Лекция 8 (22. Ноември, 2011)

Връзки
http://en.wikipedia.org/wiki/Bug_tracking_system
http://www.bugzilla.org/
http://en.wikipedia.org/wiki/Bugzilla
http://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems
http://www.espr1t.net/bugs/

Първи час
Неизменно при писането на софтуер се появяват така наречените „бъгове" – тоест грешки на програмистите, несъвместимост на парчета код, несъвместимост между кода и някой драйвер, операционна система или дори друга програма, които водят до неверен или неочакван резултат,карящ програмата да не се държи както е било първоначално предвидено. Интересен парадокс е, че в днешно време този термин обхваща почти изцяло софтуерните аспекти на това защо дадена програма би могла да не функционира правилно – докато в миналото по-често причина за това е бил хардуерът. Една от най-разпространените теории откъде произлиза терминът „bug" (буболечка, на английски) идва всъщност от хардуера. След създаването на механично устройство с някаква логика (механична схема, тоест, представяйте си платка с елементи), група инженери забелязват грешки в неговата функционалност – машината е извършвала непредвидени/непрограмирани действия. След като отворили устройството, забелязали буболечка (хайде, знаете как е в студентски – можете да си представите хлебарка), която е била влязла в кутията и окъсила схемата. Така протичането на ток на неправилното място през тялото на (твърде вероятно споминалата се от напрежението) буболечка, всъщност не е накарало устройството да спре да функционира, а само да работи по неочакван начин. По ирония на съдбата, днешната техника е толкова прецизна, че хардуерен проблем много рядко бива причината за непредвидено поведение на софтуера (освен в Студентски, където топлината от захранването на компютъра често привлича „бъгове").

И докато за проект, който разработваме сами, можем лесно да си записваме какви точно неизправности сме открили, за да можем да ги оправим по-късно (аз лично си ги записвам като коментари в кода), то в една комерсиална среда като софтуерна компания, където има интеракция на повече хора, този подход е силно непрепоръчителен. Потенциална алтернатива би било да се ползва wiki, което програмистите и потребителите биха могли да редактират, като добавят информация за новооткрити проблеми със софтуера при специфични обстоятелства. Това също не е особено scalable (този термин често се ползва за даден продукт, който в момента работи добре с даден брой потребители и би работел добре дори с много по-голям брой потребители). Представете си ако 500 програмиста постоянно добавят и премахват неща от въпросното wiki – това би предизвикало голям хаос, особено пък ако и обикновени потребители бъдат намесени в картинката. Подходът с Wiki страница би бил приемлив за сравнително малък брой програмисти (примерно под 10).

Вместо това мнозинството софтуерни компании ползват специфични софтуерни системи (в случая по-често web-базирани приложения), които да се грижат за лесното и удобно отбелязване, категоризиране и разпределение на откритите бъгове. Този тип софтуер се нарича „система за следене на бъгове" или както по-често бихте го срещнали на английски "bug tracking system" (BTS). Подобно на Version Control системите и маса други приложения, които изискват колаборацията на много хора, и bug tracking системите са най-често web-базирани приложения (или в случая сайтове).

Ако една фирма се занимава с разработката на повече от един проект, не рядко се ползват повече от една bug tracking система за следене на бъговете в отделните проекти. Примерно, ако Microsoft правеха това, те биха имали различни системи за Windows, Office и т.н. Разбира се, все пак най-честата практика е да се ползва единна BTS, където да са бъговете за всичките проекти, разделени вътрешно по някакъв начин.

Основното в една такава система е базата данни на бъговете за даден проект и/или проекти. Почти винаги тя се изгражда от повече от един човек – тоест се поддържа от много хора, които добавят нови бъгове, или премахват стари такива. Често тези хора са програмистите, които работят по проекта, но при големи приложения, които се ползват от хиляди потребители, почти винаги освен на тях се разрешава и на самите user-и да submit-ват bug report-и. Отново често разрешаването на простосмъртните потребители да се месят в работата на програмистите води до хаос и по-скоро пречи на работата, отколкото помага. Затова един от разпространените подходи е пратените от потребителите bug report-и да се преглеждат от хора, чиято работа е да филтрират и форматират по подходящ начин въпросните report-и, като ги добавят в bug tracking системата по начина, по който биха ги добавили програмистите. Това е особено валидно в големи компании като Microsoft, Google, Adobe, Facebook и т.н., където броят потребители и възникнали проблеми е достатъчно голям (и продуктите са достатъчно crappy) за да има и голям брой report-и на ден. Малка вариация на този подход е вместо bug report-ите от user-ите да се подават чрез някакъв вид съобщения (най-често e-mail-и), да се създадат две Bug Tracking системи: една външна (която е достъпна за user-ите) и една вътрешна (в която право да добавят имат единствено хора от фирмата). Така може лесно да се филтрират по-значимите и/или екзотични бъгове от едната система в другата, след евентуално преформатиране за удобство на програмистите. Предимство на този подход е, че програмистите не бива да се притесняват от това, да не издадат фирмени тайни докато описват даден бъг във вътрешната система. Примерно „абе, Dijkstra precompute-а, дето ползваме в .net приложението, което намира най-къси пътища между спирките в София нещо се бъгва ако cache-а на сървъра се претовари от заявки" би издало на външен човек, че а) приложението е на .net, б) ползва се алгоритъм на Дейкстра, в) ползва се преизчисляване на маршрутите, и г) сървърът би могъл да бъде счупен чрез атака с много заявки. Ако въпросното обяснение е видимо единствено за вътрешната BTS това не би било проблем; напротив, даже би могло да улесни комуникацията между програмистите.

Сега нека разгледаме и основните компоненти в самите Bug Tracking системи. Първо, както казахме, имаме потребители. Те могат да бъдат разделени по привилегии: такива, които могат само да виждат бъговете, такива, които могат да добавят, такива, които могат да добавят, изтриват и променят и т.н. Като цяло основно можем да ги разделим на три групи: Guest, Limited и Administrator. Това не е по-различно от произволен форум, wiki или ако щете операционна система, затова няма да му обърнем почти никакво внимание. В общия случай, работейки като програмист вие ще сте или Limited или Administrator.
Вторият (и най-основен) компонент на BTS е базата с бъговете. За всеки бъг се записва кажи-речи всяка информация, която би могла да бъде полезна за неговото отстраняване. Ето и основните „факти", които се пазят:

α) Time - време на събмитване. Това е полезно за разграничаване на бъговете на „стари" и „нови"

β) Severity – колко тежък е бъгът. Както се досещате един бъг може да варира от това даден цвят на екрана да не се display-ва като хората или часовникът на windows-а да се бърка с 1 минута на денонощие, до това програмата да crash-ва брутално и компютърът да се рестартира

γ) Importance – колко важен (належащ) е бъгът. В общия случай бъговете с по-голямо severity имат по-голям importance, но има случаи, в които това не е така. Представете си, че проблемът е доста сериозен, но бива причинен от лош драйвер за даден хардуер (или лош firmware или BIOS). Въпреки, че бъгът има голямо severity, неговият importance е малък, тъй като не зависи от работата на програмистите във фирмата (чака се update от производителя на driver-а, firmware-а или BIOS-а).

δ) Reported by – кой е регистрирал бъга. Полезно за контакт с човека, който го е открил, за евентуално допълнителна информация в какво точно се изразява той и как да се възпроизведе

ε) Description – описание на това в какво точно се изразява бъгът. Трябва да признаете, че има разлика между „Намерих бъг" и „Намерих бъг, който рестартира компютъра като цъкна 666 последователни пъти клавишната комбинация Win+D в Windows" (където Win е специалният Windows-ки клавиш на клавиатурата). Не пробвайте, не вярвам да се рестартира. Освен това сега научихте как да си minimize-нете всички прозорци и да ги възстановите отново.

ζ) How to reproduce – при какви условия е възникнал бъга, или по-точно как програмистите могат да го възпроизведат. Това е ИЗКЛЮЧИТЕЛНО полезно при report на даден бъг, тъй като за да се оправи почти винаги е нужно да се възпроизведе за да се види какво точно се омазва. Ето и какво казва Рандал по въпроса: http://xkcd.com/583/

η) Screenshot – понякога можете да attach-нете картинка, в която показвате бъга. Примерно http://www.espr1t.net/pics/other/facebook.fail1.jpg или http://www.espr1t.net/pics/other/facebook.fail2.jpg или http://www.espr1t.net/pics/other/facebook.fail3.jpg

θ) Друга полезна информация – например операционна система, под която е възпроизведен, браузър (ако става дума за сайт, като в случая с Facebook), характеристики на компютъра (свободна памет, видеокарта (ако примерно е игра)...), коя част от продукта засяга бъгът (примерно UI, kernel, …), коя версия на продукта се ползва (примерно Windows XP, SP3, build 5512), дали еднократно сте го

срещнали или се срещa всеки път; поле за код, където някой от програмистите може да paste-не част от проекта и да каже, че проблемът е в този код, но не е сигурен какво точно в него е проблемното и така нататък – всяка информация, която по някакъв начин би могла да бъде полезна за оправяне на бъга.

Освен тази информация, за всеки бъг се пази и "status" – тоест докъде се е стигнало с неговото оправяне. Статусите най-често се ограничават до:

0) Found – Бъгът е открит, но не е попълнена цялостна информация за него
1) Reported – Бъгът е открит и е попълнена информацията за него. Това е почти еквивалентно на Found (понякога напълно еквивалентно), така че можете да считате, че двете са еднакви.
2) Assigned – Бъгът е бил възложен на програмист, който трябва да го оправи. Това се прави с цел да се избегне случая, в който няколко човека оправят един и същ бъг (тоест вършат едно и също нещо без да знаят).
3) Fixed – Бъгът е оправен „уж". Тоест commit-нат е код, който оправя държанието на продукта, но още не е изтестван дали този код не разваля нещо друго и дали винаги работи.
4) Closed – Бъгът е оправен и продуктът е изтестван, че работи добре с новия код. Работата по него е прекратена и съответно бъгът е „затворен". Друг случай, в който даден бъг може да бъде със статут „closed" е ако ръководителите на проекта са решили, че няма да го оправят (примерно програмистите не могат да го възпроизведат, или пък не виждат смисъл да го оправят, или пък „It's not a bug – it's a feature!").

В общия случай процесът по работа по даден бъг може да се опише чрез следния автомат: 01(23)*4. Тоест той може да бъде found, reported, assigned, fixed, после да се открие, че чупи нещо, отново да бъде преквалифициран като assigned (тоест в процес на „оправяне"), пак на fixed и т.н. докато най-накрая всичко е наред и бъде оприличен като „closed". От друга страна може да бъде found, reported и директно closed ако не представлява интерес за мениджърите на проекта.

Понякога от Bug Tracking системите може да се вадят статистики за продуктивността на програмистите. Един от тези начини е да се следи кой програмист колко бъгове е fix-нал за изминалия месец/година. Това, както сигурно се досещате, не е особено обективна оценка, тъй като има бъгове и бъгове ☺. Докато някои от тях биха отнели 5-10 минути на запознат с проекта програмист, други биха му отнели дни или седмици. Друг начин за оценка чрез бъгове е често срещана по време на стажове в компании като Microsoft, Google и Facebook (поне за тях знам). Често на intern-ите (сиреч стажантите) се дават различни проблеми (bug-ове или improvement-и), които те трябва да реализират. На базата на това как са се справили с бъг с дадена сложност мениджърите на дадения стажант изготвят review (по средата или в края на стажа), в което описват качествата на начинаещия програмист. Примерно ако intern-ът се е справил с един лесен, един среден и един труден бъг, той ще получи „exceeds expectation" – тоест е надминал очакваното от него (може да получи оферта за full time job). Ако пък се е справил с лесния и средния, но не е могъл да направи трудния, най-вероятно оценката му ще е „satisfactory" (и може да бъде интервюиран за full time

или да получи оферта за втори internship). Ако пък е направил само най-лесния (или недай си боже и него не е направил) ще е „failing" (тоест ще получи... сещате се).

Списък с разпространените Bug Tracking системи и техни сравнения можете да намерите в един от линковете, дадени горе. Може би най-разпространената от тях е BugZilla, отчасти защото е безплатна, отчасти защото е лесна за употреба и сравнително по-отдавна се използва (изпреварвайки с близо 5 до 10 години повечето си съперници). Друг разпространен представител е JIRA, с която можете да се сблъскате при някои open-source проекти, тъй като тя е безплатна за тях.

Втори час
Освен да решаваме задачки, говорихме накратко за това как оценяват изпитващите на интервютата за работа. Обещах ви да ви дам основните критерии, както и един blog пост на един стар Google employee, който обяснява как можете да се подготвите за интервюта и какво да очаквате. Има някои интересни неща, така че можете да го прочетете ако имате време (и задължително, ако смятате скоро да кандидатствате някъде за работа).
http://steve-yegge.blogspot.com/2008/03/get-that-job-at-google.html
http://www.espr1t.net/prog101/InterviewTips.pdf