Развој софтвера 2

2. Метафоре за боље разумевање развоја софтвера

2.1. Значај метафора

Важан напредак у неком пољу често потиче од аналогија. Поређењем теме коју слабо разумемо са нечим што боље разумемо може се доћи до увида који резултирају бољим схватањем теме са којом смо мање фамилијарни. Овакво коришћење метафора се назива **моделирање**.

Историја науке је пуна открића која су заснована на експлоатисању моћи метафора:

•Хемичар Кекуле је сањао и у сну видео змију која је загризла свој реп. Када се пробудио, схватио је да молекуларна структура заснована на сличном прстену може представљати структуру бензена. Даљи експерименти су потврдили његову хипотезу.

•Кинетичка теорија флуида је била заснована на моделу билијарске кугле. Сматрало се да молекули флуида имају масу и да се одбијају при сударима, као што је то случај са билијарским куглама, па су из овог модела изведене многе корисне теореме.

•Таласна теорија светлости је великим делом развијена тако што су експлоатисане сличности између светлости и звука. Наиме, и светлост и звук имају амплитуде, фрекфенцију и друге заједничке особине.

2.1. Значај метафора

Поређење између таласне теорије звука и таласне теорије светлости је било толико продуктивно да су научници улагали велике напоре да би пронашли медијум кроз који се простире светлост као што се звук простире кроз ваздух. Том медијуму су чак били и дали име - етар – али га никад нису пронашли. У овом случају је аналогија била толико замамна да би се на крају показало да је водила у погрешном правцу.

Уопштено гледано, моћ модела је у њиховој живости и у томе што се могу дохватити и користити као концептуалне целине.Модели сугеришу особине, односе и области даљег испитивања. Понекад модел сугерише области испитивања које не воде у правом смеру, у ком случају је метафора певише растегнута. На пример, у метафора је била превише растегнута у ситацији када су научници тражили етар.

Као што се и може очекивати, неке метафоре су боље од других. Добра метафора је једноставна, добро се укалапа са осталим релевантним метафорама и објашњава велики део експерименталних резултата и уочених чињеница.

2.1. Значај метафора

Размотримо пример тешког камена окаченог о нит који се креће напред-назад. Пре Галилеја су следбеници Аристотела посматрали кретаље камена напред-назад тако што се тешки објекат природно креће од више позиције нижој тј. према стању мировања. Дакле, следебеници Аристотела су сматрали да у ствари камен отежано пада. Када је Галилео видео камен који се креће на горе описани начин, он је видео клатно. Он је сматрао да камен понавља стално исто кретање, изнова и изнова, на скоро савршен начин.

Сугестивне моћи ова два модела су сасвим различите. Аристотелијанци који су посматрали камен који се креће као објекат који пада наниже су приметили његову тежину, висину на који је подигнут пре почетка кретања и време које је потребно да се камен умири. За Галилеов модел клатна су други фактори били истакнути. Галилео је посматрао тежину камена, дужину нити клатна, угаони померај и време потребно за један замах клатна. Галилео је открио законе кретња које Аристотелијанци нису могли открити, зато што их је модел који су усвојили усмеравао да гледају према другим појавама и да постављају другачија питања.

2.1. Значај метафора

Метафоре настављају да доприносе бољем разумевању ставри у развоју софтвера на исти начин као што доприносе бољем разумевању питања науке. У свом предавању приликом доделе Тјурингове награде, Бакман је 1973. године описао промене са преовлађујућег геоцентричног према хелиоцентричном погледу на универзум. Птоломејев геоцентрични модел је трајао више од 1400 година без озбиљног изазова. Потом је 1543. године увео хелиоцентричну теорију, са идејом да је Сунце (а не Земља) центар универзума. Ова промена у менталном моделу је у крајњем довела до открића нових планета, до класификације месеца као сателита (а не као планете) и до другачијег схватања о позицији човечанства у универзуму.

Бакман је упоредио промену у астрономији од Птоломеја до Коперника са променама у рачунарском програмирању у раним седамдесетим. У тренутку кад је прављено поређење, процесирање података се померало са рачунарски-центрираног погледа на база-центриран поглед. Бакман је истакао да су старији у процесирању података желели да виде све податке као секвенцијални ток картица који пролази кроз рачунар (рачунарски-центриран поглед). Преомена је захтевала да се фокусира на податке над којима рачунар ради (база-центриран поглед).

2.1. Значај метафора

Данас је тешко замислити да било ко може мислити како се сунце окреће око земље. Слично томе, тешко је замислити да било ко може мислити да се сви подаци могу посматрати као секвенцијални ток картица. У оба случаја, једном када је одбачена стара теорија, постало је немогуће да икад ико више верује у њу. Још интересантније, људи који су веровали у стару теорију сматрали су да је нова теорија исто тако смешна и чудна као што ми сада мислимо за стару теорију.
Изазовно је да се тривијализује моћ метафора. У сваком од ранијих примера је природан следећи одговор: “Наравно да је права метафора кориснија. Она друга метафора је погрешна!”. Иако је то природан одговор, то је превише поједностављено.Историја науке није серија прелазака са “погрешне” метафоре на “праву”. То је серија промена са “горе” метафоре на “бољу”, са мање обихватне на обухватнију, са сугестивен у једном подручју на сугестивну у другом подручју.
У ствари, многи модели који су замењени са бољим моделима су и даље корисни. Инжињери и даље највећи број проблема решавају коришћењем Њутнове механике, иако је, теоретски гледано, она надограђена Ајнштајновом теоријом.

2.2. Како користити софтверске метафоре

Софтверска метафора је више као светло за тражење него као мапа пута. Оно не говори где се може наћи одговор, већ говори како да се тражи одговор. Метафора више служи као хеуристика него као алгоритам.

Алгоритам је секвенца добро дефинисаних инструкција за извршавање конкретног задатка. Алгоритам је предвидив, детерминистичан и не подлеже случајности. Алгоритам говори како да се стигне од тачке A до тачке B без скретања до тачака D, E и F и без заустављања ради мирисања ружа или испијања пића.

Хеуристика је техника која помаже у тражењу одговора. Њен резултат зависи и од среће зато што хеуристика само каже како да се тражи, а не шта ће се наћи. Она неће рећи како да се директно дође од тачке A до тачке B; она чак и не мора знати где су тачке A и B. Другим речима, хеуристика је алгоритам у кловновском оделу – много мање је предвидива и много је забавнија.

Пример: упутство за путовање.

2.2. Како користити софтверске метафоре

Како користити софтверске метафоре? Треба их користити ради добијања увида у програмерске проблеме и процесе. Користити их као помоћ у размишљању о програмерским активностима и као помоћ у замишљању бољих начина да се реализују послови.

Није могуће посматрати једну линију кода и рећи да она крши неку од метафора које ће се овде описати. С временом, ипак, особе које користе метафоре у осветљавању процеса развоја софтвера буду препознати као неко ко има боље разумевање програмирања, ко производи бољи код и ко то ради брже од људи који не користе метафоре.

2.3. Уобичајене софтверске метафоре

Око развоја софвера је израсло збуњујуће обиље метафора:
• Брукс је рекао да је писање софтвера као узгајање биља, лов на вукодлаке или дављење са диносаурусима у јами са катраном (1995).
• Грис је рекао да је то наука (1981).
• Кнут је рекао да је то уметност (1998).
• Хамфри је рекао да је то процес (1989).
• Плаугер (1993) и Бек (2000) су рекли да је то као вожња аутомобила.
• Кокбурн је рекао да је то игра (2001).
• Рејмнонд је рекао да је то као базар (2000).
• Хекел је рекао да је као снимање филма “Снежана и седам патуљака” (1994).

Које метафоре су најбоље?

2.3.1. Софтверски краснопис: писање кода

Најпримитивнија метафора за развој софтвера израста из израза “писање кода”. Метафора писања сугерише да је развој програма као писање писма – аутор седне са оловком и папиром и напише писмо од почетка до краја. То не захтева никакво формално планирање и како писање иде аутор закључује шта жели да саопшти.

Из ове метафоре су изведене многе идеје.
Бентли каже да треба да се буде у стању да се седне крај камина са чашом брендија, добром цигаром и омиљеним ловачким псом како би се уживало у исчитавању програма на исти начин као што је то случај са добрим романом.
Керинген и Плаугер су своју књигу о стиловима програмирања из 1978. назвали “Елементи *програмерског* стила” по аналогији са насловом познате књиге “Елементи стила”, аутора Струка и Вајта, која се бави стиловима писања текста.
Програмери често говоре о “читљивости програма”.

2.3.1. Софтверски краснопис: писање кода

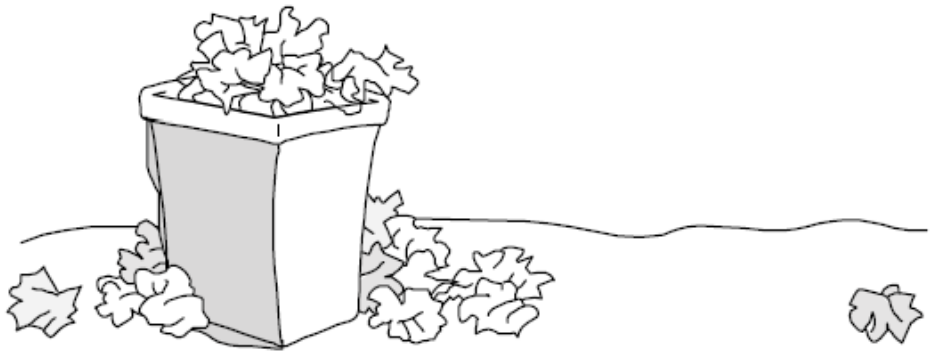
За појединца који ради на малом пројекту, метафора писања писма адекватно функционише. Међутим, за све остале ситуације рано долази до мимоилажења са реалношћу – ова метафора не описује развој софтвера адекватно нити у потпуности. Писање је обично **активност једне особе**, док софтверски пројекат обично укључује већи број људи са различитим одговорностима. Када се заврши са писањем писма, оно се стави у коверту и пошаље поштом. Више **није могуће да се** то писмо **мења** и његова сврха је испуњена слањем. Софтвер се често мења и за њега тешко икад можемо рећи да је потпуно завршен. И до 90% од укупног напора у развоју софтверског система долази после иницијалне испоруке, мада типично неких две трећине напора наступа после иницијалне испоруке (Pigoski 1997).

При писању велики значај има **оригиналност**. При конструкцији софтвера, покушај креирања оригиналног дела је често мање ефикасан од фокусирања на поновно коришћење идеја дизајна, кода и случајева тестирања направљених у претходним пројектима. Укратко, метафора писања ипмлицира да је процес развоја софтвера превише кратак и ригидан, па не може бити “здрава”.

2.3.1. Софтверски краснопис: писање кода

На несрећу, метафора писања је овековечена у једној од најпопуларнијих књига о развоју софтвера “*The Mythical Man-Month*”, аутора Фреда Брукса. Брукс пише: “Планирајте да то одбаците; у сваком случају то ће се догодити ”. Ово се дочарава сликом гомиле полуисписаних радних верзија које су бачене у корпу за отпатке. Планирање да се одбаци може бити практично када се пише учтиво писмо за тетку и може бити најмодернија пракса у развоју софтвера у 1975, када је написана књига.

Али, проширење метафоре писања софтвера на план како одбацити је лош савет за развој софтвера у 21. веку, када велики софтверски системи коштају колико десет зграда или колико прекоокеански брод.



2.3.2. Узгајање софтвера: раст система

У супротности са ригидном метафором писања, неки програмери су истакли да се треба предвидети креирање софтвера као нечега што подсећа на сејање усева или узгајање биља. Један део се дизајнира, део се кодира, део тестира и додаје се у систем мало по мало. Радећи у малим корацима, минимизира се могућност да се у неком тренутку дође у проблем.

Понекад се добра техника опише са лошом метафором. У таквим случајевима, треба покушати да се задржи техника и да се пронађе боља метафора. У горњем случају, инкрементална техника представља вредност, али метафора пољопривреде је ужасна. Идеја рада мало по мало током времена можда има неке сличности са начином како усеви расту, али аналогија са фармом је слаба и неинформативна, тако да ју је лако заменити са бољим метафорама.

2.3.2. Узгајање софтвера: раст система

Тешко је проширити метафору пољопривреде ван једноставне идеје рада део по део током времена. Ако би се проширила та метафора, дошло би се у ситуацију да се говори о нађубривању плана система, орезивању детаљног дизајна, повећању приноса кода кроз ефективно управљање земљиштем и о жетви самог кода. Могло би се догодити да се говори о остављању да се земљиште одмори на годину дана како ди се повећао ниво азота на чврстом диску.

2.3.3. Софтверска фарма острига: увећање система

Понекад људи говоре о расту софтвера а да при том мисле на увећање тј. прирастање, срастање. Те две метафоре су блиско повезане, али метафора софтверског увећања даје проницљивију слику. Увећање (енг. accretion) означава ма какав раст или повећање прираштањем тј. постепеним спољашњим додавањем или укључењем.

Увећање описује начин на који острига прави бисер, тако што постепено додаје мале количине калцијум карбоната. У геологији увећање (тј. седиментација) означава споро додавање воденог седимента земљи. У праву увећање означава повећање парцеле дуж обале тако што је вода додала седиментни садржај.

Ово не значи да треба да се научи како да се прави ко од седиментног материјала, већ како да се у софтверски ситем додају мали делови током времена. Друге речи које су блиско повезане са увећањем су “инкрементално”, “итеративно”, “адаптивно” и “еволутивно”. Инкрементални дизајн, израдња и тестирање су неки од најмоћнијих појмова у развоју софтвера.

2.3.3. Софтверска фарма острига: увећање система

У инкременталном развоју се прво направи најједноставнија могућа верзија система који ће радити. Тај систем не мора да прихвата реалистичан улаз, не мора да реализује реалистичку манипулацију над подацима и не мора да производи реалистичан излаз – он само треба да буде костур који је довољно јак да држи реални систем током његовог развоја. Он може да позива класе-макете за сваку од основних функција које су идентификоване. Овај основни почетак је као што острига започиње креирање бисера око малог зрнца песка.

Пошто се формира костур, мало по мало се додају мишићи и кожа. Свака од класа-макета се замењује реалном класом. Уместо да се програм претвара да прихвата улаз, убацује се код који омогућује прихватање стварног улаза. Уместо да се програм претвара да производи излаз, убацује се код који производи реални излаз. На тај начин се додаје мало по мало кода током времена све док се не добије потпуно функционалан систем.

2.3.3. Софтверска фарма острига: увећање система

Импресивни су анегдотски докази у корист оваквог приступа развоју софтвера.
Брукс, који је 1975. у књизи *The Mythical Man-Month* саветовао одбацивање, је рекао да ништа није тако радикално променило његову праксу и њену ефикасност као што је то случај са инкременталним развојем (1995).
Глиб је то исто истакао у његовој књизи *Principles of Software Engineering Management* (1988). Та књига је увела методологију еволутивна испорука и поставила темеље за највећи део данашњих агилних методологија.

Гледано као метафора, снага инкременталне метафоре је у томе што она не обећава превише. Њу је теже проширити на неадекватан начин него што је то случај са метафором пољопривреде. Слика остриге која формира бисер је добар начин за визуелизацију инкременталног развоја, тј. увећања.

2.3.4. Конструкција софтвера: изградња софтвера

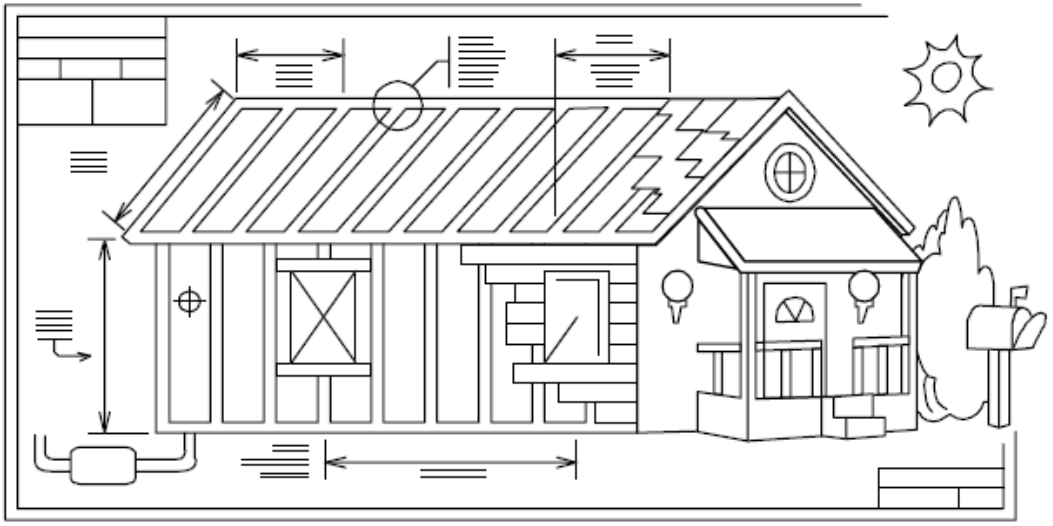
Слика “грађења” софтвера је много кориснија од “писања” или “узгајања” софтвера. То је компатибилно са идејом увећања софтвера и обезбеђује детаљније вођење. Грађење софтвера имплицира различите нивое планирања, припрема и радова који варирају по врсти и по степену у зависности од тога шта се гради. Када се истражује метафора, налазе се и друге паралеле. Градња четвороспратне куле од конзерви захтева мирну руку, равну површину и 10 неоштећених конзерви. За грађење сто пута веће куле није довољно имати на располагању сто пута више конзерви, већ је неопходна другачија врста планирања и конструкције.
Ако се гради једноставна структура, нпр. штенара, потребно је купити даске и ексере. До краја поподнева, Џеки ће имати своју кућицу. Ако се заборави да се обезбеде врата или се направи нека друга грешка, то није велики проблем – може се поправити или чак почети од почетка. Све што је изгубљено је део поподнева.

Овај опуштен приступ је одговарајући и за мале софтверске пројекте. Ако сте користили погрешан дизајн за 1000 линија кода, можете их рефакторисати или кренути од почетка а да нисте пуно изгубили.

2.3.4. Конструкција софтвера: изградња софтвера

Ако се гради кућа, тада је процес градње компликованији, па су озбиљније консеквенце лошег дизајна. Прво мора да се одлучи која врста куће ће се градити, што је у развоју софтвера налогно са дефиницијом проблема. Потом архитекта мора да креира општи дизајн и тај дизајн мора да буде одобрен, што је слично са архитектонским дизајном софтвера. Потом се цртају детаљни планови и ангажује извршилац радова. То је слично са детаљним дизајном софтвера. При градњи се прво припреми градичиште, ископају темељи, излије плоча, озидају зидови, подигне кров, омалтерише, уведе вода, струја и канализација. Ово је слично конструкцији софтвера. Кад је највећи део куће завршен, молери и тапетари долазе да начине најбољом кућу која се гради. Ово је слично софтверској оптимизацији. Током процеса, различите инспекције долазе на градилиште да провере темеље, зидове, инсталације итд. То је слично прегледу софтвера, програмирању у паровима и инспекцијама.

2.3.4. Конструкција софтвера: изградња софтвера



Већа сложеност и величина имплицирају веће консеквенце за све активности. Током градње куће, материјали јесу скупи, али главни трошак је људски рад. Уклањање постављеног преградног зида и његово померање за 30 cm је скупо не зато што се троши материјал, већ зато што се мора платити радницима додатно време потрошено на премештање зида. Потребно је направити што је могуће бољи дизајн, тако да се не троши време на поправку грешака које су могле да буду избегнуте. И у градњи софтверског производа материјали су још јефтинији а рад је још скупљи. Тако, на пример, промена формата извештаја је скупа као померање зида јер је у оба случаја главна компонента која утиче на цену време радника.

2.3.4. Конструкција софтвера: изградња софтвера

Које још паралеле деле ове две активности? При грађењу куће, нећете покушавати да направите ствари које можете купити већ направљене. Купићете машину за веш, за прање суђа, фрижидер и замрзивач. Нећете разматрати опцију да их сами правите. Такође ћете купити већ направљене ормане, прозоре, врата и делове купатила. Исто ћете урадити и када градите софтверски систем. Интензивно ћете користити какрактеристике високих програмских језика, а нећете писати код на нивоу оперативног система. Такође можете користити већ направљене библиотеке контејнерских класа, научних функција, класа са корисничким интерфејсом и класа за манипулацију са базом података. У општем случају, нема смисла да се кодирају ствари које могу да се купе.
Ако се прави луксузна кућа са првокласним намештајем, ондам се може догодити да ормари буду специјално прављени за кућу. Машине за суђе и за веш, као и фрижидер ће бити уграђене у цпецијално прављене ормаре. Надаље, и прозори могу бити специјлано направљени, са неуобичајеним облицима и величинама. Ово подешавање има паралалу у развоју софтвера. Ако се прави првокласни производ, може се одлучити да се праве сопствене научне функције ради побољшања брзине и ефикасности. Може се одлучити за изградњу сопствених контејнерских класа или GUI класа.

2.3.4. Конструкција софтвера: изградња софтвера

И градња куће и градња софтвера имају користи од адекватног нивоа планирања. Ако се софтвер гради погрешним редоследом, онда га је тешко кодирати, тешко га је тестирати и тешко га је дебагирати. Требаће дуже да се комплетира такав код, а може се десити и да пројекат пропадне јер је рад сваког учесника превише сложен и крајњи призвод, добијен када се сви такви делови саставе, бива превише збуњујућ – чиме је онемогућен даљи рад.

Пажљиво планирање не мора значити да треба превише планирати. Може се испланирати само структурна подршка, а после одлучити каква је врста пода и шта иде на зидове, која ће боја бити коришћена, којим ће се материјалом покривати кров итд. Добро планиран пројекат повећава могућност да се касније предомислимо око неких детаља. Што смо искуснији у градњи одговарајућег софтвера, то више детаља можемо узети са сигурношћу. Само је потребно да се довољно испланира тако да недостатак плана не доведе касније до озбиљних проблема.

2.3.4. Конструкција софтвера: изградња софтвера

Аналогија са грађењем такође помаже у објашењу зашто различити софтверски пројекти имају користи од различитих приступа за развој. У грађевинарству се користе различити нивои планирања, дизајна и обезбеђења квалитета у зависности од тога да ли се прави складиште или трговачки центар или медицински центар или нуклеарни реактор.

Даље, различити су приступи у зависности од тога да ли се гради школа, небодер и кућа. Слично томе, и у развоју софтвера се могу користити флексибилнији тј. лакши приступ развоју или нешто ригиднији тј. тежи приступ који се захтева када треба обезбедити постизање сигурносних и других циљева.

Прављење промена у сосфтверу доводи до још једне паралеле са грађевинарством. Помераље зида за 30 cm је много лакше, једноставније и јефтиније уколико тај зид није носећи. На сличан начин, прављење структурних промена у програму кошта више него додавање или уклањање периферијских карактеристика.

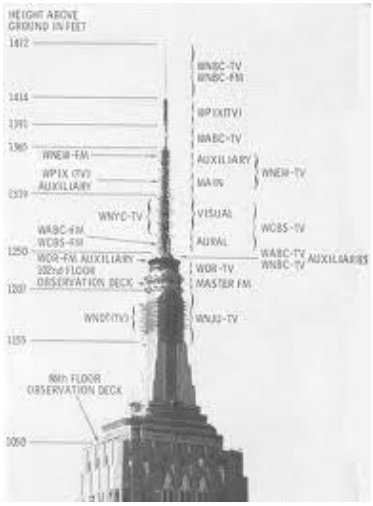
2.3.4. Конструкција софтвера: изградња софтвера

На крају, аналогија са грађењем обезбеђује боље разумевање изузетно великих софтверских пројеката. Будући да је казна за неуспех у градњи екстемно великих структура изузетно оштра, то се традња таквих структура мора темељно припремити. Градитељи морају да пажљиво припреме и брижно испитају своје планове. Они морају да граде у оквиру граница толеранције сигурности – боље је платити 10% више за бољи материјал него доћи у ситуацију да солитер може да се сруши. Велика пажња се поклања постављању временских оквира. Када се градио Empire State Building, сваки камион за испоруку је имао петнаестоминутну маргину током које се морала извршити испорука. Ако камион не би стигао на време, онда би каснио цео пројекат.

Слично томе, и код изузетно великих софтверских пројеката планирање треба да буде вишег реда него што је то случај код осталих пројеката. Каспер Џонс (1998) је утврдио да софтверски систем са милион линија кода захтева 69 врста докумената у просеку. Спецификација захтева за такав систем ће обично имати око 4000-5000 страна, а документација која описује дизајн ће бити два до три пута обимнија од спецификације захтева. Тешко је очекивати да ће једна особа бити у стању да схвати целокупни дизајн тако великог пројекта. Стога је неопходан виши ниво припреме.

Конструкција софтвера: изградња софтвера

Када се реализују софтверски пројекти који су по економској величини упоредиви са градњом зграде Empire State Building, тада и техничка и менаџерска контрола морају морају бити одговарајућег обима.



Аналогија се може проширити у већем броју праваца, и то је разлог зашто је метафора грађења тако моћна. Многи уобичајени термини софтверског инжињерства потичу од метафоре грађења: софтверска архитектура, постављање скела (енг. scaffolding), конструкција, цепање кода (енг. tearing code apart), прикључивање класе (енг. plugging in a class). Вероватно ћете чути још много таквих термина.

Примена софтверских техника: интелектуални алати

Људи који су ефикасни у развоју квалитетног софтвера су провели године у раду и током тог периода акомулирали десетине технологија, трикова и “магчиних” чини. Технике које користе нису чврста правила, већ се ради о аналитичким алатима. Вешт мајстор зна који је алат прави за дати посао и како се тај алат исправно користи.

Иста је ствар и са програмерима. Што више знате о програмирању, то вам је скуп менталних алата испуњенији са аналитичим помагалима и са знањем када да се та помагала користе и како да се користе на прави начин.

Софверски консултанти често сугеришу комплетно коришћење одређених метода за развој уз искључивање свих осталих метода. То често није срећна одлука, јер ако се 100% ослоните на једну методологију, онда ћете цео свет посматрати кроз појмове те методологије. На тај начин ћете пропустити могућност коришћења неке друге методе која је можда боље прилагођена текућем проблему. Метафора скупа алата помаже да сви методи, технике и перспективе буду спремне за коришћење у правом тренутку.

2.4. Комбиновање метафора

Будући да метафоре имају хеуристичку, а не алгоритамску природу, оне не искључују једна другу. Тако је могуће истовремено користити и метафору увећања и метафору грађења. Ако је потребно, може се користити и метафора писања, комбинована са метафором вожње, лова на вукодлаке или куповине. Могу се користити било које метафоре и њеихове комбинације којима се стимулише процес размишљања при решавању проблема.

Коришћење метафора је мутан (енг. fuzzy) посао. Метафоре треба проширити тако да се добије користи од хеуристичких увида које те метафоре пружају. Али, ако се метафоре превише прошире у погрешном правцу, оне могу навести на криви пут. Исто као што се може злоупотребити моћан алат, тако се могу злоупотребити и метафоре. Међутим, моћ метафора их чини вредним делом интелектуалног скупа алата сваког програмера.

2.5. Рекапитулација

● Метафоре су хеуристике, а не алгоритми и као такве имају тенденцију да буду помало траљаве.
● Метафоре помажу у схватању процеса развоја софтвера тако што се формира њихов однос са другим активностима које су већ познате од раније.
● Неке метафоре су боље од других.
● Третирање развоја софтвера на сличан начин као грађења објеката усгерише да је неопходна пажљива припрема. Такво третирање осветљава разлику између великих и малих пројеката.
● Размишљање о праксама у развоју софтвера као елементима интелетиуалног скупа алата сугерише да сваки од програмера поседује разне алате и да не постоји јединствен алат који је погодан за сваки посао. Избор правог алата за сваки од проблема је један од кључева за успешан рад програмера.