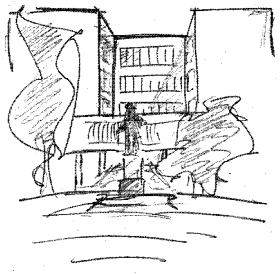
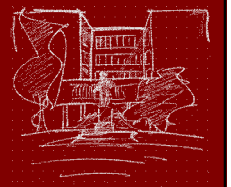
Развој софтвера 9



Саша Малков
Универзитет у Београду
Математички факултет
2023/2024

[P290]
Развој софтвера
Саша Малков



Тема 13

Дебаговање

Појмови

- *Бāгови*
  - све оне непланиране и неочекиване ситне радости које пружа рад на рачунару
- *Прогрāмирање*
  - процес прављења и сакривања багова
- *Дебāговање*
  - трансформисање познатих багова у непознате

Универзитет у Београду - Математички факултет

Пропусти у развоју софтвера

- Развој софтвера је сложен производни процес
- Неминовно долази до прављења пропуста
  - испољавају се на различите начине
- Према фази развоја софтвера у којој настају, пропусти се разликују по сложености, прикривености и начину испољавања

Универзитет у Београду - Математички факултет

Дебаговање / Основни појмови

Пропусти у развоју софтвера (2)

- Пропусти у развоју софтвера се често називају *грешкама* или *бағовима*
- Проблеми са терминима:
  - Често се развој софтвера поистовећује са програмирањем
  - Постојање *грешака* или *бағова* се често неоправдано повезује искључиво са пропустима у програмирању
- *Грешка* у развоју софтвера има много шире значење
  - Обухвата **све** пропусте који настају **у свим** фазама развоја софтвера
  - Термини *пропуст* и *грешка* се често користе у ширем контексту развоја, а термин *баг* у ужем контексту програмирања

Универзитет у Београду – Математички факултет

Дебаговање / Основни појмови

Пропусти у развоју софтвера (3)

- *Пропуст* (грешка, баг) у развоју софтвера је све оно што ствара проблеме у функционисању софтвера као завршног производа
  - „Све оно што има за последицу да се софтвер не понаша у складу са спецификацијом“
  - „Све оно што има за последицу да се софтвер не понаша у складу са спецификацијом или очекивањем корисника“

Универзитет у Београду – Математички факултет

Дебаговање / Основни појмови

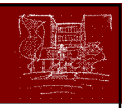
Шире *дефиниције*

- Багови су све врсте проблема који настају при развоју софтвера
- Багови су све оно што производи проблеме
- Багови су све оно што проузрокује непријатности код корисника и аутора

Универзитет у Београду – Математички факултет

Дебаговање / Врсте багова

Врсте багова



- Могу да се класификују на много начина
- Једна од уобичајених класификација је према начину испољавања:
  - неконзистентности у корисничком интерфејсу
  - неиспуњена очекивања
  - слабе перформансе
  - падови система (програм) или оштећења података

Универзитет у Београду – Математички факултет

Неконзистентности у корисничком интерфејсу

- Недоследности у корисничком интерфејсу могу да направе много неугодних последица
  - пример:
    - сви програми за *MS Windows* користе пречицу *Ctrl+F* за тражење
    - *Outlook* је користио ту пречицу за прослеђивање поруке (*forward*)
    - *MS Word* за *OS/2*...
    - процедуре употребе раде за деснoруке а не за леворуке кориснике...
- Најчешћи узроци:
  - Пропусти при пројектовању корисничког интерфејса, или чак при установљавању основних концепата софтвера
  - Пропусти при планирању и спровођењу тестирања

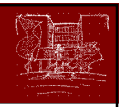
Неиспуњена очекивања

- Добијање неочекиваног (погрешног) резултата је међу најнеугоднијим баговима
  - подврсте:
    - тип 1: неисправан позитиван резултат
    - тип 2: неисправан негативан резултат
    - тип 3: неисправан резултат израчунавања
- Најчешћи узроци:
  - грешке у комуникацији, када развојни тим не разуме исправно потребе клијента
  - грешке у пројектовању, када пројектанти направе пропусте
  - грешке у кодирању, када програмери направе грешке у коду

Слабе перформансе

- Слабе перформансе могу да
  - прилично фрустрирају корисника због сталног или повременог ишчекивања резултата услед слабог одзива система
  - могу да воде потпуној неупотребљивости програма / система
- Најчешћи узроци:
  - грешке у процени оптерећења
  - грешке у процени расположивих ресурса
  - грешке у пројектовању решења
  - грешке у кодирању решења

Падови софтвера и оштећења података



- Најопаснији вид багова
  - могу оставити трајне последице по систем и/или податке
- Најчешћи узроци:
  - грешке у пројектовању начина употребе података
  - грешке у кодирању
  - грешке у повезивању компоненти система

Дебаговање

Управљање пропустима

- Превенција пропуста
- Отклањање грешака - *дебаговање*

Дебаговање / Превенција пропуста

Превенција пропуста

- Суштина је у препознавању околности које имају непосредан или посредан утицај на настајање пропуста и њихово касније отклањање

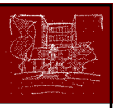
Дебаговање / Превенција пропуста

Превенција пропуста (2)

- Отклањање потенцијалних узрока
  - тј. услова који подстичу настајање грешака
- Стварање услова који помажу да се избегну грешке
- Обезбеђивање услова за лакше проналажење евентуалних грешака

Дебаговање / Превенција пропуста

Оклоности које *подстичу* грешке



- Недовољна стручност развојног тима
  - необученост чланова тима
  - неразумевање захтева
  - изостајање посвећености квалитету
  - приступ “кодирај па размишљај”
  - …
- Непотребно повећан ниво стреса у тиму
  - кратки или чак немогући рокови
  - прековремени рад
  - низак ниво комуникације у тиму
  - …

Околности за избегавање грешака

- Висока стручност тима
  - стално усавршавање чланова тима
  - добра комуникација са клијентом
  - систематичност
- Посвећеност квалитету
  - тестови прихватљивости
  - тестови јединица кода
  - писање робусног софтвера
  - управљање ризицима

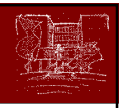
Околности за лоцирање грешака

- Информисаност
  - чување историје верзија програмског кода
  - чување комуникације чланова тима, како међусобне тако и са клијентом
  - редовно писање неопходне документације
- Систематичност и редовност
  - често и планско грађење кода
  - посвећивање пажње упозорењима која се добијају од преводилаца
  - увођење и поштовање правила кодирања и коментарисања

Отклањање грешака

- Уобичајено се назива *дебаговање* (енгл. *debugging*)
- Чине га:
  - уочавање да постоји грешка
  - разумевање грешке
  - лоцирање грешке
  - исправљање грешке

Отклањање грешака (2)



- Обично је најтежи део посла исправно разумевање и тачно лоцирање грешке
- Једном када се грешка лоцира, њено исправљање обично није посебан проблем

Отклањање грешака (3)

- Начин односења према различитим корацима дебаговања обликује три основна приступа дебаговању:
  - “Неформални” метод
  - Емпиријски “научни” метод
  - Хеуристички метод

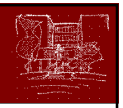
“Неформално” дебаговање

- Једноставан и површан приступ:
  1. Покушати са неком једноставном поправком
  2. Понављати корак 1 док се проблем не реши
- Мотивација
  - неке грешке уочене током писања кода, често могу да се лако отклоне
    - синтаксне грешке
    - пропуштање да се измене вредности неких променљивих
    - погрешне границе блокова кода
    - неисправни индекси
    - и слично...
- Ниска ефикасност
  - овакав приступ “ради” само у једноставним случајевима
  - у сложенијим случајевима често производи нове проблеме

Емпиријски научни метод

- Поступак ***лоцирања проблема*** сличан уобичајеном истраживачком методу у природним наукама:
  - Посматрамо уочен проблем
  - Постављамо хипотезу о узроку проблема
  - На основу хипотезе направимо предвиђање понашања
  - Експериментално проверавамо исправност предвиђања
  - Понављамо претходне кораке, уз поправљање или замењивање хипотезе, све док се не потврди исправност хипотезе или не понестану могућности за њено даље унапређивање

Емпиријски научни метод (2)



- Уопштено посматрано, ово је најбољи приступ дебаговању
- Али није увек јасно шта је следеће потребно да се уради?
  - проблем може да се посматра на много различитих начина
  - могу да се посматрају различити аспекти проблема
  - на које аспекте проблема би требало да се односи хипотеза?
  - како постављати хипотезе?
  - ...
- Постављање хипотезе је препуштено интуицији и искуству
  - помоћ могу да представљају некакве хеуристике

Хеуристичко дебаговање

- Хеуристичко дебаговање почива на примени неке изабране стратегије или неког скупа правила
- Представља надградњу емпиријског метода
- Главни циљеви примене правила су
  - усмеравање посматрања према узроку проблема
  - избегавање прављења превида при посматрању
  - сужавање скупа кандидата за исказивање хипотеза

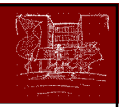
"Хеуристика"

- "Хеуристике" или "хеуристичке технике" су стратегије (или правила) за решавање неког проблема
  - почивају на претходном искуству
  - не представљају оптимално решење
  - често представљају "довољно добро решење"

Хеуристичко дебаговање (2)

- Као добар пример хеуристичког дебаговања размотрићемо систем правила који је формулисао *David Agans*
  - 9 једноставних правила
  - не односе се стриктно на дебаговање програма
    - већ на било коју врсту техничког стваралаштва (инжењерства)
  - *David Agans*, ***Debugging Rules!***, *http://www.debuggingrules.com/*

Хеуристичко дебаговање (3)



- Опрезно!
- Као и сваки други систем хеуристичких правила, ни овај није без недостатака
- При разматрању и употреби правила ваља поступати промишљено
  - наведена правила су углавном једноставна и зато могу да изгледају тривијално и бескорисно
  - насупрот томе, једноставност правила је углавном последица врло прецизне и корисне основне идеје којом се руководи
  - свако од правила може да буде веома корисно у великом броју случаја
  - свако од правила може да буде и потпуно бескорисно у неким другим случајевима
- Правила су, пре свега, упутства за *размишљање* о дебаговању

Тешкоће у отклањању багова

- “Ако проналажење узрока неког бага узме много времена, то је зато што је занемарено неко од основних правила – једанпут када се оно примени решење се обично брзо пронађе .”
  - *David Agans*

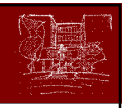
Тешкоће у отклањању багова (2)

- “Особе којима дебаговање иде од руке су обично оне које су (свесно или несвесно) усвојиле и примењују основна правила дебаговања – особе које их нису разумеле обично имају великих проблема при дебаговању.”
  - *David Agans*

Основна правила

1. Разумети систем
2. Навести систем на грешку
3. Најпре посматрати па тек затим размишљати
4. Подели па владај
5. Правити само једну по једну измену
6. Правити и чувати трагове извршавања
7. Проверавати и наизглед тривијалне ствари
8. Затражити туђе мишљење
9. Ако нисмо поправили баг, онда он није поправљен

1. Разумети систем



- “Да би се разумео систем потребно је добро разумети систем”
- Разумевање система
  - није исто што и разумевања проблема
  - већ је разумевање простора у коме постоји проблем
    - програмски модул?
    - апликација?
    - цео рачунарски систем?
  - представља предуслов за разумевање проблема
- Да бисмо у неком систему пронашли грешку и узроке те грешке, морамо да добро познајемо тај систем

1. Разумети систем (2)

- Основни аспекти разумевања система
  - Читање упутстава за систем и компоненте које га чине
    - Али опрезно, упутства могу да буду неисправна!
      - потенцијално проблематична ажурност документације
  - Детаљно читање упутстава
    - не остајати на оквирном упознавању са концептима
  - Разумети шта је очекивано нормално понашање
  - Разумети токове података и активности
  - Познавати алате
  - Обратити пажњу на детаље

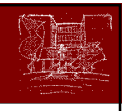
2. Навести систем на грешку

- Грешку је потребно поновити из више разлога:
  - Да би могла да се посматра
  - Да би могла да се посвети пажња узроцима
  - Да би могло да се провери да ли је грешка отклоњена

2. Навести систем на грешку (2)

- Грешка може да се понови на више начина:
  - Поновити поступак
    - поуздан начин да се грешка понови је да поновите поступак пред публиком
    - документовати кораке који воде проблему
  - Почети од почетка
    - некада је непосредан узрок лако поновити, али је припрема окружења за његово понављање компликована
  - Симулирати проблематичну секвенцу
    - ако је тешко или споро мануелно понављати поступак, онда аутоматизовати понављање
  - Не симулирати грешку него услове у којима се испољава
    - симулирање механизама сувише блиских узорку може да заобиђе неисправан део кода
  - Не одбацивати проблем као “немогућ”
  - Чувати направљене алате – могу поново да затребају

2. Навести систем на грешку (3)



- Шта ако се проблем испољава само повремено?
  - нпр. једанут у 5, 10 или 500 понављања
- То значи да нам нису довољно добро познате околности испољавања
- Потребно је:
  - размотрити разне неконтролисане услове:
    - неиницијализоване податке
    - случајно генерисане податке
    - улазне податке
    - везаност за тренутак или трајање извршавања
    - синхронизацију нити
    - спољашње уређаје
  - контролисање услова често води немогућности понављања грешке
    - то значи да један од контролисаних услова у неком специфичном стању (различитом од контролисаног) производи проблем
  - некада такве услове није могуће контролисати, али их је могуће учинити произвољнијим
    - и то може да помогне у разумевању услова настајања проблема
    - али може и да произведе нове грешке

2. Навести систем на грешку (4)

- Шта ако смо све покушали, али проблем се и даље испољава само повремено?
  - Важно је имати на уму да проблем није “својеглав”, већ има врло прецизан узрок, који се сигурно може пронаћи
  - Покушаћемо да испунимо циљеве понављања без самог понављања
- Како?
  - Ако је теже поновити проблем онда је потребно пажљивије посматрати случајеве када се понављање оствари
  - Прикупљати и анализирати информације
    - Уочити услове који су **увек** повезани или **никада** нису повезани са понављањем проблема
  - Да би се (релативно) поуздано установило да је проблем решен, потребно је секвенцу поновити статистички значајан број пута

Универзитет у Београду - Математички факултет

3. Прво посматрај па размишљај (1)

- Размишљање без довољно информација је суштински промашај у приступу
  - може да води искривљеном тумачењу података

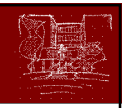
Универзитет у Београду - Математички факултет

3. Прво посматрај па размишљај (2)

- Како?
  - Док се грешка не види, не доносити никакве закључке
  - Посматрати детаље
    - свако посматрање доноси нова сазнања о условима или механизму настајања проблема
  - Правити алате који истичу проблематичне услове или проблематичан део кода
    - разматрати неке видове таквих алата још у фази пројектовања
      - нпр. трајно записивање информација о проблемима
      - омогућити аутоматско разликовање порука ради лакшег претраживања и анализирања
    - додавати такве алате током дебаговања
    - не бежати од привременог замењивања делова кода
  - Користити спољашње алате: софтверски дебагери, хардверски алати...
  - ...

Универзитет у Београду - Математички факултет

3. Прво посматрај па размишљај (3)



- Како?
  - ...
  - Чувати се *ефекта посматрача*
    - “Свако посматрање мења систем зато што су и посматрања саставни део система”
    - Тестирање и мењање кода може да има неугодне последице, од промене понашања па до прикривања грешака
    - Све измене и тестове правити у сасвим малим корацима, како би се смањила могућност прикривања проблема
  - Правити претпоставке само ради бољег фокусирања при тражењу
    - Ако се уведе претпоставка о узроку грешке, не користити је за “отклањање грешке” већ само за фокусирање тражења грешке на неки аспект проблема
    - Изузетак: неке грешке су и честе и лаке за отклањање – само тада може имати смисла покушати са отклањањем пре поузданог потврђивања узрока проблема
      - али и тада само ако постоји поуздано средство за проверу да ли је отклоњен, тј. ако је могуће поновити проблем

Универзитет у Београду - Математички факултет

4. Подели па владај (1)

- Ако је систем сложен (а практично увек јесте)
  - није могуће сагледати све елементе система једнако детаљно
  - потребно је обратити пажњу само на делове који повезују узрок проблама и уочене последице

4. Подели па владај (2)

- Како?
  - Сужавати област тражења узастопним апроксимацијама
    - постављати хипотезе о месту или условима појављивања проблема
    - затим тестовима установљавати тачност хипотезе
    - бирати хипотезе тако да што успешније деле могући простор проблема
  - Пронаћи праве оквире простора за тражење проблема
    - пре сужавања потребно је поставити довољно широке почетне оквире
    - неопходно је поуздано проверити да су почетни оквири довољно широки
    - ово је посебно проблематично ако је оквир потенцијално шири од једног рачунарског система
  - ...

4. Подели па владај (3)

- Како?
  - ...
  - Препознати на којој страни је грешка
    - хипотеза дели простор на два дела
    - мора бити јасно у ком од тих делова је проблем
    - ако није, онда хипотеза није корисна
  - Користити тест-узорке који се лако уочавају
    - често није лако уочити грешке на “живим” подацима
    - потребно је направити вештачке узорке за које је познато како би требало да изгледа резултат
    - пожељно је да узорци буду што мањи и да механизам провере исправности понашања буде што једноставнији
  - ...

4. Подели па владај (4)

- Како?
  - ...
  - Почети од грешке па ићи према узроцима
    - обично је боље тражење започети од места испољавања грешке
    - могућих узрока обично има много и није исплативо проверавати један по један
  - Поправити препознате грешке и наставити тражење
    - багови често иду заједно
    - ако смо пронашли један, то не значи да је и једини
      - обавезно покушати понављање проблема
  - Отклонити шум
    - ако је могуће, потребно је привремено “смирити” делове система који производе висок ниво динамичности, како не би одвлачили пажњу или време
    - један баг може бити узрок другим баговима
    - један начин да се прикрију други багови је привремено искључивање делова кода који нису непосредно везани са местом испољавања посматраног проблема

5. Правити само једну по једну измену (1)

- Ако се начине две измене, тешко је исправно оценити њихов ефекат
  - можда ће једна решити проблем, а друга направити нови?
  - можда је само једна измена довољна?
  - можда ниједна није добра?
  - можда би оне и могле да решавају проблем али нису исправне?

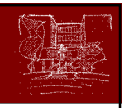
5. Правити само једну по једну измену (2)

- Савети?
  - Изоловати кључни фактор
    - “користити пушку а не сачмару”
    - правити само добро изоловане и локализоване измене
    - то је основни предуслов за исправно закључивање о резултатима измене
    - више измена које нису локализоване у коду значајно отежавају уочавање последица
    - ако је *заиста* уочено у чему је проблем, једна измена је довољна, а ако није, онда је потребно даље посматрати
  - Бити уздржан
    - када се уочи да се нешто дешава, потребно је застати и добро сачекати са реаговањем док се не уоче сви симптоми и не донесе недвосмислен закључак
    - у супротном је могуће да се ради о нагађању и реаговању на основу нагађања
  - …

5. Правити само једну по једну измену (3)

- Савети?
  - …
  - Мењати и испробавати само једну ствар
    - ако покушај поправке или тестирања није помогао, прво је потребно да се програмски код врати у почетно стање, па тек онда да се праве нове измене
  - Поредити са исправним случајем
    - поређење услова при којима настаје проблем и услова при којима нема проблема је један од најкориснијих поступака при разрешавању багова
      - поредити:
      - код
      - трагове извршавања
      - дневнике грешака
      - и све остале расположиве информације
  - …

5. Правити само једну по једну измену (4)



- Савети?
  - …
  - Установити шта је измењено од последње познате ситуације у којој је понашање било исправно
    - некада наизглед небитна измена може да има значајан утицај на систем
    - ако је систем некада радио исправно, прави пут је у
      - локализовању интервала у коме је почело појављивање проблема
      - локализовање измена насталих у том интервалу

6. Правити и чувати трагове извршавања

- Зашто?
  - Некада су наизглед безначајни фактори од пресудног утицаја на исправност рада система
    - пример: службеник за подршку често не може да установи зашто дискете престају да раде после прве употребе док не види како се тачно рукује дискетама

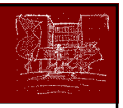
6. Правити и чувати трагове извршавања (2)

- Како?
  - Записивати шта се ради, којим редом, као и шта су уочене последице
    - писани трагови о активностима имају велики значај за касније локализовање догађаја у времену
  - Важно је разумети да сваки од детаља може бити онај који је важан
    - писани траг никада није сувише детаљан
  - Повезивати догађаје
    - међусобним повезивањем симптома и исхода (било успешних или неуспешних) олакшава се уочавање узрока проблема
  - Дневници са траговима извршавања су важни за тестирање
    - на пример, остављање трагова при активирању делова кода или обављању одређених послова, отварању датотека и сл.
  - Записивати, колико год се то чинило тешко
    - “најкраћа оловка је дужа од најдужег памћења”

7. Проверавати и наизглед тривијалне ствари (1)

- “Проверити да ли је уопште укључено”
- Решења проблема су често сасвим једноставна...
- ...али је то често веома тешко уочити

7. Проверавати и наизглед тривијалне ствари (2)



- Савети?
  - Доводити у питање претходно уведене претпоставке
    - ако смо претпоставили да је нека компонента исправна, то не значи да они то и јесте
    - никада није добро бити убеђен да су претпоставке исправне
      - посебно ако су у средишту неког необјашњивог проблема
  - ...

7. Проверавати и наизглед тривијалне ствари (3)

- Савети?
  - ...
  - Почети од почетка
    - да ли су услови за обављање посла испуњени?
    - примери:
      - пита се није испекла – да ли је рерна укључена?
      - аутомобил не може да се покрене – пре него што га раставимо, можда би ваљало проверити да ли има горива?
      - ако не иницијализујемо податке експлицитно, најгоре је што ће систем можда понекад исправно радити
  - ...

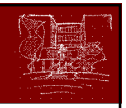
7. Проверавати и наизглед тривијалне ствари (4)

- Савети?
  - ...
  - Проверити алате
    - ако сматрамо да алат нешто ради на неки начин, то не значи да то није потребно и проверити, посебно ако постоји проблем који не разумемо
    - ако алат показује да је све у реду, а ми видимо да није
      - можда алат није исправан
      - или не умемо да га користимо

8. Затражити туђе мишљење (1)

- Не устручавати се да тражите туђу помоћ
- Савети?
  - Укључити свеже снаге
    - много времена проведеног уз неки систем ствара имплицитне (подсвесне) претпоставке које ометају објективно расуђивање
    - неоптерећен небитним детаљима неко “са стране” ће често лакше уочити узрок проблема
  - За (скоро) сваку област постоје експерти
    - уместо да губимо сате (дане? недеље?) много је лакше и јефтиније затражити помоћ експерта
      - експерти знају где је потребно “ударити чекићем”
  - ...

8. Затражити туђе мишљење (2)



- Савети?
  - ...
  - Помоћ је доступна на разним странама
    - сасвим је могуће да је неко већ имао сличан проблем
    - штавише, можда је већ документован
    - потражити “водиче кроз проблеме” (*troubleshooting guides*)
    - потражити дискусионе групе на интернету
  - ...

8. Затражити туђе мишљење (3)

- Савети?
  - ...
  - Не ваља бити (сувише) поносан
    - грешке се дешавају
    - понос треба да помогне да се истраје на решавању проблема
    - али понос не треба да спречава да се потражи помоћ
    - ипак, некада ни други нису у праву – рачунати и са тим
  - Извештавати о симптомима, а не о претпоставкама
    - ако укључујемо некога у решавање проблема, потребно га је снабдети само поузданим информацијама
    - у супротном се праве предуслови да и помагач усвоји наше превиде

Универзитет у Београду - Математички факултет

9. Ако нисмо поправили баг, онда он није поправљен (1)

- Ако симптоми проблема нестану сами од себе, то не значи да је проблем решен – уобичајено је да се појави поново када нам то буде најмање одговарало
- Савети?
  - Проверити да ли је заиста поправљен
    - ако смо пратили ранија правила, онда знамо како да поновимо проблем
    - па, шта се дешава када покушамо да га поновимо?
  - ...

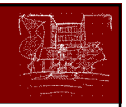
Универзитет у Београду - Математички факултет

9. Ако нисмо поправили баг, онда он није поправљен (2)

- Савети?
  - ...
  - Када помислимо да смо решили проблем, неопходно је да проверимо да ли смо га заиста решили начињеним поправкама или је нешто друго у питању?
    - покушајмо да поновимо проблем без наше исправке
      - ако се тада појављује, а са исправком не, онда смо га решили
      - иначе су у питању измењене околности и сасвим је вероватно да је наше "решење" безначајно
    - наравно, ово некада нема смисла...
  - ...

Универзитет у Београду - Математички факултет

9. Ако нисмо поправили баг, онда он није поправљен (3)



- Савети?
  - ...
  - Проблеми *никада* не нестају сами од себе
    - ако је проблем нестао "сам од себе" то само значи да
      - нисмо довољно добро упознали околности под којима се појављује
      - те непознате околности су се промениле
      - сасвим је вероватно да ће се у неком трнутку поново променити и да ће се проблем поново појавити
  - ...

Универзитет у Београду - Математички факултет

9. Ако нисмо поправили баг, онда он није поправљен (4)

- Савети?
  - ...
  - Поправити узроке проблема
    - ако прегори осигурач, замењивањем осигурача ћемо само привремено решити проблем
    - потребно је пронаћи узроке прегоревања, или ће и нов осигурач прегорети
  - Поправити развојни процес
    - ако је проблем настао, значи да смо направили грешку у процесу
    - грешка у процесу захтева поправке процеса

Универзитет у Београду - Математички факултет

Технике и алати

- Унутрашње технике и алати
  - Све оно што се уграђује у програмски код само ради помоћи у превенцији и отклањању грешака (и тестирању)
- Спољашње технике и алати
  - Различити алати који се употребљавају током развијања или дебаговања програма

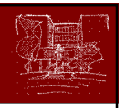
Универзитет у Београду - Математички факултет

Унутрашње технике

- Прављење претпоставки (*assert*)
- Остављање трагова при извршавању (*trace*)
- Интерни описи стања
- Коментарисање значајних одлука и места у коду
- Тестирање јединица кода

Универзитет у Београду - Математички факултет

Прављење претпоставки (*assert*)



- Претпоставке (*asserts*) су делови кода који током извршавања програма проверавају да ли су на датом месту и у датом тренутку испуњене неке претпоставке које морају важити
  - обично прекидају или привремено обустављају извршавање кода са одговарајућим обавештењем

Универзитет у Београду - Математички факултет

Прављење претпоставки (2)

- *assert(...)*
  - већина библиотека има различите облике макроа *assert* који проверава да ли је дати услов задовољен
  - ако није, прекида извршавање програма
  - ако није верзија за дебаговање, онда се провера уопште не укључује у код

Прављење претпоставки (3)

- Пример:

```
#include <cassert>
...
int fact( int n ){
  assert( n<100 );
  int r = 1;
  while( n > 1 )
    r *= (n--);
  return r;
}
```

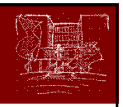
- Претпоставке се проверавају ако није дефинисан макро *NDEBUG*
- Не проверавају се ако је дефинисан:

```
#define NDEBUG
```


Прављење претпоставки (4)

- Библиотека *QT* има следеће елементе:
  Q_ASSERT(bool test)
  Q_ASSERT_X(bool test, const char where, const char* what)*
  Q_CHECK_PTR(ptr)

Прављење претпоставки (5)



- Како?
  - наводити сасвим једноставне услове
    - ако није задовољен неки сложени услов, како ћемо знати који његов део није задовољен?
    - сложене услове је боље поделити на више једноставнијих услова
  - проверавати да ли аргументи потпрограма задовољавају неопходне услове
    - ако не задовољавају, значи да прослеђујемо неисправне вредности
  - проверавати да ли резултат функције задовољава неопходне услове у односу на аргументе
    - ако не задовољава, значи да имамо грешку у имплементацији функције
  - у сложеним потпрограмима проверавати међурезултате

Прављење претпоставки (6)

- C++ од верзије 17 има статичке претпоставке, које се проверавају при превођењу програма:

```
template <class T>
struct data_structure
{
  static_assert(std::is_default_constructible<T>::value,
                "Default-constructible elements required!");
  static_assert(sizeof(T) <= 16,
                "Element size is too large!");
  ...
};
```


Остављање трагова извршавања (*trace*)

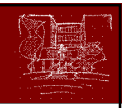
- Хеуристике су већ обухватиле ову технику
- У зависности од начине имплементације дневника, прави се макро или шаблон (или више макроа/шаблона) који исписују текуће стање у дневник
- Пример:

```
void TRACE(const char* s){
  cerr << gettime() << s << ... << endl;
}
```


Интерни описи стања

- Додавање у програмски код делова који праве описе стања, али их не исписују:
  - string state1 = ...;
- Корисно при дебаговању, зато што може да се лако провери стање једне описне променљиве уместо већег броја мањих
- Такав код може да се огради макроима да би се преводио само у верзији за дебаговање

Коментарисање свих значајних одлука и места у коду



- Коментарисање је вид документовања корака који су довели до тога да програм изгледа како изгледа
- Коментари морају да описују:
  - намену неког дела кода
  - мотивацију да решење буде такво какво је
    - посебно важно у случају накнадног мењања
  - начине повезивања са другим деловима кода
    - посебно начин употребе потпрограма
- Коментари не смеју да описују:
  - оно што је очигледно
  - оно што не припада конкретном нивоу апстракције

Тестирање јединица кода

- ... изложено на претходним часовима ...

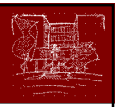
Спољашње технике и алати

- Дебагер
- Алати за праћење верзија програмског кода
- Алати за подршку и праћење комуникације
- Алати за аутоматизовање прављења документације
- ...

Технике употребе дебагера

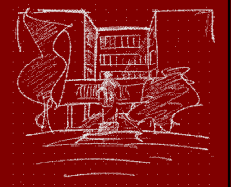
- Извршавање корак по корак
- Постављање тачака прекида
  - у коду
  - на подацима
- Праћење вредности променљивих
- Праћење локалних променљивих
- Праћење стања стека
- Праћење на нивоу инструкција и стања процесора

Литература за тему



- *Andreas Zeller*, ***Why Programs Fail – A Guide to Systematic Debugging***, *Morgan Kaufmann Publishers, 2006.*
- *John Robbins*, ***Debugging Applications***, *Microsoft Press, 2000.*
- *David Agans*, ***Debugging Rules!***, *http://www.debuggingrules.com/*
- Разни други извори...

Хвала на пажњи!



МАТФ
Универзитет у Београду
Математички факултет