítačovej vedy. Expertné systémy zasa predstavujú počítačové programy, ktoré dokážu automatizovať riešenie štruktúrovaných i neštruktúrovaných problémov, čím sú schopné zastúpiť rolu človeka v rozhodovacích procesoch. Expertné systémy sú nasadzované napríklad v medicíne pri diagnostike ochorení, v elektrotechnickom priemysle pri navrhovaní a konfigurovaní výrobných komponentov a v neposlednom rade sa expertné systémy využívajú aj v biznis procesoch, pri dolovaní dát a v procese inferencie nových znalostí z existujúcich znalostných báz.

3.3 Scenár 3: Som zamestnancom v IT firme a chcel by som sa stať softvérovým vývojárom

Naše odporúčanie, ako na to: Aby sme vám vedeli čo možno najlepšie pomôcť, musíme sa vás spýtať nasledujúce otázky:

1. **Na akej pozícii aktuálne pracujete?**
   Vaše súčasné pracovné zaradenie je významné, pretože podľa neho vieme určiť, koľko prostriedkov bude nutné investovať do pracovnej zmeny na pozíciu softvérového vývojára. Aby sme si našu analýzu uľahčili, rozdelíme všetky pracovné pozície IT firmy do troch skupín: technické pozície, ekonomické pozície a iné pozície. Najmenej úsilia, energie a prostriedkov vynaložíte vtedy, keď už pracujete na technickej, hoci inej ako vývojárskej pozícii. Napríklad, môžete byť súčasťou tímu, ktorý dohliada na dodržiavanie kvality softvéru. Alebo sa venujete administrácii databázových serverov. Prípadne vykonávate pomocné programátorské práce – trebárs píšete automatizačné skripty či makrá. Ak je to tak, najlepší postup bude pre vás takýto:

   - Zoznámte sa so základmi objektového modelovania v jazyku UML.
   - Na teoretickej a praktickej úrovni sa oboznámte s objektovým a paralelným objektovým programovaním.
   - Naučte sa programovať v robustnom hybridnom a viacparadigmovom programovacom jazyku. Svoje šance zvýšite, keď postupne zvládnete viacero programovacích jazykov a dosiahnete v nich vyšší stupeň proficiencie. Z pohľadu najrýchlejšieho zvládania programovacích jazykov odporúčame takúto triádu: jazyk Visual Basic[1] → jazyk C# → jazyk C++.
   - Oboznámte vášho nadriadeného s tým, že by ste radi participovali na činnostiach, ktoré priamo súvisia s vývojom počítačového softvéru.
   - Staňte sa platným členom tímu softvérových vývojárov.

[1] V tomto kontexte máme samozrejme na mysli „dotnetový" Visual Basic, najlepšie v aktuálnej verzii, ktorou bola v čase písania tejto príručky verzia Visual Basic 2010.

Ak pracujete na niektorej z ekonomických či iných pracovných pozícií, odporúčame vám absolvovať sériu rekvalifikačných a tréningových informatických kurzov, v ktorých teoreticky a prakticky zvládnete všetky technické oblasti, ktoré budete ako budúci softvéroví vývojári k svojej práci potrebovať.

2. **Aké je vaše najvyššie dosiahnuté vzdelanie a v akom študijnom odbore ste ho získali?**

   Pokiaľ disponujete formálnym informatickým vzdelaním 2. stupňa vysokoškolského štúdia (ste teda nositeľom magisterského či inžinierskeho vysokoškolského titulu), máte všetky predpoklady na to, aby ste rýchlo migrovali na pozíciu softvérového vývojára. Ako absolvent informatického študijného programu dobre poznáte teoretické aspekty softvérového inžinierstva. Ak cítite, že vaše praktické zručnosti (napríklad v programovaní) nie sú na takej úrovni, na akej by mali byť, stanovte si motivujúci harmonogram, v ktorom si naplánujete podporné programátorské aktivity. K nim môže patriť riešenie vybraných úloh samoštúdiom alebo individuálnym elektronickým vzdelávaním (e-learningom), absolvovanie praktických počítačových seminárov či realizovanie technických stretnutí s inými softvérovými vývojármi vo vašej firme.

   Ak ste hrdým majiteľom doktorského titulu PhD. v informatike (teda disponujete vedeckou hodnosťou, ktorá dokazuje absolvovanie najvyššieho – tretieho – stupňa vysokoškolského vzdelávania), vaše možnosti sú samozrejme ešte väčšie. Ľudia s takýmto vzdelaním sa spravidla stávajú kmeňovými (a vysoko cenenými) členmi vývojárskych tímov. Prirodzene, s PhD. titulom sa môžete veľmi dobre uplatniť aj v akademickej a vedeckej sfére. Domnievame sa, že optimálne je, keď všetky tieto sféry prepojíte dovedna, čím sa vám podarí „nakresliť" trojuholník „veda – prax – vzdelávanie".

3. **Viete programovať?**

   Keďže softvérový vývojár sa živí programovaním, akosi implicitne očakávame kladnú odpoveď. Ak ste sa však s programovaním ešte nikdy nestretli, budete musieť zvládnuť pokročilé aspekty objektového programovania vo vybranom

programovacom jazyku. Možno, že ste programovali na vysokej škole, no od jej ukončenia už predsa len uplynulo viacero rokov. Žiaden strach, programovanie sa nezmenilo, iba zaznamenalo citeľný progres smerom vpred (toto konštatovanie ostatne neplatí len pre programovanie, ale snáď pre všetky oblasti ľudských činností). Ak ste už niekedy programovali, zvládnutie moderných programovacích jazykov a s nimi spojených integrovaných vývojových prostredí nebude vôbec ťažké.

4. **V ktorých programovacích jazykoch viete plynulo písať kód?**
   Na tomto mieste by sme mohli parafrázovať známe príslovie a vyhlásiť, že „koľko programovacích jazykov ovládate, toľkokrát ste softvérovým vývojárom". V komerčnej praxi majú najväčšie zastúpenie programovacie jazyky garantujúce vysokú pracovnú produktivitu softvérových vývojárov. Ako už bolo spomenuté, na platforme spoločnosti Microsoft ide o jazyky C# a Visual Basic. Na svete však jestvujú desiatky rôznych programovacích jazykov a rovnako desiatky oblastí vývoja počítačového softvéru. Pritom platí zásada, že sa učíme hlavne ten programovací jazyk, ktorý je platný pre našu pracovnú doménu. Prirodzene, čím viac programovacích jazykov ovládate, tým sa zvyšuje hodnota vášho ľudského kapitálu.

4 Ako efektívne študovať algoritmizáciu a programovanie

Z našich akademických skúseností vyplýva, že algoritmické a programátorské predmety patria z pohľadu študentov k najťažším informatickým predmetom. Áno, súhlasíme s tým, že požiadavky kladené týmito predmetmi na abstraktné myslenie študentov sú istotne vyššie než u iných, aj keď rovnako informatických predmetov (ako sú napríklad databázové technológie a dátové sklady, počítačové siete či manažérska informatika). Aby sme pomohli študentom (a teda budúcim softvérovým vývojárom) zvládnuť zaťaženie, ktoré generuje ovládnutie algoritmizácie a programovania,

dovoľujeme si v tejto kapitole predstaviť metodiku *Ako efektívne študovať algoritmizáciu a programovanie*:

1. **Naučiť sa dobre programovať je beh na dlhú trať.** Je veľmi dôležité, aby si študenti hneď na začiatku uvedomili, že jedinou cestou k úspechu je pravidelná študijná príprava. Alebo ako radi hovoríme, programovanie je najlepšie vtedy, keď sa podáva častejšie a v menších porciách. Tým sa snažíme poslucháčom doručiť posolstvo, že v záujme dosiahnutia stanoveného cieľa je nutné s programovaním udržiavať pravidelný kontakt. Programovanie rozhodne nepatrí k predmetom, ktoré sa dajú učiť „nárazovo", takže bohužiaľ úplne mylná je predstava mnohých študentov, ktorí sa nazdávajú, že na zvládnutie programovania im bude stačiť približne týždeň pred skúškou.

2. **Programovanie je teoreticko-praktickou disciplínou.** Keď diskutujeme o programovaní, tak musí byť jasné, že nejde nijakým spôsobom oddeliť teóriu od praxe a prax od teórie. Zdá sa nám samozrejmé, že programovať znamená tvoriť počítačové programy. No túto činnosť dokážu študenti vykonávať len vtedy, keď dokonale chápu všetky teoretické princípy a techniky programovania. Spravidla platí, že určitý fragment študentov má tendenciu uprednostniť memorovanie teórie vrátane zdrojových kódov programov pred praktickým kontaktom s programovaním. Dovoľujeme si podotknúť, že toto je výsostne neefektívny prístup, ktorý skôr alebo neskôr vyústi do mnohých kolíznych stavov. Jedným z najkritickejších je učenie sa zdrojových kódov programov naspamäť. Problémy sa začnú s vysokou frekvenciou objavovať vo chvíli, keď je nutné, aby študenti napísali iný program, či vtedy, keď je úlohou študentov upraviť pôvodný program podľa vopred stanoveného zadania.

3. **Programovať znamená predovšetkým logicky myslieť.** Logika hrá v programovaní nesmierne dôležitú úlohu. Vždy sa snažíme prízvukovať, že programovanie je v skutočnosti abstraktná hra s algoritmami, ktoré sú konštruované podľa všeobecne platných logických pravidiel. Nie je pochýb o tom, že programovanie pomáha zvyšovať úroveň abstraktného myslenia študentov, čo

je fakt, ktorý má jednoznačne pozitívne implikácie (a to nielen v programovaní, ale tiež v iných predmetoch a koniec koncov aj v každodennom živote).

4. **Programovanie bude vždy o technickej analýze zdrojových kódov vytvorených programov.** Ak už sme uviedli postulát „Programovať znamená vytvárať programy", tak musíme doplniť aj ďalší „Programovať znamená do najmenších podrobností vysvetliť, ako vytvorené programy pracujú". Našou snahou je viesť študentov k tomu, aby vedeli uskutočniť rigoróznu technickú analýzu akéhokoľvek programu (alebo jeho časti). Stručne povedané, študent musí vedieť nielen to, ako analyzovaný program funguje, ale tiež to, ako sa správa každý jeden príkaz tohto programu. Pri technickej analýze nesmie zostať žiadny príkaz a dokonca ani žiadny výraz v danom príkaze neznámy či neobjasnený. Veď, napokon, programovanie je deterministická disciplína, nejde o žiadnu mágiu. Inými slovami, nič v zdrojovom kóde programu sa nesmie tváriť ako „čierna skrinka", do ktorej nedokážu študenti nahliadnuť. Ako radi konštatujeme, zdrojový kód je priezračne čistý: študenti majú všetko pred sebou, len to musia byť schopní pochopiť a správne interpretovať.

5. **Programovať sa možno naučiť jedine sústavným programovaním.** Hoci to možno bude znieť samozrejme, svoje umenie programovať dovedú študenti k dokonalosti tak, že budú vymýšľať stále nové a nové algoritmy, a tie potom implementovať vo svojich programoch. Aj keď vedľa prednášok sú neoddeliteľnou súčasťou vysokoškolského štúdia tiež praktické semináre a cvičenia, v záujme lepšieho zžitia sa s programovaním je určite potrebná aj dodatočná iniciatíva zo strany študentov. Optimálne je, keď študenti venujú programovaniu istú časť svojho voľného času, pričom riešia úlohy zadané prednášajúcim alebo cvičiacim učiteľom (alebo si iniciatívne vymýšľajú svoje vlastné úlohy).

6. **Štúdium programovania musí byť zábavné.** Nazdávame sa, že pedagogickí pracovníci by sa mali zo všetkých síl snažiť o to, aby na programovanie študenti pozerali ako na zábavnú hru s algoritmami a nie ako na nutné zlo, ktoré ich na

vysokej škole postretlo. Správnym cieľom je vyvolať u študentov záujem o programovanie a viesť ich k samostatnej tvorivej programátorskej práci. Sme úplne presvedčení o tom, že paradigma „Programovanie je pre nás zábavné" umožní študentom nájsť k tomuto predmetu priateľský vzťah, a následne ho, pochopiteľne, aj úspešne absolvovať.

7. **Ak je to nutné, nebojte sa požiadať o pomoc.** Nové veci sa vždy zvládajú lepšie, keď sa môžete obrátiť na osobu, ktorá vám pomôže a ukáže vám, ako postupovať ďalej. Ak študenti cítia, že takáto asistencia by im prišla vhod, majú na výber z niekoľkých variantov ďalšieho postupu. Predovšetkým, každý prednášajúci vedie počas semestra sériu svojich konzultačných hodín. Tie sú vyhradené na exkluzívnu pomoc a podporu študentov. Študenti sa teda môžu so svojím pedagógom dohodnúť na pravidelných konzultáciách, ktoré budú potom svedomito navštevovať. Podobne ako prednášajúci, tiež vedúci cvičení vypisujú svoje konzultačné hodiny, čiže aj ich pedagogické kapacity sú študentom v príhodnom čase k dispozícii. Všetkým študentom preto odporúčame, aby túto formu intelektuálnej pomoci využili. Bohužiaľ, v praxi sa stáva, že niektorí študenti majú (aj keď z nášho pohľadu značne nejasné) dôvody, prečo nechcú za svojím prednášajúcim či cvičiacim prísť o poprosiť ho o pomoc. Nevieme, čo sa za tým skrýva, či strach alebo neopodstatnené obavy zo zlyhania. Pokiaľ sa nepodarí študentom tieto dôvody prelomiť, tak im odporúčame, aby požiadali o asistenciu kamaráta, spolužiaka, staršieho súrodenca, prípadne inú im známu osobu, ktorá programovaniu rozumie.

5 Ako prebieha pracovný pohovor na pozíciu softvérového vývojára

Úspešní absolventi informatických vysokých škôl sa môžu zamestnať v rôznych IT firmách alebo softvérových spoločnostiach. Vyberať môžu z nasledujúcich typov firiem:

- zahraničné IT firmy,
- slovenské pobočky (filiálky) či dcérske spoločnosti zahraničných IT firiem,
- slovenské IT firmy.

Prvá alternatíva vyžaduje obvykle permanentné pôsobenie v zahraničí, zostávajúce dva varianty takúto požiadavku pred ašpirantov nekladú.

Radi by sme predoslali, že v tejto kapitole vyložíme priebeh pracovného pohovoru na pozíciu softvérového vývojára. To však samozrejme neznamená, že softvérový vývojár je jedinou profesiou v IT priemysle. Okrem tvorcov softvéru sú na trhu žiadaní tiež ďalší pracovníci, ku ktorým patria IT analytici, softvéroví architekti, databázoví profesionáli, počítačoví konzultanti či hardvéroví špecialisti. Rovnako by sme radi uviedli, že priebeh pracovného povohoru, ktorý budeme charakterizovať, v sebe integruje všeobecné rysy pracovného náboru na pozíciu softvérového vývojára. Keďže existujú rôzne IT firmy s rôznymi požiadavkami, nie je vylúčené, že do pracovného pohovoru bude zaradená i ďalšia činnosť, ktorú v našom modeli nespomenieme. Toto platí aj naopak – vo vašom pracovnom pohovore môžu absentovať niektoré aspekty, o ktorých budeme hovoriť. To sa pochopiteľne môže stať, pretože neexistuje žiadna pevne daná forma pracovného pohovoru na pozíciu softvérový vývojár. Firmy majú v tomto smere voľné ruky, a preto môžu konfigurovať fázy náborového procesu tak, ako sami uznajú za vhodné. My sme čerpali z našich skúseností, pričom sme uplatnili stratégiu detekcie a následnej implementácie najčastejšie sa vyskytujúcich štádií pracovného pohovoru naprieč analyzovanými IT firmami.

Pracovný pohovor môže byť jednokolový alebo viackolový. Jednokolové pracovné pohovory sú typické pre pozície začínajúcich softvérových vývojárov (ide o takzvané junior pozície). Junior pozície sú zamerané predovšetkým na čerstvých absolventov vysokých škôl so žiadnou alebo minimálnou praxou. Viackolové pracovné pohovory sa obvykle uskutočňujú pri nábore skúsených a ostrieľaných, takzvaných senior softvérových vývojárov.

Pracovný pohovor sa skladá z nasledujúcich etáp:

1. **Úvodný pohovor s pracovníkom personálneho oddelenia.** Na úvodnom pohovore budete privítaní personalistom, ktorý vás stručne oboznámi s IT spoločnosťou, jej trhovým pôsobením, produktovým portfóliom a poskytovanými službami. Personalista vás rovnako bližšie zoznámi s charakterom pracovnej pozície a overí materiály dokumentujúce vaše vzdelanie a prax v odbore (ak je vyžadovaná). Dĺžka úvodného pohovoru zvyčajne nepresiahne 30 minút.

   

   Naše odporúčania, ako všetko zvládnuť v pohode: Na pracovný pohovor príďte elegantne oblečení, v dobrej nálade a s úsmevom. Uvedomte si, že toto je váš deň, pretože práve dnes máte svetu ukázať, akí ste dobrí v algoritmizácii a programovaní, teda v tom, čo máte najradšej. Vždy je vhodné, keď urobíte dobrý prvý dojem. Zistite si vopred informácie o IT firme, do ktorej sa hlásite. V úvodnom pohovore s personálnym pracovníkom potom neváhajte tieto znalosti použiť. Ukážete, že ste prišli pripravení a o ponúkanú prácu máte skutočný záujem.

2. **Technický pohovor s vedúcim vývojárskej divízie alebo s manažérom softvérového tímu či delegovaným skúseným softvérovým vývojárom IT firmy.** Ide o najťažšiu časť pracovného pohovoru, ktorej cieľom je verifikácia vašich algoritmických a programátorských vedomostí a zručností. Technický pohovor môže byť vedený ústne, písomne alebo kombinovanou formou. V našich podmienkach je celkom obľúbená kombinovaná forma technického pohovoru. Začína sa písomným preskúšaním teoretických vedomostí uchádzača. Predložený

test môže obsahovať uzatvorené aj otvorené otázky. Test obsahuje technické úlohy rozličnej náročnosti, pričom náročnosť úloh spravidla pozvoľne stúpa. Nezriedka sa stáva, že v teste sa nachádzajú veľmi náročné a na prvý pohľad snáď až neriešiteľné úlohy. Potešujúcou správou je, že pri tejto sorte úloh sa ani neočakáva, že by ste ich v poskytnutom čase kompletne vyriešili (pri takýchto úlohách ide skôr o nájdenie možného riešenia, no už nie o jeho kompletnú implementáciu). Na riešenie písomného testu je vždy presne stanovený čas a ak nie je určené inak, platí, že uchádzači nemôžu využívať žiadne pomôcky (ako sú napríklad počítače, učebnice alebo poznámkové bloky). Po dokončení písomnej časti technického pohovoru sa prechádza na ústnu časť.

Tá je vedená formou technickej diskusie, čo znamená, že môžete očakávať nasledujúce veci:

- fragmenty zdrojových kódov programov (alebo ich vybraných častí) s požiadavkou ich komplexnej technickej analýzy,
- úlohy orientované na výber najefektívnejšieho algoritmu zo všetkých algoritmov, ktoré pripadajú do úvahy (teda algoritmov schopných vyriešiť predložené úlohy),
- úlohy, v ktorých bude treba navrhnúť algoritmus riešeného problému a charakterizovať ho z hľadiska časovej a pamäťovej zložitosti,
- dopyty na konkrétne predvedenie programátorských konštrukcií či techník v dohovorenom hybridnom objektovom programovacom jazyku (zvyčajne ide o C++, C# alebo Visual Basic),
- praktické otázky súvisiace s použitím vybraných softvérových nástrojov, platforiem a technológií.

Technický pohovor štandardne trvá 1 – 2 hodiny, vo výnimočných prípadoch aj viac, no jeho celková dĺžka nikdy nepresiahne hranicu 3 hodín. Cieľom osoby, ktorá technický pohovor vedie, je zistiť, kde sú hranice vašich schopností. (Ako títo ľudia často s obľubou vravia: „Chceme zistiť, kde sa nachádza *bod zlomu* testovaného uchádzača.“) Predmetom sledovania je aj skutočnosť, ako dobre viete pracovať v simulovaných záťažových podmienkach. Technická diskusia je založená na mechanizme predkladania nových a nových problémových úloh, z ktorých všetky nasledujúce úlohy sú ťažšie ako tie pred nimi. Ak sú viac ako 2 za sebou idúce úlohy v tomto reťazci nad sily adepta, dochádza k ukončeniu technickej diskusie. To je, mimochodom, rozumné, pretože ak uchádzač nezvládol dané úlohy, je vylúčená eventualita, že by vyriešil nastávajúce, oveľa ťažšie úlohy. Na druhej strane, kandidát na pracovnú pozíciu je tým kvalitnejší, čím dlhšie dokáže tomuto náporu odolávať.

Je možné, že celá technická diskusia alebo jej časť bude vedená v cudzom jazyku (takmer vždy sa jedná o angličtinu). Motívom tohto počínania je zistiť úroveň vašej hovorovej odbornej (technickej) angličtiny pri riešení reálnych problémov.

Naše odporúčania, ako všetko zvládnuť v pohode: Hoci ide o testovanie vašich odborných znalostí, zachovajte pokoj a rozvahu. Ak vám niečo nie je pri písomnom teste jasné, nebojte sa poprosiť o vysvetlenie či usmernenie. Nech vás nevyvedie z koľají ani to, ak nebudete vedieť jednu či dve úlohy vôbec vyriešiť. Za týchto okolností sa pokúste aspoň naznačiť, ako by ste postupovali. Pracovný pohovor nie je skúška na vysokej škole – napriek tomu, že vaše odborné know-how je podrobované záťažovému testu, vy a skúšajúca osoba vystupujete ako partneri.

Pri ústnej technickej diskusii myslite nahlas, racionálne komentujte svoje kroky a podľa potreby interpretujte ich praktické prínosy. Ak vidíte viacero možností ďalšieho postupu, spomeňte ich, no pre praktické nasadenie vyberte práve jednu možnosť, ktorá je v danom kontexte najefektívnejšia. Pritom vysvetlite vašu voľbu: ukážte, prečo je vami zvolený variant vhodný, zatiaľ čo ostatné varianty nie.

V istých okamihoch prevezmite iniciatívu a obráťte garde – nadviažte so skúšajúcou osobou dialóg a kladťe jej otázky, ktoré sú technicky náročné, no relevantné vzhľadom na povahu riešenej úlohy. Pokiaľ sa vyskytne otázka, na ktorú neviete odpovedať, priznajte to a nevymýšľajte si ad-hoc odpovede s neoverenými riešeniami. Nikto nevie všetko, veď sa učíme celý život.

Skôr, ako pôjdete na pracovný pohovor, majte na pamäti skutočnosť, že jeho časť môže byť vedená v angličtine (túto informáciu by ste mali v každom prípade dostať s dostatočným časovým predstihom). Počítajte s tým a v domácom prostredí simulujte diskusiu, v ktorej využijete odbornú programátorskú terminológiu a naučíte sa známe obraty (ak sú použité správne a v príhodnom čase, citeľne zvýšia vašu kredibilitu).

3. **Ekonomický pohovor s vedúcim vývojárskej divízie a ekonomickým manažérom.** Do tejto fázy sa uchádzač dostane len vtedy, keď úspešne absolvuje predchádzajúci technický pohovor. Ak ste sa dostali až sem, blahoželáme vám, pretože ste už len na krôčik k celkovému úspechu. S ekonomickým manažérom budete konzultovať druh pracovného pomeru, výšku mzdy, pracovné podmienky, prípadne iné aspekty pracovnej pozície, na ktorú ste sa prihlásili. Softvéroví vývojári majú často flexibilnú pracovnú dobu s pevne stanoveným časovým rozpätím, v rámci ktorého musia byť prítomní na svojom pracovisku. Otázka výšky platového ohodnotenia je vždy citlivá, a preto je k nej nutné pristupovať uvážlivo, avšak so zdravým sebavedomím. Ako sme už v tejto príručke spomenuli, platy softvérových vývojárov sú, v závislosti od ich znalostí a skúseností, mierne alebo aj vysoko nadštandardné. Rozptyl v mzdách softvérových vývojárov však môže byť rovnako signifikantný, dokonca nie je vylúčené, že sa bude pohybovať aj v tisícoch eur.

Naše odporúčania, ako všetko zvládnuť v pohode: Azda najkomplikovanejšou je otázka zladenia vašich mzdových očakávaní s disponibilnými možnosťami IT spoločnosti. V niektorých firmách vám sami povedia úrovne, na ktorých sa môže vaše platové ohodnotenie pohybovať.

Inde sa vás zasa rovno spýtajú na vašu predstavu, pričom očakávajú presné číslo. Pokiaľ hovoríme o výške mzdy, máme vždy na mysli hrubú (teda brutto) mzdu, ktorá nie je očistená o daň z príjmu a potrebné odvody. Vami požadovanú mzdu si premyslite vopred – jej výšku nikdy nestanovujte až v priebehu pohovoru.

5.1 Ukážkový písomný test algoritmizácie a programovania v jazyku C++, ktorý bol použitý pri pracovnom pohovore na pozíciu softvérového vývojára

Na nasledujúcich riadkoch sa nachádza písomný test z algoritmizácie a programovania v jazyku C++. Test je tvorený uzatvorenými otázkami, pričom každá otázka je bodovaná a má vždy práve jednu správnu odpoveď. Test uvádzame aj s riešením a doplňujúcim vysvetlením.

Otázka č. 1: Čo predstavuje nasledujúci príkaz jazyka C++?

```
(*new X(&obj)).k(10, *e);
```

Možné odpovede:

a) Príkaz predstavuje inštanciáciu triedy **X**, inicializáciu založenej inštancie parametrickým konštruktorom a volanie parametrickej inštančnej metódy v súvislosti s alokovanou inštanciou.

b) Sémantiku príkazu nemusíme bližšie skúmať, pretože je chybný. Príkaz sa totiž pokúša inštanciovať anonymnú triedu, čo v jazyku C++ nie je možné.

c) Príkaz definuje λ-výraz (lambda-výraz), ktorý uskutočňuje inštanciáciu triedy **X**.

d) Príkaz vytvárajú hlbokú kópiu pôvodnej inštancie triedy **X** pomocou explicitného kopírovacieho konštruktora.

e) Príkaz generuje vektorové pole inštancií triedy **X**, ktoré sú inicializované parametrickými konštruktormi. Potom sú v súvislosti s inštanciami triedy **X** volané ich parametrické metódy.

Body: 3

Správna odpoveď: a)

Zdôvodnenie: Príkaz vykonáva inštanciáciu triedy **X**, pričom alokovaná inštancia je inicializovaná parametrickým inštančným konštruktorom. Konštruktoru je ako argument odovzdaný smerník na entitu s identifikátorom **obj**. Operátor **new** vracia smerník na inštanciu triedy **X**, ktorý je okamžite podrobený dereferencii. Tak získavame explicitný prístup k inštancii triedy **X**, na ktorej spúšťame parametrickú metódu **k**. Tejto metóde poskytujeme dva argumenty: celočíselnú konštantu (10) a objekt, na ktorý je nasmerovaný smerník uložený v smerníkovej premennej **e**.

Otázka č. 2: Čo predstavuje nasledujúci príkaz jazyka C++?

```
template<typename T, typename U> T* m(const T& a, U* b) { ... }
```

Možné odpovede:

a) Príkaz vykonáva explicitnú inštanciáciu parametrickej šablónovej funkcie.
b) Príkaz vykonáva implicitnú inštanciáciu parametrickej šablónovej funkcie.
c) Príkaz definuje parametrickú šablónovú funkciu.
d) Príkaz je chybný, pretože dátový typ návratovej hodnoty parametrickej šablónovej funkcie nie je správne určený.
e) Príkaz je chybný, pretože ak je jeden formálny parameter šablónovej funkcie konštantný, musí byť takým aj jej druhý formálny parameter.

Body: 2

Správna odpoveď: c)

Zdôvodnenie: Príkaz definuje šablónovú funkciu s identifikátorom **m** a s dvomi typovými formálnymi parametrami **T** a **U**. Z hlavičky šablónovej funkcie vieme vyčítať, že vracia smerník na inštanciu typu **T** a definuje 2 formálne parametre: prvým je referencia na

konštantný objekt typu **T** a druhým je smerník na (nekonštantný) objekt typu **U**. Telo šablónovej funkcie je len symbolicky vyznačené, takže nevieme povedať, aký generický algoritmus šablónová funkcia v skutočnosti implementuje.

Navyše, ak by sme chceli vykonať inštanciáciu šablónovej funkcie, najjednoduchším riešením je implicitná inštanciácia. Tá by mala v prípade šablónovej funkcie nasledujúcu podobu:

```
A* c = m(g, d);
```

V premennej **g** je uložený objekt triedy **A**, ktorý je pre inštanciu šablónovej funkcie konštantný. Premenná **d** je inicializovaná smerníkom na triedu **B** (deklarácie tried **A** a **B** nie sú v kóde uvedené).

Otázka č. 3: Môže byť v tele triedy W definovaný súkromný konštruktor?

```
class W
{
    W() { ... }
};
```

Možné odpovede:

a) Áno, môže. Prítomnosť súkromného konštruktora v tele hocijakej triedy jazyka C++ je validná.
b) Áno, môže, avšak súkromný konštruktor nemôže byť použitý na automatickú inštanciáciu triedy.
c) Áno, syntakticky je uvedený kód triedy správny. Problémom ale je, že klientsky kód nie je schopný založiť inštanciu triedy, ktorá definuje len súkromný konštruktor.
d) Nie, nemôže. Toto je chyba, ktorá bude zachytená prekladačom v čase kompilácie zdrojového kódu programu.

e) Nie, nemôže. Ide o chybu, avšak, bohužiaľ, prekladač ju nedeteguje. Chyba bude generovaná až v režime spracovania programu, pričom zapríčiní jeho abnormálne ukončenie.

Body: 1

Správna odpoveď: c)

Zdôvodnenie: Jazyková špecifikácia pre C++ sa nebráni vloženiu definície súkromného konštruktora do tela triedy. Prekladač ale neumožní klientom triedu inštanciovať, pretože jej konštruktor (ktorý je zodpovedný za náležitú inicializáciu inštancie triedy) nie je prístupný. V jazyku C++ je začlenenie súkromného konštruktora do triedy technikou, ktorá znemožní klientom svojvoľne vytvárať inštancie tejto triedy. Ak má inštancia triedy vzniknúť, musí sa tak stať priamo v jej tele. Riešením je dodanie statickej metódy, ktorá zabezpečí inštanciáciu triedy, inicializáciu zrodenej inštancie a napokon poskytne klientom smerník na okamžite použiteľnú inštanciu triedy. Dodajme, že v podobnom duchu sú v jazyku C++ implementovaní jedináčikovia, teda triedy, ktoré môžu mať maximálne jednu inštanciu (tieto triedy sú pripravené podľa návrhového vzoru Singleton).

Otázka č. 4: Navrhnite deklaráciu triedy, ktorá využíva preťažený konštruktor s inicializačnou časťou.

Možné odpovede:

a)
```
class A
{
    int x; float y;
    public:
        A() { x = y = 1; }
};
```

b)
```
class B
{
    int x; float y;
    public:
        B() : x(1), y(2.3f) { }
};
```

c)
```
class C
```

```
    {
        int x; float y;
        public:
            C() { x = y = 1; }
            C(int a, double b) { x = a; y = b; }
    };
d)  class D
    {
        int x; float y;
        public:
            D() : x(0), y(0) { }
            D(int a, float b) : x(a), y(b) { }
    };
e)  class E
    {
        int x; float y;
        public:
            E() { E(1, 3.5f); }
            E(int x, float y) { this->x = x; this->y = y; }
    };
```

Body: 2

Správna odpoveď: d)

Zdôvodnenie: Preťažiť konštruktor znamená vložiť do tela triedy viacero definícií konštruktora, ktorá sa líšia svojimi signatúrami (teda počtom formálnych parametrov alebo dátovými typmi formálnych parametrov). Konštruktor s inicializačnou časťou je konštruktor, ktorý za signatúrou uvádza symbol dvojbodky (:) a podľa vopred stanoveného syntaktického predpisu inicializuje dátové členy alokovanej inštancie triedy. O preťaženom konštruktore s inicializačnou časťou hovoríme vtedy, keď všetky verzie konštruktora inicializujú dátové členy inštancie pomocou svojich inicializačných častí.

Otázka č. 5: Čo je to mechanizmus typovej inferencie (MTI) a je dostupný v C++?

Možné odpovede:

a) MTI uskutočňuje typovú inferenciu lokálnych entít podľa dátových typov hodnôt ich inicializačných výrazov. Áno, MTI je v C++ dostupný, napríklad: **auto x = e;**.

b) MTI predstavuje automatickú dedukciu dátového typu determinovanej entity. V jazyku C++ však nie je (na rozdiel od C#) tento mechanizmus implementovaný.

c) MTI samočinne odstraňuje typovú nekompatibilitu entít, napríklad v priraďovacích príkazoch. Do jazyka C++ však ešte nebol začlenený.

d) MTI je výsostne prvok objektového programovania, pretože pomocou neho možno implementovať polymorfné správanie inštancií tried.

e) Niečo také ako MTI vôbec neexistuje a ani nie je potrebné. Automaticky inferovať typy entít je zbytočné, veď programátor vždy pozná typy objektov, s ktorými pracuje. Ak náhodou dôjde k typovému nesúladu, tento bude eliminovaný implicitnou alebo explicitnou typovou konverziou.

Body: 2

Správna odpoveď: a)

Zdôvodnenie: Mechanizmus typovej inferencie slúži na strojové odvodenie typov lokálnych objektov podľa ich inicializačných výrazov. Počnúc novým ISO štandardom jazyka C++ (zatiaľ pracovne označovaným ako C++0x) dochádza k implementácii MTI do jazyka C++ a jeho prekladača. Pre úplnosť doplňme, že sa mení správanie kľúčového slova **auto**, ktoré odteraz nereprezentuje špecifikátor pamäťovej triedy objektu, ale skutočnosť, že dátový typ objektu je automaticky dedukovaný.

Otázka č. 6: Čo je to polymorfizmus a ako by ste ho implementovali v jazyku C++?

Možné odpovede:

a) Polymorfizmus je technika, ktorá nám umožňuje vytvárať podtriedy z bázových tried. Implementujeme ho syntaxou pre dedičnosť.
b) Polymorfizmus je schopnosť množiny objektov reagovať odlišne na príjem rovnakej správy. V C++ ho zavedieme dedičnosťou a virtuálnymi funkciami.
c) Polymorfizmus predstavuje spôsob vytvárania abstraktných tried, v telách ktorých sú deklarované čisté virtuálne funkcie.
d) Polymorfizmus je metódou vytvárania hlbokých objektových kópií. V jazyku C++ sa realizuje preťaženým kopírovacím konštruktorom.
e) Objektová teória definuje polymorfizmus ako schopnosť objektov meniť svoje verejné rozhranie. V C++ implementujeme polymorfizmus dynamickou substitúciou rozhrania objektu podľa požiadaviek klientov.

Body: 2

Správna odpoveď: b)

Zdôvodnenie: Všeobecná teória objektového programovania vraví, že polymorfizmus je schopnosť konečnej a neprázdnej množiny objektov reagovať inak na doručenie rovnakého podnetu. V jazyku C++ môžeme polymorfizmus navrhnúť len prostredníctvom dedičnosti. Postupujeme pritom tak, že v bázovej triede definujeme virtuálne metódy, ktoré následne v odvodenej triede prekryjeme metódami s rovnakými rozhraniami, no odlišnou funkcionalitou.

Otázka č. 7: Programátor mal za úlohu deklarovať v jazyku C++ abstraktnú triedu. Postupoval správne?

```
class Q
{
    Q() { }
    public:
        virtual void m(double, long int) = 0;
};
```

Možné odpovede:

a) Nie, pretože nič také ako abstraktná trieda v jazyku C++ neexistuje.
b) Nie, pretože trieda obsahuje chybnú deklaráciu virtuálnej metódy.
c) Nie, pretože trieda obsahuje chybnú definíciu virtuálnej metódy.
d) Áno, trieda je abstraktná. Znamená to, že nemôže byť inštanciovaná (zásluhou súkromného konštruktora) a musí pôsobiť ako bázová trieda (vďaka čistej virtuálnej metóde situovanej v jej tele).
e) Áno, trieda je abstraktná, avšak čistá virtuálna metóda musí byť ešte korektne definovaná mimo jej tela. Inak prekladač zahlási chybu v čase kompilácie.

Body: 3

Správna odpoveď: d)

Zdôvodnenie: Hoci jazyková špecifikácia C++ priamo nedefinuje kľúčové slovo **abstract**, ktoré by sme mohli dodať do hlavičky triedy, vieme si abstraktnú triedu naprogramovať sami. Musíme však vedieť, že abstraktná trieda determinuje len rozhranie odvodených tried, ktoré budú môcť byť inštanciované. Postupujeme preto tak, že do tela abstraktnej triedy umiestnime súkromný konštruktor a množinu čistých virtuálnych metód. Keďže tunajšie metódy sú čisté virtuálne, musia byť odvodenými triedami prekryté (nahradené rovnomennými metódami s kompletnou implementáciou).

Otázka č. 8: Vysvetlite sémantiku nasledujúceho príkazu jazyka C++:

```
(*p).~H();
```

Možné odpovede:

a) Príkaz nemá zmysel, pretože nie je možné explicitne dereferencovať premennú **p**.
b) Príkaz vykonáva finalizáciu a dealokáciu inštancie triedy **H**.
c) Príkaz predstavuje explicitnú aktiváciu deštruktora inštancie triedy **H**. Hoci bude deštruktor spracovaný, nedochádza k dealokácii inštancie tejto triedy.
d) Príkaz explicitne volá deštruktor triedy **H**, čo však nie je v jazyku C++ možné. Deštruktor sa nikdy nevolá priamo, pretože jeho aktivácia je podmienená použitím operátora **delete**.
e) Príkaz finalizuje inštanciu triedy **H**, no táto operácia prebehne v poriadku len vtedy, keď bola inštancia tejto triedy alokovaná automaticky.

Body: 3

Správna odpoveď: c)

Zdôvodnenie: Predložený príkaz iniciuje spracovanie deštruktora v súvislosti s objektom, na ktorý ukazuje smerník uchovaný v smerníkovej premennej **p**. Deštruktor objekt finalizuje, no neuvoľní ho z operačnej pamäte. Výsledkom príkazu je teda finalizovaný, avšak nedealokovaný objekt, čo je situácia, ktorá môže spôsobiť problémy. Ak klient nemá vedomosť o aktuálnom stave objektu, môže od neho žiadať poskytnutie istých služieb. Bohužiaľ, objekt nedokáže garantovať realizáciu vyžiadaných služieb, pretože nie je dátovo konzistentný. To vedie k vzniku kolízneho stavu v programe. Až na pár ojedinelých výnimiek je v praxi podobná technika považovaná za mimoriadne nebezpečnú. Programátorom odporúčame vždy finalizovať a uvoľňovať dynamické inštancie tried pomocou operátora **delete**.

Otázka č. 9: Aký výstup bude generovať nasledujúci program jazyka C++?

```
#include <iostream>
using namespace std;
class A
{
private:
    static unsigned int p;
public:
    A() {p++;}
    ~A() {p--;}
    static unsigned int m() {return p;}
};
unsigned int A::p = 0;
int main()
{
    A *a;
    A *pole[10];
    for(int i = 1; i <= 10; i++)
    {
        a = new A();
        pole[i - 1] = a;
    }
    for(int i = 0; i < 10; i++)
        delete pole[i];
    cout << A::m() << endl;
    return 0;
}
```

Možné odpovede:

a) 10
b) 9
c) 11
d) 8
e) 0

Body: 2

Správna odpoveď: e)

Zdôvodnenie: Trieda **A** je navrhnutá tak, aby dokázala poskytovať informácie o aktuálnom počte svojich existujúcich inštancií. Vo funkcii **main** vytvárame 10 dynamických inštancií triedy **A**, pričom smerníky na ne ukladáme do vektorového poľa. Potom všetky žijúce inštancie triedy **A** finalizujeme a dealokujeme. Vzhľadom na to, že po tejto akcii už neexistuje žiadna inštancia triedy **A**, program vypíše nulovú hodnotu.

Otázka č. 10: Charakterizuje validitu nasledujúceho príkazu jazyka C++:

```
T *obj = new U(...);
```

Možné odpovede:

a) Príkaz nie je platný, pretože sa líšia dátové typy entít na oboch stranách priraďovacieho operátora.

b) Príkaz nie je platný, pretože do smerníkovej premennej typu **T*** nemôžeme priradiť smerník na dynamickú inštanciu triedy **U**.

c) Príkaz je platný, avšak len vtedy, keď je **T** bázová trieda a **U** je odvodená trieda z triedy **T**.

d) Príkaz je platný vždy, a to bez ohľadu na to, či medzi triedami **T** a **U** existuje puto dané dedičnosťou.

e) Príkaz nie je platný, pretože hodnotou inicializačného výrazu je objekt a nie smerník na objekt. Za týchto okolností bude volaný kopírovací konštruktor, avšak kópiu objektu nemožno uložiť do smerníkovej premennej.

Body: 3

Správna odpoveď: c)

Zdôvodnenie: Predostretý príkaz prakticky demonštruje implementáciu substitučného princípu objektového programovania v jazyku C++. Ako vieme, inštancia odvodenej triedy sa môže vyskytnúť všade tam, kde je očakávaná inštancia bázovej triedy. Analogicky, smerník na inštanciu odvodenej triedy môže byť dosadený všade tam, kde sa očakáva smerník na inštanciu bázovej triedy. Medzi triedami **T** a **U** musí existovať vzťah dedičnosti, pričom **T** je bázová trieda a **U** je odvodená trieda.

5.2 Vyhodnotenie ukážkového písomného testu algoritmizácie a programovania v jazyku C++

Písomný test mal 10 otázok, pričom každá z otázok disponovala svojím bodovým ohodnotením. Po dokončení testu sa vytvorí sumárna tabuľka (tab. A), v ktorej sa vyznačí dosiahnuté skóre uchádzača.

Tab. A: Vyhodnotenie písomného testu algoritmizácie a programovania v jazyku C++

Otázka	Maximálny počet bodov	Získaný počet bodov uchádzača
1.	3	
2.	2	
3.	1	
4.	2	
5.	2	
6.	2	
7.	3	
8.	3	
9.	2	
10.	3	
Spolu	**23**	(doplniť skutočne dosiahnuté bodové ohodnotenie uchádzača)

Pre úspešné absolvovanie písomného testu je potrebné získať aspoň 85 % z celkového bodového ohodnotenia, čo konkrétne znamená zisk viac ako 19 bodov.

O autorovi

Ing. Ján Hanák, PhD., MVP, vyštudoval Ekonomickú univerzitu v Bratislave. Tu, na Katedre aplikovanej informatiky Fakulty hospodárskej informatiky (KAI FHI), pracuje ako vysokoškolský pedagóg. Prednáša a vedie semináre týkajúce sa programovania a vývoja počítačového softvéru v programovacích jazykoch C, C++ a C#. K jeho favoritom patrí tiež Visual Basic, F# a C++/CLI.

Je nadšeným autorom odbornej počítačovej literatúry. V jeho portfóliu môžete nájsť nasledujúce knižné tituly:

1. **Ako sa stať softvérovým vývojárom**. Bratislava: Microsoft Slovakia, 2010.
2. **C++: Akademický výučbový kurz**. Bratislava: Vydavateľstvo EKONÓM, 2010.
3. **Inovácie v jazykoch Visual Basic 2010, C# 4.0 a C++**. Brno: Artax, 2010.
4. **Programování v jazyce C**. Kralice na Hané: Computer Media, 2010.
5. **Visual Basic 2010 – Hotové riešenia**. Bratislava: Microsoft Slovakia, 2010.
6. **C#: Akademický výučbový kurz, 2. aktualizované a rozšírené vydanie**. Bratislava: Vydavateľstvo EKONÓM, 2010.
7. **Praktické objektové programování v jazyce C# 4.0**. Brno: Artax, 2009.
8. **Praktické paralelné programovanie v jazykoch C# 4.0 a C++**. Brno: Artax, 2009.
9. **C++/CLI - Praktické príklady**. Brno: Artax, 2009.
10. **C# 3.0 - Programování na platformě .NET 3.5**. Brno: Zoner Press, 2009.
11. **C++/CLI - Začínáme programovat**. Brno: Artax, 2009.
12. **C#: Akademický výučbový kurz**. Bratislava: Vydavateľstvo EKONÓM, 2009.
13. **Základy paralelného programovania v jazyku C# 3.0**. Brno: Artax, 2009.
14. **Objektovo orientované programovanie v jazyku C# 3.0**. Brno: Artax, 2008.

15. **Inovácie v jazyku Visual Basic 2008**. Praha: Microsoft, 2008.
16. **Visual Basic 2008: Grafické transformácie a ich optimalizácie**. Bratislava: Microsoft Slovakia, 2008.
17. **Programovanie B – Zbierka prednášok (Učebná pomôcka na programovanie v jazyku C++)**. Bratislava: Vydavateľstvo EKONÓM, 2008.
18. **Programovanie A – Zbierka prednášok (Učebná pomôcka na programovanie v jazyku C)**. Bratislava: Vydavateľstvo EKONÓM, 2008.
19. **Expanzívne šablóny: Príručka pre tvorbu "code snippets" pre Visual Studio**. Bratislava: Microsoft Slovakia, 2008.
20. **Kryptografia: Príručka pre praktické odskúšanie symetrického šifrovania v .NET Framework-u**. Bratislava: Microsoft Slovakia, 2007.
21. **Príručka pre praktické odskúšanie vývoja nad Windows Mobile 6.0**. Bratislava: Microsoft Slovakia, 2007.
22. **Príručka pre praktické odskúšanie vývoja nad DirectX**. Bratislava: Microsoft Slovakia, 2007.
23. **Príručka pre praktické odskúšanie automatizácie aplikácií Microsoft Office 2007**. Bratislava: Microsoft Slovakia, 2007.
24. **Visual Basic 2005 pro pokročilé**. Brno: Zoner Press, 2006.
25. **C# – praktické příklady**. Praha: Grada Publishing, 2006 (kniha získala ocenenie „Najúspešnejšia novinka vydavateľstva Grada v oblasti programovania za rok 2006").
26. **Programujeme v jazycích C++ s Managed Extensions a C++/CLI**. Praha: Microsoft, 2006.
27. **Přecházíme z jazyka Visual Basic 6.0 na jazyk Visual Basic 2005**. Praha: Microsoft, 2005.
28. **Visual Basic .NET 2003 – Začínáme programovat**. Praha: Grada Publishing, 2004.

V rokoch 2006 – 2010 bol jeho prínos vývojárskym komunitám ocenený celosvetovými vývojárskymi titulmi **Microsoft Most Valuable Professional (MVP)** s kompetenciou **Visual Developer – Visual C++**.

Ing. Ján Hanák, PhD., MVP, je nositeľ titulu Microsoft Most Valuable Professional s kompetenciou Visual Developer – Visual C++. Je autorom 28 odborných kníh, príručiek a praktických cvičení o programovaní a vývoji počítačového softvéru. Pracuje ako vysokoškolský pedagóg na Katedre aplikovanej informatiky Fakulty hospodárskej informatiky Ekonomickej univerzity v Bratislave. Prednáša a vedie semináre z programovania v jazykoch C, C++ a C#.

V rámci svojej vedeckej činnosti sa zaoberá problematikou štruktúrovaného, objektovo orientovaného, komponentového, funkcionálneho a paralelného programovania.