Агилни развој софтвера

Развој софтвера (3)

проф. др Милан Гњатовић

Факултет за компјутерске науке
Универзитет Џон Незбит

22.03.2016.

Садржај

Са претходних предавања

- На првом предавању смо дискутовали о важности увођења инжењерске дисциплине у процес развоја софтвера.
- На другом предавању смо разматрали како моделовање процеса развоја софтвера уводи инжењерску дисциплину.
- Данас дискутујемо становиште да моделовање процеса није увек довољан услов за развој квалитетног софтвера.

О „херојском“ приступу

- Развојни процес је користан, али не и од пресудне важности за успех софтверских пројеката.
- Проблеми у развоју софтвера нису јасно одређени и приступ њиховом решавању се не може увек формализовати.
- Развој софтвера је адаптивни систем, а његов најважнији фактор је човек.
  - „Херој“ је (стручни) појединац који преузима иницијативу у решавању нестандардних проблема.
  - „Херојство“ се огледа у спремности да се изађе из оквира формализованих модела.
  - „Херојство“ може бити рационално и патолошко.

Приступ 1. Котролисање процеса

- Претпоставка: Контролисањем квалитета процеса развоја може се постићи квалитет софтверског производа.
- Овај приступ не предвиђа хероје — акценат је на процесу.
- Моделовање процеса је једино што је потребно за успех софтверског пројекта.
- Главну улогу у овом приступу играју руководиоци на врху хијерархије управљања.

Приступ 2. Фокус на људима (енгл. *peopleware*)

- Претпоставка: Софтвер је резултат сарадње између људи.
- Пажњу треба усмерити на развијање сарадње, размишљања, софтверских метода[1], ...
- ... и стварање услова рада у којима се ове вештине могу ефективно применити.
- Овај приступ афирмише рационално херојство:
  - Људи се посматрају као саставни део процеса.
  - Приступ се базира на мотивисаним и подржаним тимовима људи, а не искључиво на надареним појединцима.
- Главну улогу у овом приступу играју руководиоци на нижим нивоима хијерархије управљања.

[1] Подсетник са претходног предавања: појам модела процеса није исто што и појам софтверске методе.

Приступ 3. „Каубојски“ приступ

- Претпоставка: Надареним људима нису потребни вођство и подршка да би креирали софтвер.
- Овај приступ афирмише патолошко херојство — негира се важност моделовања процеса и тимског рада.
- Главну улогу у овом приступу играју појединци.
- Зашто је овај приступ ризичан?
  - Појединци имају иманентна биолошка ограничења. Нпр., стрес умањује способност појединца за доношење исправних одлука[2].
  - Моделе можемо схватити као листе корака које појединцу у стресној ситуацији помажу да не превиди критичне аспекте развоја софтвера, и не донесе погрешне одлуке.

[2] „Хипоксични хероји су рањиви“, в. стр. 9.

Интермецо

- Придев „каубојски“ се овде употребљава у пејоративном смислу, са намером да асоцира на програмера који се „одметнуо“ од модела процеса.

 [3]

[3] The Great Train Robbery (1903), source: The Kobal Collection, author: Edwin S. Porter, http://en.wikipedia.org/wiki/File:Great_train_robbery_still.jpg.

Интермецо

- Шта значи фраза „хипоксични херој“?
- Хипоксија је стање организма изазвано недостатком кисеоника — нпр., на великим висинама.
- Симптоми укључују: смањење мисаоне активности, заборавност, поспаност, доношење погрешних одлука, . . .
- Да ли вас то асоцира на „програмера под стресом“?

4

[4] Монт Еверест и околина, извор: Wikipédia francophone, http://fr.wikipedia.org/wiki/Fichier:Panoramique_mont_Everest.jpg.

Модели или хероји?

- Моделе не треба игнорисати.
  - Они интегришу искуство стотина или хиљада професионалаца у области.
- Не треба се ослањати само на моделе.
  - Модели су само помоћно средство (подсетник).
  - Они не могу да замене недостатак експертизе, искуства, посвећености, итд.[5]

[5] Слично као што листе за проверу које пилоти користе пре полетања не служе за учење како се управља авионом.

Зашто модели нису довољни?

- Једна од основних намена модела је да се пажљивим планирањем спрече касније измене.
  - Подсетник са претходног предавања: Непожељни сценарио је да се грешка која је направљена у раној фази развоја открије тек у касној фази. Отклањање оваквих грешака може да услови преправку великог обима (и цене).
- Међутим, у току реализације пројекта може доћи до великих промена у вези са:
  - захтевима,
  - обимом пројекта,
  - технологијама које се користе, итд.

Основна идеја агилног развоја

- Ове промене нису узроковане грешкама у планирању.
- Развојни тим нема увек контролу над променама:
  - потенцијалне промене се не могу предвидети на почетку развоја пројекта,
  - нити се њихов настанак у току реализације пројекта може спречити.
- Основна идеја агилног развоја софтвера:
  - није да се спрече (ионако неизбежне) измене у каснијим фазама развоја,
  - него да се редукује цена тих измена.

Основни принципи

- Функционални кôд:
  - Поузадано мерило тренутног стања развоја (и за програмере и за клијенте).
- Ефикасност посвећеног тимског рада:
  - Људи брже размењују идеје кроз непосредни дијалог него писањем и читањем документације.
  - Резултат заједничког рада више програмера може бити бољи него што би био да су радили појединачно.
  - Комуникацијом између програмера и клијената се могу решити класе проблема које иначе не би могле бити решене (одређивање приоритета, разматрање опција, превазилажење потешкоћа, итд.).

Манифест агилног развоја софтвера[6]

- Откривамо боље начине за развој софтвера, развијајући га сами или помажући другима да то учине.
- Кроз ову праксу, научили смо да вреднујемо:
  - појединце и интеракцију више него процесе и алате,
  - функционални софтвер више него детаљну документацију,
  - сарадњу са клијентима више него преговарање о уговору,
  - реакције на промене више него придржавање плана.
- Тј., иако су ставке наведене на десној страни од значаја, више вреднујемо ставке са леве стране.

[6] http://agilemanifesto.org/iso/sr

Приступи агилном празвоју

- Постоји више приступа агилном развоју:
  - екстремно програмирање,
  - „скрам“ (енгл. *scrum*),
  - адаптивни развој софтвера,
  - развој вођен карактеристикама,
  - „кристал“ (енгл. *crystal*), итд.
- Али сви имају заједничку основу . . .

Заједничка основа (1)

- Кратке итерације (2-6 недеља):
  - Развојни тим редовно (тј., на крају сваке итерације) добија повратне информације од клијената и руководиоца.
  - Током итерације, развојни тим се прилагођава новим информацијама.
- Планирање карактеристика:
  - Планирају се карактеристике софтвера, уместо пројектних задатака.
  - Карактеристике софтвера су оно што клијенти разумеју.
  - Планирање се примарно односи на једну итерацију.
- Динамично одређивање приоритета:
  - На крају сваке итерације, клијент може да промени приоритет карактеристика.

Заједничка основа (2)

- Повратне информације и промене:
  - Редовне повратне информације у вези са техничким одлукама, захтевима клијената и ограничењима.
  - Промене се охрабрују — пошто су већ неизбежне у динамичном окружењу развоја софтвера.
- Тимски рад:
  - Интензивна интеракција између чланова тима је најупечатљивија одлика агилних метода.
  - Тимски рад се односи и на блиско партнерство са клијентима.

Екстремно програмирање (1)

- Двонедељне итерације:
  - Сваке две недеље клијент добија увид у тренутни софтверски систем да би проценио да ли је унапређен у последњој итерацији.
  - Уколико систем није унапређен, клијент усмерава развој у наредне две недеље.
  - Након неколико првих итерација, клијент стиче утисак о брзини развоја и може да одлучи да ли му одговара.
  - Клијент може да заустави развој у сваком тренутку.
- Планирање је краткорочно — односи се на (само) две недеље.
  - Питање: Да ли ће непостојање дугорочног плана довести до хаотичног развоја?
  - Одговор: Очекује се да честе повратне информације од клијента надокнаде непостојање дугорочног плана и елиминишу грешке.

Екстремно програмирање (2)

- Измене у кôду су честе.
  - Питање: Да ли ће сталне промене у кôду довести до тога да он постане лоше структуриран и претежак за одржавање?
  - Одговор: Екстремно програмирање предвиђа контролу квалитета.
- Контрола квалитета:
  - Програмери морају да оставе кôд у најједноставнијој форми која задовољава захтеве дефинисне од стране корисника.
  - Програмери раде у пару (за истим рачунаром) — кôд је резултат комуникације између (најмање) два програмера.
  - Пре него што додају кôд у систем, програмери морају да напишу тест који постојећи кôд не пролази, а кôд развијен у наредној фази треба да прође — корпус тестова расте упоредо са софтвером.
- Свођење кôда на најједноставнију форму се назива рефакторисање.

Резиме

- „Имплементирај и поправи“ — развој вођен кôдом.
- Секвенцијални модел — развој вођен документима.
- Спирални модел — развој вођен ризицима.
- Већина агилних приступа — развој вођен карактеристикама.

„Сребрни метак не постоји“[7]

- Који је приступ најбољи?
- Нема универзалног решења — ниједан није идеалан.
- Различити приступи су прикладни за различите класе проблема.
- Да бисте могли да оцените адекватност примене појединачних приступа, важно је да разумете њихове предности и ограничења.

[7] Brooks, F.P., No Silver Bullet: Essence and Accidents of Software Engineering, *Computer* 20(4), pp. 10-19, April 1987.

Додатна литература

- Bach, J., Enough about process: what we need are heroes, *IEEE Software* 12 (2), pp. 96-98, 1995.
- Gray, L., Gray rebuts Bach: no cowboy programmers!, *Computer* 31 (4), pp. 102 - 103, 105 , 1998.
- Highsmith, J., Cockburn. A., Agile Software Development: The Business of Innovation, *Computer* (September 2001), pp. 120-2, 2001.
- Martin, R.C., eXtreme Programming Development through Dialog, *IEEE Software* (July/August 2000), pp. 12-13, 2000.

Хвала на пажњи

- Питања су добродошла.