Лекция 15 (10. Януари, 2012)

Links
http://en.wikipedia.org/wiki/Code_review
https://issues.apache.org/jira/browse/HBASE-2823
https://review.cloudera.org//r/426/#review577
http://en.wikipedia.org/wiki/MapReduce
http://en.wikipedia.org/wiki/FLOPS
http://en.wikipedia.org/wiki/Secure_Shell

Първи час

Първо нека разгледаме какво представлява едно Code Review? Това е систематично изследване/разглеждане на source code-а (най-често от друг програмист, което се нарича "peer review", но понякога от група програмисти или специфичен софтуер). Това се прави с цел да се намерят и поправят грешки, пропуснати при първоначалната разработка на софтуера, което води до подобряване както на цялостното качество на кода, така и на възможностите на програмистите. При ревю на кода често можем да научим нови „трикове", библиотеки, похвати, алгоритми и дори просто различен (по-добър) стил на писане. Възможна е и обратна връзка – проверяващият да критикува нашия стил на писане, алгоритми и т.н. като така developer-ът научава нови неща. Code Review-тата биват провеждани по различни начини като например pair programming, informal walkthrough или формални инспекции. Code ревютата често помагат за намиране и премахване на уязвими части от кода (security issues) като например format string exploits - тоест възможност за въвеждане на текст, който би могъл да crash-не или в по-лошия случай даде администраторски привилегии на потребител. Това често се случва при web програмирането, където част от текста, който се въвежда в полетата, после се превръща в част от web-страницата и може да бъде изпълнен. Също така една от много честите грешки в непрофесионални сайтове с бази данни е разрешаване на валиден SQL код в стринговете (http://xkcd.com/327/). Други възможни проблеми, който често биват откривани при code review-та са race conditions, memory leaks, buffer overflows, non-scalable algorithms (logical problems) и други. Когато сорсът бива пазен чрез Revision Control система (Subversion, Mercurial или Git) е възможно кодът да бъде ревизиран от повече от един човек (тъй като е лесно sharable). Освен това, съществуват специфични програми, създадени за review на кода, които ще разгледаме в малко по-големи детайли. Те опростяват допълнително review-тата като ги правят по-бързи и по-ефективни, като запазват комуникацията между developer-а и reviewer-а, и имат възможността да я направят достъпна до повече хора.
Някои автоматизирани code review системи улесняват ревютата на големи парчета код, като предоставят възможност за систематична проверка на кода за известни vulnerabilities (уязвими негови части). Статистиката сочи, че обикновено ревютата са със скорост от около 150 реда код на час. Инскепктирането на повече от няколко стотин линии код на час за критичен софтуер (като например комуникация с user или изпълняване на security checking) би могло да намали рязко ефективността и да обезмисли code review-то. Индустриалните изследвания

сочат, че code review-тата много рядко премахват повече от 85% от дефектите в кода, като средно-статистически този процент е 65%.
Съществуват различни типове code review-та в зависимост от какво точно се цели. Например има голяма разлика дали това е код, който стажант е писал за не особено важна част от проекта, или е финално ревю преди пускане на продукт в експлоатация. Основно можем да разграничим ревютата спрямо това кой е инспектирания обект (код на стажант, програмист, team от програмисти...), в какъв етап от разработката на проекта е даденото review (development фаза (тоест стандартен commit) или крайна фаза (final commit)), по каква част от проекта е инспектирания код (security-critical, performance-critical, или пък non-critical) и други. Сами се досещате, че ако кодът е final commit на екип, който е разработвал security-critical част от проекта ще трябва много по-обстойно и внимателно review, отколкото ако е ежедневен commit на някой junior developer, в non-critical част от кода. В зависимост от нуждите са възникнали няколко начина за review.
Формалното code review е стандартния (тежък) начин на инспекция от кода, в който се извършва внимателно и детайлно разглеждане на кода от много reviewer-и и потенциално на много фази (примерно logic check, security check, performance check, etc.). Този тип инспекции са с идеята да са наистина задълбочени и откриват голяма част от грешките, допуснати в кода, но, естествено, това е на цената на много време и човешки ресурси. Такива ревюта обикновено биват провеждани при предаване на краен продукт (примерно преди official release) или при пренаписването на важна част от кода, както и при интегриране на нов код в проекта (при нов код обикновено има най-много проблеми).
По-леките code review-та обикновено изискват по-малко усилия от формалните такива, но ако се извършват правилно биха могли да доведат до почти същите резултати. Те обикновено са част от нормалния ход на разработка на софтуера. Ако сте junior developer или intern (което най-вероятно ще сте в началото на кариерата си) обикновено ще ви се дава задача, вие ще пишете код, който да я решава, и после вашият project manager или mentor ще прави informal review на кода, който искате да commit-нете преди да имате възможност да го вкарате в проекта. Съществуват няколко вида неформални code review-та (също така наричани „lightweight").

- Over-the-shoulder: след завършване на писането на код, друг разработчик раглежда кода „през рамото" на оригиналния developer, докато той му показва стъпка по стъпка какво прави в кода си и как го е реализирал
- Email pass-around: след завършване и commit-ване на кода, автоматизирана e-mail система разпраща на засегнатите програмисти (примерно тима, който се занимава с тази част от проекта) за да може останалите да се запознаят с неговата функционалност и да разгледат за потенциални проблеми
- Pair programming: Едно нещо, което някои хора нимират (намираме) за ужасно досадно ☺ Но все пак се прилага на някои места: кодът се пише от ДВАМА програмисти на един компютър, като единият пише кода, а другият следи в real-time какво бива написано. Така често ако се изчистват на момента потенциални имплементационни проблеми (излизане извън масиви, лош стил, неясна логика, т.н.), но не винаги се

хващат логически проблеми. Pair programming-а води до код със сравнително по-малко потенциални проблеми, но пък изисква двама програмисти вместо един за написването си (и въпреки това после обикновено е хубаво да се направи review на кода, който двамата са писали)

- Tool-assisted code review: Авторите и ревизорите използват специфичен софтуер за peer code review (това е най-разпространено в днешно време, тъй като съчетава голяма част от предимствата на горните три и елиминира повечето неудобства, създавани от тях).

Разбира се, горните не са взаимно-изключващи се методи и могат да се комбинират за по-цялостно и качествено review. Във всеки случай в една голяма фирма informal code review-тата (независимо по кой от горните начини са направени) са задължителна част от разработката на софтуера (валидно за Google, Microsoft, Facebook, Oracle…).

Както казахме по-рано, съществува специализиран софтуер, който да помага за по-лесни code review-та, с хубаво history и достъпни за целия екип/фирма. Те представляват нещо средно между revision control система и bug tracking система, като са тясно свързани и с двете. След като в bug tracking системата е assign-нат някой бъг на определен програмист, или той просто е написал нова част / модифицирал стара част от проекта и е решил да commit-не новия/променения код в repository-то, той обикновено трябва да мине през някакъв тип ревю (обикновено peer review, тоест ревю от само един ревизор, който пък от своя страна е обикновено менторът или project manager-ът). Така след написване на кода, той го submit-ва за review в дадената система (при някои видове интеграция това става при svn commit). Системата праща информация на ревизора (и добавените viewer-и, които обикновено са останалите хора от екипа) получават мейл (или някакъв друг вид съобщение), че има код, който чака review. Всяко такова ревю получава уникално id (или ticket), което може да бъде видяно от viewer-ите и да бъде оценено/критикувано от reviewer-а. Понякога viewer-ите също могат да дават оценка на кода и предложения. При повечето такива системи новият код бива показан в удобен вид (примерно като web-страница, в който е даден новия код и евентуално разликите със стария, ако има такъв). Във форумо-подобен стил участниците в ревюто (авторът на кода, ревизорът и viewer-ите) могат да пишат коментари, като така дискутират какво трябва да бъде променено и как да бъде променено. Ако промените са козметични, след тяхното оправяне се изпраща ново съобщение до участниците, че кодът е променен и очаква ново review. Ако по време на ревюто се стигне до големи промени (примерно цялостна смяна на алгоритъм или структура данни) след тяхното имплементиране може да се създаде изцяло ново ревю (където няма коментарите и код от предходното за да не настъпва объркване), което следва сходен процес на работа. След изчистване на всички проблеми, reviewer-ът маркира кода като "approved" и разрешава неговия commit в repository-то. Някои от системите за code review-та не правят разлика между reviewer и viewer, тоест те имат сходни права и възможности, като при тях viewer-ите се считат за co-reviewer.

Добавянето на коментари по кода е направено изключително удобно – може да се слагат коментари на специфични редове, което прави комуникацията много изчистена и ясна. Примерно нека имаме следния код:

```
while(!q.empty())
{
	State cur = q.top();
	q.pop();

	if (vis[cur.node])
		continue;
	vis[cur.node] = true;

	if (cur.node == endNode)
		return cur.price;

	for (int i = 0; i < next[cur.node].size(); i++)
	{
		State next;
		next.node = next[cur.node][i].toNode;
		next.price = cur.price + next[cur.node][i].edgePrice;
		next.last = cur.node;
		q.push(next);
	}
}
```

(това е проста имплементация на алгоритъма на Dijkstra за намиране на най-къс път в граф)

Което е пропуснато в нея е да се провери дали новото състояние (state) не е вече посетено преди да се добави в приоритетната опашка. Въпреки всичко, този алгоритъм би работел, макар и по-бавно и с цената на ползване на повече памет, така че лесно би могъл да бъде пропуснато при разработката му а и дори при ползване на Unit тестове. В системите за review на кода можем да маркираме специфичен ред (в случая този с q.push(next)) и да напишем коментар „изпусната е проверка дали даденото състояние не е вече посетено". Така програмистът ще може много лесно да се ориентира какво и по-важно къде го е объркал с не особено пълно описание на проблема от ревизора.

Демонстрация: https://issues.apache.org/jira/browse/HBASE-2823 и https://review.cloudera.org//r/426/#review577

Напълно независимо от горното, реших да спомена и какво представлява SSH, което на пръв поглед звучи хакерски термин и нещо, с което не бихте се занимавали (всъщност го има в „Матрицата" когато Тринити хаква компютъра на една електроцентрала и във филма „Хакери"). Може би е удачно да почна от малко по-ранна ера (въпреки, чеееe като се сетя „Хакери" от коя година е...). Както и да е, навремето компютрите са били едни огромни машини, струващи безбожни пари (стотици хиляди долари) и с изчислителна МОЩ от ... примерно 10 килохерца (300000 пъти по-слабо от един стандартен компютър днес). Както можете да се досетите, хората не са давали толкова пари и не са внасяли една стая апаратура за да може един програмист да седне с една клавиатурка пред

нея и да почне да си коди нещо. Вместо това са се свързвали различни терминали с този компютър, като един терминал е представлявал монитор и клавиатура, от които може един програмист да разработва код (по-точно да извършва някакви изчисления, тъй като тогава писането на код е било малко по-различна история от това, което представлява днес). Колкото и забавно и примитивно да звучи това, доста подобна схема се ползва и днес в редица фирми. Вярно, днес всеки си има компютър или лаптоп, на който може да разработва код, но дори днешните 300 хиляди пъти по-бързи процесори не винаги са достатъчни за извършване на дадени изчисления (примерно някой голям MapReduce) и ни трябва да ползваме някой суперкомпютър (в общия случай клъстър от машини) с обща мощ няколко петафлопа (което пък е няколко стотин хиляди пъти по-мощно от средностатистическия настолен компютър). Или пък не искаме да разработваме кода на работния ни лаптоп, тъй като той може да бъде откраднат, а на някой secure компютър, който и ние не знаем къде е. Или пък на компютър с различна от нашата операционна система. Във всеки случаи има доста случаи, в които бихме искали да изпълняваме команди на различен от нашия компютър, но все пак задавайки ги от него. Тоест отново имаме аналогията с арахичните терминали.
За целта е изобретена командата SSH в Unix (тоест днес основно Linux и Mac OSX), което е съкратено от Secure Shell. Някои от вас сигурно знаят, че Shell е стандартната конзола в Unix-based операционните системи, от която можем да изпълним практически всичко на дадена машина. А е secure, защото позволява обмяната на информация по secure channel между две машини (всъщност SSH е мрежови протокол). И всъщност това е цялата тази „хакерщина" – обикновен протокол, който позволява обмяната на информация между две машини, като обикновено се ползва за да може човек да се „log-не" на remote машина за да изпълнява команди на нея, вместо на своята. Само че виждате конзолата на вашия компютър, вместо да трябва да се разхождате до този, на който изпълнявате операциите. Разбира се, SSH може да бъде ползван за редица други неща, но това е кажи-речи основното, за което бива употребяван. И не е страшен (не хапе (много)).

Втори час
Задачи от интервюта