Доставка на софтуер - особености и процеси

Тема №5

Критерии за успешна доставка

Съдържание

1 Въведение

За да можем успешно да осъществим доставка следва да имаме ясно дефинирани и проверими критерии. Тези критерии следва да се дефинират на възможно най-високо ниво в организацията с цел хомогенности и по-добър контрол.

Критериите имат за цел да служат за като рамка и списък за отмятане, чрез които да може да се провери и гарантира, че доставката се извършва по установения и желан начин и че самия продукт или услуга отговаря на очакваните генерални изисквания и нива на качество.

В тази лекция ще разгледаме различни основни критерии, които са широка разпространени. Важно е да отбележим, че този списък по никакъв начин няма да бъде изчерпателен и различните организации и отдели могат да дефинират допълнителни критерии. В крайна сметка тези критерии са вътрешни за организацията, но в същото време не малко от тях съществуват, за да отразят дадено бизнес изискване, регулаторно изискване или друг фактор оказващ изисквания към софтуера и процесите. И също така целят да гарантират удовлетворяването им.

2 Идеята за корпоративни изисквания/стандарти

Всяка организация иска да е успешна. Един от ключовите фактори за успеха е управлението на рисковете. Друг е намаляване на излишните разходи. Съответно компаниите могат да формулират под формата на система или рамка конкретни мерки и изисквания, изпълнението на които решава определен казус относно успеха на компанията.

В този ред на мисли, компанията може да поеме като вход всички приложими регулации и законови изисквания, контроли относно търсени стандарти, добри практики, генерални клиентски изисквания (приложими отвъд конкретиката на определен продукт), модели на процесите в организацията и други. На база целия този вход, компанията може да определи система от конкретните действия, изисквания, забрани, лимити, препоръки и други, така че да се постигне съответствие с тези входни правила/цели в рамките на всеки екип, отдел, продукт, процес на доставка и т.н.

Идеята им е да са всеобхватни и да регулират и унифицират работата, разработката и доставката на софтуер, правейки ги по-предсказуеми и контролируеми.

При съставянето на по-сложни такива системи е добра практика всички такива изисквания да се категоризират. По този начин се постига по-добра групираност - например изисквания за сигурност, изисквания за достъпност, изисквания за процес на доставка и други.

Изискванията, разбира се, трябва да са документирани ясно и да има хора, отговарящи за тяхната поддръжка. Тези хора биха могли да консултират и да нанасят промени. Всеки служител, който трябва да се съобразява с тези изисквания би следвало да има и достъп до тях. За това те трябва и да са разписани по добър и структуриран начин, така че да бъдат разбрани и приложение от експерти с различен профил и опит. Добра практика е да се използват примери и да се конкретизират критерии за приложимост (например, дадено изискване може да е приложимо само за cloud софтуер).

Ще изброя няколко примерни корпоративни изисквания (само като обобщение, без конкретни детайли):

- Всяка нова версия на продукт трябва да се тества за производителност. Резултатите се сравняват спрямо предходните 5 версии (или ако няма налични толкова, спрямо всички предходни). При по-лоши резултати и ако влошаването надвишава 2%, версията не може да бъде доставена и проблема трябва да се отстрани. Самите тестове трябва да покриват всички основни сценарии и да се изпълняват при равни условия за всяка версия.

- Всеки продукт, който доставяме трябва да бъде класифициран спрямо експортния контрол на ЕС и САЩ. На база това трябва да се определят и приложимите рестрикции относно експорта. Този процес се прави от екипа отговарящ за експортен контрол. Екипа разработващ продукта следва да подаде заявка процеса да се задейства.

- Всеки продукт, който доставяме трябва да бъде доставен чрез утвърден канал за доставка, при условие, че са спазени всички изисквания на канала.

- Последните patch версии на всеки наш продукт не трябва да съдържат уязвимости с рейтинг High (открити преди повече от 30 дни) или Very High (открити преди повече от 7 дни). Тоест трябва в дадените срокове да пуснем версия, в която уязвимостите са отстранени.

- Всяка промяна в кода трябва да има 80% покритие от тестове. Това трябва да е доказано използвайки инструмента X.

- Всеки софтуерен продукт трябва да обработва само и единствено минималното количество лични данни, необходими за функционирането му и удовлетворяването на нуждите от него (това изискване произтича от GDPR).

Важно е да отбележим, че поради факта, че тези изисквания произтичат от фирмата, в определени ситуации може да се правят изключения. За целта трябва да се дефинират процеси за искане и получаване на такива изключения. Изключения лесно могат да се дадат на изисквания като "Софтуерът се разработва само на Java" тъй като те са единствено с цел унификация и неизползваемост. Съответно при истинска нужда от друга технология няма реален проблем това да се позволи. При изисквания произтичащи от регулациите обаче, нещата биха били много по трудни и строги, а понякога и невъзможни.

В остатъка на лекцията ще разгледаме в повече детайли конкретни примери за дейности и изисквания, които са масово разпространени като критерии за успешна доставка. Списъка не е

изчерпателен и в различните организации съществуват множество други критерии, които са не по-малко важни.

3 Управление на софтуерните уязвимости

Във времена, където информационната сигурност е най-голямото предизвикателство и най-критичната цел на софтуерните продукти, няма как да не започнем от управлението на софтуерните уязвимости. Конкретните начини / процеси, чрез които компаниите управляват уязвимостите се различават помежду си, но споделят основните си характеристики и ядрото от дейности.

3.1 Какво е уязвимост?

Уязвимост представлява слабост в изчислителната логика (например, код), открита в софтуерни и хардуерни компоненти, с която може да бъде злоупотребена с цел неоторизиран достъп или извършване на неоторизирани действия върху компютърна система. Тоест експлоатацията ѝ може да доведе до загуба на конфиденциалност, интегритет или наличност. Уязвимости могат да позволят на злонамерени лица да получат директен достъп до система или мрежа, да изпълнят код, да инсталират зловреден софтуер и да достъпят вътрешни системи с цел кражба, унищожение или модификация на чувствителна информация. Ако остане неустановена, тя може да позволи на атакуващия да се представи като суперпотребител или системен администратор с пълен достъп до привилегии.

Справянето с уязвимостите в този контекст обикновено включва промени в кода, но може също така да включва промени в спецификациите или дори премахване на спецификации (например, премахване на засегнати протоколи или функционалности изцяло).

Алтернативно, уязвимостта може да е в следствие на друг софтуер който използваме (3rd party software, open source software). В такъв случай справянето с уязвимостта ще изисква нова версия на компонента, в която уязвимостта е отстранена и замяната на текущата с новата. Но може и да се сведе до промяна в конфигурациите и начина на употреба на компонента.

Известни уязвимости, за които все още не съществува решение се наричат zero-day vulnerabilities.

3.2 Какво е CVE?

Общите уязвимости и изложения (Common Vulnerabilities and Exposures, CVE) представлява база данни с публично разкрита информация за проблеми в областта на информационната сигурност. Номерът на CVE уникално идентифицира една уязвимост от списъка. CVE предоставя удобен и надежден начин за обмен на информация за киберсигурност между производители, предприятия, академици и всички други заинтересовани страни. Предприятията обикновено използват CVE, както и съответните CVSS (Common Vulnerability Scoring System) оценки (ще ги разгледаме след малко), за планиране и приоритизация в своите програми за управление на уязвимостите.

CVE е финансиран от правителството на САЩ, като както Министерството на вътрешната сигурност (Department of Homeland Security, DHS), така и Агенцията за киберсигурност и инфраструктурна сигурност (Cybersecurity and Infrastructure Security Agency, CISA) участват със средства за оперирането му. CVE е общодостъпен и безплатен за всеки, който желае да го използва.

3.3 Какво е NVD?

Националната база данни за уязвимости (National Vulnerability Database, NVD) е база данни, поддържана от Националния институт по стандарти и технологии (NIST). Списъкът CVE подава информация към NVD, така че двете са синхронизирани по всяко време. NVD предоставя допълнителна информация, извън това, което се съдържа в списъка CVE, включително наличие на пачове и оценки (CVSS). NVD предоставя и по-лесен механизъм за търсене по широк спектър от променливи. NVD също е достъпна за свободно използване от всеки.

Въпреки, че са бази спонсорирани от правителството на САЩ, NVD и CVE са дефакто стандарт в каталозите на уязвимости в световен мащаб. Това обаче не изключва и други бази от уязвимости, които би могло да съществуват (и действително такива има, но няма да ги разглеждаме в курса).

3.4 Какво е CVSS?

Системата за оценка на общите уязвимости (CVSS) е метод, използван за предоставяне на качествена мярка за сериозност (severity). CVSS не е мярка за риск. CVSS се състои от три групи метрики: Основни, Времеви и Околна среда. Основните метрики произвеждат оценка в диапазона от 0 до 10, която може да бъде променяна чрез оценяване на Времевите метрики и тези на Околната среда. Оценката на CVSS също се представя като векторен низ, компресирано текстово представяне на стойностите, използвани за извличане на оценката. Така CVSS е подходяща като стандартна система за измерване за индустрии, организации и правителства, които се нуждаят от точни и последователни оценки на сериозността на уязвимостите. Две обичайни приложения на CVSS са изчисляване на сериозността на уязвимостите, открити в системите, и като фактор при определяне на приоритетите за дейности за отстраняване на уязвимости. Националната база данни за уязвимости (NVD) предоставя CVSS оценки за всички публикувани записи в CVE.

NVD поддържа CVSS v2.0 и v3.x. За новопубликувани записи в CVE се използва само версия 3 на CVSS. NVD предоставя "основни оценки" на CVSS, които представляват базовите характеристики на всяка уязвимост. NVD не предоставя "времеви оценки" (метрики, които се променят с течение на времето поради събития външни на уязвимостта) или "оценки на околната среда" (оценки, персонализирани, за да отразят въздействието на уязвимостта върху конкретна организация). За целта обаче има калкулатори, чрез които тези данни могат да се изчислят за конкретна ситуация и да се получи конкретна CVSS оценка, на база на която да се извършат последващи дейности.

Спецификациите на CVSS се притежават и управляват от FIRST.Org, Inc. (FIRST) - неправителствена организация, базирана в САЩ, чиято мисия е да помага на екипите за реагиране на компютърни инциденти със сигурността по целия свят.

През Ноември 2023 е публикувана и версия 4 на CVSS. Но тъй като все още не е придобила индустриална масовост, няма да разглеждаме тази версия за целите на курса.

3.4.1 Основни метрики

Групата от основни метрики представлява характеристики на самата уязвимост. Тези характеристики не се променят с течение на времето и не зависят от реалната възможност за злоупотреба или от компенсационни фактори, които предприятието е въвело, за да предотврати злоупотребата. Независимо от това, което нападател, доставчик или предприятие предприемат, CVSS базовата оценка не се променя.

При търсене на CVSS оценка за уязвимост в система от трета страна, като Националната база данни за уязвимости на NIST, предоставената оценка е почти винаги CVSS базова оценка. Публичните класации за сериозност, като тези в Националната база данни за уязвимости на NIST (NVD), се отнасят изключително до базовите CVSS оценки.

Основните метрики на CVSS имат три подгрупи.

3.4.1.1 Метрики за експлоатация Метриките за експлоатация се отнасят конкретно към уязвимостта, без да се взема предвид конкретна конфигурация или други компенсационни контроли. Метриките за експлоатация включват четири компонента.

3.4.1.1.1 Вектор на атака Векторът на атака е показател за нивото на достъп, необходимо за нападателя, за да експлоатира уязвимостта. Уязвимост, която изисква физически достъп до целевата система, е много по-трудна за експлоатиране от тази, която може да се експлоатира отдалечено през Интернет.

Метриката за Вектор на атака се оценява в едно от четири нива:

- Мрежов (N) - Уязвимости с този рейтинг са отдалечено експлоатируеми, от един или повече хопа, включително отдалечената експлоатация през Интернет.
- Съседен (A) - Уязвимост с този рейтинг изисква мрежова близост за експлоатация. Атаката трябва да бъде извършена от същата физическа или логическа мрежа.
- Локален (L) - Уязвимости с този рейтинг не са експлоатируеми по мрежа. Атакуващият трябва да има достъп до системата локално, отдалечено (чрез протоколи като SSH или RDP) или използвайки социално инженерство или други техники, за да заблуди неподозиращ потребител и го накара да инициира експлоатацията.

- Физически (P) - В този тип атака нападателят трябва физически да взаимодейства с целевата система.

3.4.1.1.2 Сложност на атаката Тази метрика указва условия, извън контрола на нападателя, които трябва да съществуват, за да се експлоатира уязвимостта. Това обикновено се отнася до необходимото потребителско взаимодействие или конфигурации на целевата система.

Метриката за Сложност на атаката се оценява на две нива.

- Ниска (L) - Няма конкретни предусловия за експлоатация.
- Висока (H) - Има условия извън контрола на нападателя за успешна атака. В този тип атака нападателят трябва да изпълни определен брой предварителни стъпки, за да получи достъп. Това може да включва събиране на данни за разузнаване, преодоляване на митигации или ставане на човек по средата (man-in-the-middle).

3.4.1.1.3 Изисквани привилегии Тази метрика точно както звучи, описва нивото на привилегии или достъп, които нападателят трябва да притежава преди успешна експлоатация.

Изискваните привилегии се оценяват в три степени.

- Никакви (N) - Не е необходимо привилегии или специален достъп за извършване на атаката.
- Ниски (L) - Нападателят изисква базови "потребителски" привилегии, за да използва уязвимостта.
- Високи (H) - Изискват се административни или подобни привилегии за успешна атака.

3.4.1.1.4 Потребителско взаимодействие Тази метрика описва дали потребител, различен от нападателя, трябва да направи нещо или да участва в експлоатацията на уязвимостта.

Потребителското взаимодействие е на две нива.

- Никакво (N) - Не е необходимо потребителско взаимодействие.
- Изисква се (R) - Потребителят трябва да извърши определени стъпки, за да успее атаката, например инсталиране на софтуер.

3.4.1.2 Метрики за въздействие Метриките за въздействие измерват въздействието върху основните цели на информационната сигурност (Конфиденциалност, Интегритет, Наличност) на засегнатата система. Иначе казано, какъв е крайният негативен резултат, който настъпва като следствие от експлоатация.

3.4.1.2.1 Конфиденциалност Конфиденциалността се отнася до разкриването на чувствителна информация пред упълномощени и неупълномощени потребители, с цел само упълномощени потребители да имат достъп до целевите данни.

Конфиденциалността има три метрични стойности.

- Никакъв (N) – никакви данни не са достъпни за неупълномощени потребители в резултат на експлоатация.
- Нисък (L) – нападателят има частичен достъп до информацията, без контрол върху това, което конкретно може да достъпа.
- Висок (H) – нападателят има пълен достъп до всички ресурси в засегнатата система, включително изключително чувствителна информация като ключове за криптиране.

3.4.1.2.2 Интегритет Интегритетът се отнася до това дали защитената информация е била нарушена или променена по някакъв начин. Ако няма начин за атакуващия да промени точността или пълнотата на информацията, интегритетът е запазен.

Интегритетът има три метрични стойности.

- Никакъв (N) – няма загуба на интегритет на всякаква информация.
- Нисък (L) – Определено количество информация може да бъде нарушено или променено, но няма сериозно въздействие върху защитената система.
- Висок (H) – нападателят може да променя всякаква/всяка информация върху целевата система, което води до пълна загуба на интегритет.

3.4.1.2.3 Наличност Информацията трябва да бъде достъпна, когато е необходимо. Ако атаката прави информацията недостъпна, например когато системата се срива или след атака с DDoS, наличността е негативно засегната.

Наличността има три метрични стойности.

- Никакъв (N) – няма загуба на наличност.
- Нисък (L) – Наличността може да бъде спорадично ограничена, или производителността може да бъде отрицателно засегната като резултат на успешна атака.
- Висок (H) – Има пълна загуба на наличност на засегнатата система или информация.

3.4.1.3 Обхват Метриката за обхват представлява определение дали уязвимост в една система или компонент може да има последствия върху друга система или компонент. Според FIRST, "Авторитет за сигурността е механизъм (например, приложение, операционна система, фърмуер, среда за изпитване), който дефинира и налага контрол върху достъпа относно това как определени обекти/субекти (например, хора, процеси) могат да имат достъп до определени ограничени обекти/ресурси (например, файлове, CPU, памет) по контролиран начин. Всички субекти и обекти под юрисдикцията на един авторитет за сигурността се считат, че са под един обхват на сигурността. Ако уязвимост в уязвим компонент може да засегне компонент, който е в различен обхват на сигурността от този на уязвимия компонент, настъпва промяна в обхвата."

Обхватът има две възможни стойности.

- Променен (C) – Експлоатирана уязвимост може да има последици върху друга система.
- Непроменен (U) – Експлоатираната уязвимост е ограничена във вредители само до локалния авторитет за сигурност.

3.4.2 Времеви метрики

От друга страна, времевите метрики се променят с течение на времето в резултат на дейности, извършвани както от софтуерни доставчици, така и от хакери. Тези метрики се докладват понякога, но не винаги, в NVD. Ако доставчикът на софтуер е създал пач, и този пач е широко наличен, времевият резултат за тази уязвимост ще бъде по-нисък. От друга страна, ако съществуват известни начини на експлоатация за уязвимост, и те се използват и разпространяват широко, времевият резултат ще бъде по-висок. С разпространението на пачове и код за експлоатация, основните атрибути на Времевите метрики ще се променят, променяйки времевия резултат и общия CVSS резултат.

Времевите метрики измерват текущото състояние на техниките за експлоатация или наличността на код, съществуването на пачове или алтернативни решения или увереността в описанието на уязвимостта. В тази група от метрики има три подметрики.

3.4.2.1 Зрялост на кода за експлоатация Зрялостта на кода за експлоатация отговаря на въпроса "Тази експлоатация случва ли се в действителност?" Много експлоатации са само теоретични по своята същност и никога фактически не биват експлоатирани от противници. Други се експлоатират, но код за оперативна им употреба никога не става широко разпространен, което ги прави неосъществими за неопитни хакери, които представляват мнозинството.

Зрялостта на кода за експлоатация се оценява на един от пет нива.

- Неопределено (X) – няма достатъчно информация, за да се присвои една от другите стойности. Тази стойност не влияе на Времевия резултат.
- Високо (H) – има широка наличност на надежден, лесен за използване и функционален код за експлоатация.
- Функционален (F) – наличен е работещ код, който е поне отчасти надежден.
- Proof-of-concept (P) – кодът съществува, но може да не е надежден и може да изисква много умел атакуващ, за да бъде използван успешно.
- Непроверен (U) – това се отнася, когато експлоатацията е само теоретична и/или не съществува известен код за експлоатация.

3.4.2.2 Ниво на поправка Нивото на поправка се отнася до наличието и зрялостта на поправка или пач за уязвимостта. При по-зряла поправка, Времевият резултат ще намалее.

Нивото на поправка се оценява на едно от пет нива.

- Неопределено (X) – няма достатъчно информация, за да се присвои една от другите стойности. Тази стойност не влияе на Времевия резултат.
- Недостъпно (U) – няма налична митигация или пач за уязвимостта.
- Обходен път (W) – има наличен или неофициален пач, или конфигурация/настройка, която може да ограничи въздействието на уязвимостта.
- Временна поправка (T) – има временен, но създаден от доставчика, пач или оправка.
- Официална поправка (O) – поправка за уязвимостта е налична като постоянен пач или като ъпгрейд от доставчика.

3.4.2.3 Докладна увереност Тази метрика измерва степента на увереност, че уязвимостта действително съществува, както и подробностите на проблема. Например, ако доставчикът публично признава, че съществува уязвимост, има много високо ниво на увереност, че уязвимостта е реална.

Докладната увереност се оценява на едно от четири нива.

- Неопределено (X) – няма достатъчно информация, за да се присвои една от другите стойности. Тази стойност не влияе на Времевия резултат.
- Потвърдено (C) – или доставчикът потвърждава, че уязвимостта съществува, е доказано възпроизвеждането на уязвимостта, или е наличен изходен код, който потвърждава проблема.
- Разумен (R) – Детайлите са публикувани, но уязвимостта не е била независимо проверена.
- Неизвестно (U) – Има съобщения или слухове, че уязвимостта съществува, но има някакъв повод за съмнение в отношението на тези съобщения или уязвимостта не е последователно възпроизводима.

3.4.3 Метрики от Околната среда

Метриките от околната среда се прилагат към конкретната среда, в която съществува уязвимостта. Тези метрики са, по дефиниция, специфични за всяка организация. Тези метрики са всъщност модификатори към Основната група от метрики. Те са предназначени да отчитат аспекти на организацията, които могат да увеличат или намалят общата степен на сериозност на уязвимостта.

Има две основни групи.

3.4.3.1 Модифицирани основни метрики Ако организацията има компенсиращи контроли или мерки за ограничаване, тези усилия са предназначени да намалят възможността или вероятността от експлоатация на уязвимостта. Например сървър, защитен от защитна стена, или неизползван сървър, който е изключен, имат по-ниска вероятност от нарушение в сравнение със сървър, който е публично изложен в Интернет.

Модификацията на основните метрики е толкова проста, колкото изглежда. Организацията прави качествена оценка на въздействието върху всеки от факторите на основните метрики и намалява или увеличава съответния резултат.

3.4.3.2 Изисквания за сигурност Изискванията за сигурност са индикатор за бизнес критичността на актива, измерена в термини на Поверителност, Цялост и Наличност. Ако едно или повече от тези фактори са по-важни от очакваното, предприятието ще увеличи резултата за съответната област.

Изискванията за сигурност се оценяват с едно от четири нива. **За всяко от изискванията имаме отделна метрика!**

1. Неопределено (X) – тази стойност няма влияние върху общият околен резултат и се използва обикновено в ситуации, когато има недостатъчно информация за присвояване на стойност.

2. Високо (H) – Въздействието, предизвикано от загубата на Поверителност, Цялост или Наличност, би било катастрофално за предприятието.

3. Средно (M) – Въздействието, предизвикано от загубата на Поверителност, Цялост или Наличност, би имало сериозен неблагоприятен ефект върху предприятието.

4. Ниско (L) – Въздействието, предизвикано от загубата на Поверителност, Цялост или Наличност, би имало ограничено или изолирано въздействие върху предприятието.

3.5 Откриване на уязвимости

Откриването на уязвимости зависи от типа им и по-точно от къде точно възникват.

Уязвимости, произтичащи от логиката в нашия продукт, могат да се открият чрез следните методи:

- **Преглед на архитектурата и имплементацията:** Този процес включва детайлен преглед на архитектурните решения и кодовата база на продукта. Чрез анализ на кода и структурата на системата могат да бъдат идентифицирани потенциални слабости в логиката.

- **Статично сканиране на кода:** Използването на инструменти за статично сканиране на кода помага за откриване на потенциални проблеми в изходния код още по време на разработката. Това включва проверка за често срещани програмистки грешки и потенциални уязвимости.

- **Тестове за проникване (penetration testing):** Чрез провеждане на тестове за проникване, етични хакери се опитват да симулират дейности на потенциални злонамерени атакуващи. Този процес помага за откриване на слабости в системата чрез активни атаки.

За откриване на уязвимости в използваните компоненти често се използват автоматизирани инструменти, които идентифицират използваните компоненти и техните версии. Тези инструменти търсят информация за уязвимости в базите данни, свързани със съответните компоненти и версии. След анализа се определят засегнатите компоненти и конкретните уязвимости. На базата на тази информация може да се извърши допълнителен анализ и да се определят следващите стъпки за отстраняване на риска.

3.6 План за справяне

В зависимост от CVSS оценката всяка организация ще има своя политика по сроковете за справяне с уязвимостта. Тя може и да се базира на стандарт или регулация. В зависимост от това, следва да се задели необходимия ресурс и да се разработи решение на проблема, след което да се достави поправена версия. Важно е да се отбележи, че само имплементираното решение не решава уязвимостта. Докато няма доставена версия с поправката проблема е се счита за активен.

Относно уязвимости в използваните компоненти следва да се приложат версии с поправки щом такива са налични. Тук се появява един риск. А именно, че организацията разчита на своевременни поправки на тези компоненти. Този проблем го няма при лицензиран / платен софтуер, но при софтуера с отворен код е на лице. За това е важно да използваме утвърден софтуер с отворен код, с история на навременни поправки и активно развитие. Така минимизираме тези рискове.

4 Тестово покритие на кода

Част от намаляването на рисковете за организациите е предотвратяването на грешки и некоректно поведение на софтуера. Никоя компания не желае потребителите ѝ да стават свидетели на грешки в софтуера. Всеки ред код има потенциала да породи грешка. Съответно компаниите често залагат на метриката за тестово покритие на кода (code coverage) и изискват всичките ѝ продукти да имат определено ниво на покритие (и всяка промяна сама по себе си трябва да отговаря на такова изискване).

Това е метрика, която може да ви помогне да разберете колко от вашия изходен код е тестван. Инструментите за покритие на кода използват един или повече критерии, за да определят как вашият код е бил използван или не по време на изпълнението на вашия тестов набор. Общите метрики, които може да видите в вашите доклади за покритие, включват:

- **Покритие на функции:** колко от дефинираните функции са били извиквани.
- **Покритие на оператори:** колко от операторите в програмата са били изпълнени.
- **Покритие на клонове:** колко от клоновете на структурите за управление (например оператори if) са били изпълнени.
- **Покритие на условия:** колко от булевите подизрази са били тествани за стойности true и false.
- **Покритие на линии:** колко от линиите на изходния код са били тествани.

Тези метрики обикновено се представят като броят на фактически тестваните елементи, елементите, намерени във вашия код, и процент на покритие (тествани елементи / намерени елементи). Тези метрики са свързани, но различни.

Няма универсално решение в покритието на кода, и висок процент на покритие все още може да бъде проблематичен, ако критични части от приложението не се тестват, или ако съществуващите тестове не са достатъчно надеждни, за да уловят проблемите предварително. Въпреки това обикновено се приема, че 80% покритие е добра цел. Опитите да се постигне по-високо покритие може да се окаже скъпо, без да произвежда достатъчно ползи.

Unit test-овете се състоят в това да се уверите, че отделните методи на класовете и компонентите, използвани от вашето приложение, функционират. Те обикновено са лесни за изпълнение и бързи за стартиране и ви предоставят общо уверение, че основата на платформата е стабилна. Прост начин да увеличите бързо покритието на вашия код е да започнете с добавянето на unit tests, тъй като, по дефиниция, те трябва да ви помогнат да се уверите, че вашият тестов набор достига всички редове код.

В един момент ще имате толкова много тестове във вашия код, че ще бъде невъзможно да знаете коя част от приложението ви се проверява по време на изпълнението на тестовия ви набор. Ще знаете какво не работи, когато получите червен резултат, но ще бъде трудно за вас да разберете кои елементи са участвали в тестовете. Тук докладите за покритие могат да предоставят конкретно ръководство за вашето екип. Повечето инструменти позволяват да разглеждате докладите за покритие, за да видите реалните елементи, които не са били покрити от тестове, и след това да използвате това, за да идентифицирате критични части на вашето приложение, които все още трябва да бъдат тествани.

Проверките за постигане на търсеното ниво на покритие следва да се осъществят по автоматизиран начин в CI процесите.

Постигането на отлично покритие е отлична цел, но тя трябва да бъде съчетана с наличието на стабилен тестов набор, който може да гарантира, че отделните класове не се счупват, както и да потвърди цялостта на системата.

5 Управление на софтуерните лицензи (на използвания софтуер)

Софтуерът с отворен код е неотделима част от разработката на продукти за организации от всички размери и форми. Докато инженерните екипи преди предимно се доверяват на собствен код, отвореният код в момента обхваща повече от 90% от компонентите на модерните приложения.

С разрастването на софтуера с отворен код, управлението на съответствието с правните и интелектуалните изисквания става от съществено значение, което означава, че лицензите за софтуер с отворен код също придобиват сериозна важност.

Лицензите уведомяват потенциалните ползватели за условията, под които могат или не могат да използват проект с отворен код. В крайна сметка, лицензите гарантират, че разработчиците на софтуер с отворен код - общността зад много от най-иновативните технологии днес - могат да защитават своите творения по свой вкус. Лицензите за отворен код служат като правно споразумение между автора на отворения код и потребителя: авторите предоставят софтуер с отворен код безплатно, но с определени изисквания, които потребителят трябва да спазва.

Обикновено условията на лицензa за отворен код влизат в сила при разпространение на софтуера - ако използвате компонент с отворен код само за вътрешен инструмент, например, вероятно няма да бъдете обвързани с изисквания, които обикновено биха възникнали.

Разбирането на лицензите за софтуер с отворен код е изключително важно както за авторите, така и за потребителите на отворен код. Ако използвате компонент с отворен код, вие сте правно

отговорни за спазването на условията на неговия лиценз. Ако сте автор на софтуер с отворен код, изборът на грешен лиценз може да позволи на компании да използват вашия проект по начин, с който не сте съгласни.

5.1 Видове лицензи за софтуер с отворен код

Съществуват два основни типа лицензи за софтуер с отворен код: лицензи с минимални ограничения и "copyleft" лицензи. Лицензите с минимални ограничения обикновено имат по-малко ограничения за използване на лицензирания код, отколкото "copyleft" лицензите.

5.1.1 Лицензи с минимални ограничения (Permissive Licenses)

Лицензи с минимални ограничения обикновено позволяват използването на лицензирания код с малко ограничения. Потребителите могат да вземат софтуера с минимални ограничения, направят го собствен с промени или добавки и да разпространяват този променен софтуер със само няколко условия.

Например, най-често използваният лиценз с минимални ограничения (лицензът MIT) изисква само разпространените програми, които използват лицензирания код, да включват известие за авторските права и копие на текста на лиценза. Друг популярен лиценз с минимални ограничения, лицензът BSD 3-Clause, е подобен на лиценза MIT, но с допълнително условие, което забранява използването на името на автора или неговите сътрудници (на кода с лиценз BSD 3) за маркетинг на разработвания софтуер, включващ компонента с отворен код, без явно писмено разрешение.

Лицензът Apache 2.0 също е подобна на лицензията MIT, но изисква разпространените програми, използващи лицензирания код, *а)* да упоменават всички основни промени в оригинала и *б)* да включват копие на файла NOTICE с бележки за признание (ако оригиналната библиотека го има), заедно с изискванията за авторските права и текст на лиценза. Той включва и явно предоставяне на патентни права и защитна клауза за прекратяване. За да бъде ясно, Apache 2.0 не изисква разкриване на изходния код, само "за всички модифицирани файлове **трябва** да има ясно и видимо известие, че въпросните файлове са променени".

Така че, въпреки че различните лицензи с минимални ограничения налагат леко различни условия за използване на лицензирания код, софтуер с отворен код под този вид лицензи може обикновено да бъде променян, копиран, добавян, изваждан и разпространяван с относително малко ограничения. Това ги прави перфектни за включване в продукти със затворен код, които планираме да комерсиализираме.

5.1.2 Copyleft лицензи (Copyleft Licenses)

За разлика от лицензите с минимални ограничения, copyleft лицензите обикновено изискват, че всяко производно произведение на софтуера с такъв лиценз трябва да бъде пуснато под същия лиценз като оригинала. И с други думи, променениият код трябва да бъде точно толкова "отворен", колкото и оригиналът.

Този вид лицинзи биват два вида: силни и слаби. Силните изискват всеки, който разпространява производни произведения от лицензирания код, да направи наличен съответния изходен код под същия лиценз. Тези лицензи се прилагат към целия софтуерен продукт, включително свързаните библиотеки и други компоненти. GPL v2, GPL v3 и AGPL са примери за силни копие лицензи. Те са силно неподходящи за употреба в софтуер със затворен код, който смятаме да дистрибутираме (on premise, mobile).

Слабите "copyleft" лицензи (като LGPL и Mozilla Public License 2.0) имат подобни изисквания, но те се отнасят до по-тесен набор от код. Ако потребител модифицира и разпространява код, покрит от такъв лиценз, той трябва да пусне модифицираната версия под същия лиценз като оригинала. Обаче, ако програмата просто използва компонент с такъв лиценз (и го държи в отделен файл), тогава тя може да го комбинира с допълнителен и/или променен код и да разпространява този по-голям продукт под различен лиценз.

5.2 Откриване на използваните лицензи

За компаниите е от огромно значение да следят от какъв софтуер е съставен продукта им по множество причини. Една от тях е именно, за да предотвратят нарушения в лицензите на използваните компоненти. Исторически това се е правело с ръчни проверки и поддържане на списъци. Като

всяка друга подобна дейност, това подлежи на човешки грешки и консумира твърде много ценно време. За това, в днешно време тази дейност се извършва по автоматизиран начин. Подобно на анализите за уязвимости във внедрените компоненти (често инструментите поддържат и двата типа анализ), автоматизирано могат да се открият използваните компоненти и техните версии, на база на това да се идентифицира лицензa (или лицензите, тъй като двойното лицензиране е възможно - при него консуматора избира спрямо кой лицензa да ползва софтуера). По този начин може да се направи необходимия анализ и да се идентифицира има ли наличие на проблемни лицензи, които не позволяват софтуера да се разпространи и комерсиализира по желания начин.

5.3 Действия при откриване на нежелани лицензи

Отстраняването на несъответствията между желания начин на доставка и комерсиализация от една страна и изискванията на всички лицензи на внедрения софтуер от друга старана е критично за успеха на доставката. В противен случай производителя на продукта вероятно ще се сблъска със съдебни дела. Тези процеси водят до финансови загуби, доста загубено време и загуба на репутация.

За това след откриването на несъответствие е важна да се предприемат следните действия:

1. Инструментите не са перфектни. Те могат да открият грешен компонент или да идентифицират погрешка лицензa (особено ако има двойно лицензиране може да идентифицират само едната опция). За това като първа стъпка следва да се идентифицира коректността на откритото несъответствие. Ако се окаже некоректно откритието и в действителност няма проблем, нямаме нужда от повече стъпки.

2. Ако се окаже коректно откритието следва да идентифицираме естеството на употребата. Дали е зависимост на друг внедрен софтуер? Или пък нашия код директно взаимодейства с проблемния софтуер? Това са много важни въпроси, които изясняйки ще можем да направим заключение как би изглеждало прекратяването на употребата ни на проблемния компонент. Ако се окаже зависимост на друг компонент, който използваме и той ще трябва да се превърне в кандидат за премахване освен ако тази зависимост не може да бъде нарушена (например самия компонент, който има проблемна зависимост позволява да се пренастрои, за да използва алтернатива). За да предприемем тези стъпки трябва ясно да знаем точно каква функционалност удовлетворяват компонентите, които трябва да премахнем.

3. Следва нуждата да намерим алтернативи на компонентите, които ще премахнем. Трябва да намерим компоненти, които предоставят функционалността, която премахваме (само тази, която използваме от премахнатия компонент, а не целия му набор възможности). Тези компоненти трябва да бъдат с подходящ лицензи (все пак не искаме веднага да повторим упражнението) и да предоставят тази функционалност с необходимото качество. Потенциално може и да не открием подходящ заместител. Може и да решим, че функционалността е твърде тривиална и не си заслужава рисковете, свързани с използването на допълнителен външен софтуер.

4. Без значение дали сме се спрели на алтернативен компонент или сме решили да си имплементираме сами функционалността, тази дейност трябва да се извърши и да се изпълнят необходимите валидации върху променения код (проверки за уязвимост, тестове и т.н.), включително и проверката за лицензи, която да потвърди и да ни даде формално доказателство, че вече няма лицензионни проблеми.

5.4 Използване на комерсиален софтуер

До сега се бяхме фокусирали върху софтуера с отворен код. Именно, защото нарушения при него е по-лесно да бъдат допуснати поради лесното сдобиване и начало на употреба на този тип софтуер.

Често обаче в компонентите, изграждащи един продукт се включват и външни компоненти, които са комерсиализирани и са със затворен код (и софтуера с отворен код може да бъде комерсиализиран, обикновено под формата на консултантски услуги и поддръжка, но няма да задълбаваме в този аспект). Тоест работят на принципа на закупуване на право на ползване. Обикновено този софтуер идва със свой специфичен лицензи, в който отново се уреждат правата за използване. Те могат да бъдат много разнообразни и е важно да са добре разбрани (може би с помощта на юристи), тъй като ще липсват публично достъпно обобщение и инструменти за идентификация на потенциални

проблеми. Тъй като зад този софтуер вероятно стои голяма компания със сериозни ресурси и тя знае за вашата употреба тъй като сте закупили компонента от нея (придобиването на софтуер с отворен код обикновено е анонимна транзакция) е много по-вероятно да бъде проверена употребата и да се стигне до дело при нарушения.

6 Качество на кода

Кодът на софтуерните продукти и услуги, които се разработват е както основния източник на стойност за клиентите, така и основния източник за проблеми и неблагоприятни въздействия. Също така всяко доставяне на допълнителна (последваща) стойност ще изисква изменение и разбиране на този код. Същото се отнася и до предоставяне на поддръжка (често поддръжката е по-доходоносна от самия софтуер). За това е важно и кода да отговаря на определени изисквания, свързани с неговото качество, така че да може лесно да бъде разбиран и изменян. Също така качествения код е такъв, който е малко-вероятен да породи грешки и неконсистентно поведения, който менажира правилно ресурси и т.н.

Метрики за качество на кода има много. Наличието на уязвимости в кода е такава метрика. Ние вече говорихме за това в конкретната ни секция за уязвимостите. Някои могат да кажат, че и тестовото покритие е такава метрика. Това е спорно и докато има аргументи в подкрепа на такава теза, за целите на курса ще го разглеждаме като нещо отделно (за това и вече го разгледахме в отделна секция).

Нека обърнем внимание на две метрики, които са от значение за качеството на кода (обикновено метриките за качеството на кода са много тясно свързани с нефункционални изисквания). Разбира се, има и много други количествени и качествени метрики, които компаниите могат да решат да измерват и да налагат изисквания за тях.

6.1 Цикломатична сложност

Цикломатичната сложност е софтуерна метрика, която количествено измерва сложността на програмен код. Тя помага на разработчиците да изградят разбиране за сложността на своите програми, позволявайки им да вземат решения относно промените в кода и поддръжката му. С подкрепата на това по-дълбоко разбиране за структурата на програмата си, програмистите могат да измерват сложността на кода с по-голяма точност и да разбират последиците, които тази сложност може да има върху поддръжката, надеждността и общото качество на софтуерен продукт.

6.1.1 Изчисляване

Нека разберем как се изчислява цикломатичната сложност. Изчислява се броя на линейно независимите пътища през изходния код на програмата, което съответства на броя на точките за вземане на решения в кода, плюс единица. Същността на тази формула е да оцени колко разклонения се случват в програма, тъй като всяко разклонение въвежда допълнителни потенциални пътища за изпълнение.

За да се приложи тази формула ефективно в тестването на софтуера, е от съществено значение първоначално да се представи изходния код като граф на контролния поток (CFG). CFG е насочен граф, където всеки връх представлява основен блок или последователност от не-разклоняващи се оператори, а ребрата означават контролния поток между тези блокове. След като е създаден CFG за изходения код, може да се започне изчисляването на цикломатичната сложност, използвайки някои от трите метода, изброени малко по-надолу.

Ключовите компоненти на CFG на програма включват:

- **Върхове:** Единични команди или оператори.
- **Ребра:** Свързват върхове.
- **Свързани компоненти:** Изолирани секции от графа.

При изчисляването на цикломатичната сложност може да използвате три различни подхода:

1. **Основна формула за цикломатична сложност:** $\textit{Цикломатична сложност} = E - N + 2P$, където E е броят на ребрата, N е броят на върховете, а P е броят на свързаните компоненти.

2. **Броене на затворените региони на графа:** *Цикломатична сложност = Брой затворени региони + 1.*

3. **Сумиране на предикатните върхове (върховете, които са взимане на решение):** *Цикломатична сложност = Сума на всички предикатни върхове + 1.*

С всеки от тези подходи ще се получи цяло число, което представлява броят на уникалните пътища през кода. Този брой указва не само колко трудно може да бъде разбран от разработчиците, но и влияе на способността на тестовете да гарантират оптимална производителност на програмата или системата. По-високите стойности предполагат по-голяма сложност и намалена разбираемост, докато по-ниски числа създават очакване за по-проста и лесна за следване структура.

6.1.2 Значение на метриката

Една от основните ползи от цикломатичната сложност е нейната способност да разкрие нивото на усилие, необходимо за тестване и отстраняване на грешки в програма. Количественото измерване на броя на възможните пътища в даден сегмент код с помощта на цикломатичната сложност позволява на разработчиците да определят приоритети в области, които представляват по-голям риск или трудност по време на разработка. По този начин те могат да разпределят ресурсите по време на процеса на тестване по-ефективно, гарантирайки, че критичните компоненти на проекта се изследват внимателно, докато по-малко сложните елементи все още получават достатъчно внимание.

Друго предимство, което следва от използването на цикломатична сложност, е нейният потенциал да подобри общото качество на софтуера, като идентифицира секции от код с високи стойности на сложност. Високата цикломатична сложност често указва на заплетена логика, излишни разклонения или прекомерно сложни влагани структури, които могат да доведат до намалена четливост и поддръжка. Такива сегменти от код също са по-склонни към дефекти поради тяхната вътрешна сложност. Разработчиците могат да се фокусират върху тези области за преформатиране или опростяване, за да подобрят яснотата и последователността на софтуера, като минимизират потенциалните затруднения.

6.2 Когнитивна сложност

Когнитивната сложност се отнася до нивото на умствено усилие, необходимо на разработчика да разбере и работи с конкретен част от кода, влияещо от фактори като поток на управление, поток на данни, сложност на функции/методи, зависимости, именуване, документация и общ размер на кода.

Когнитивната сложност настоява за създаването на софтуер, който е опростен, еднозначен и ориентиран към човека. Използването на тази теория позволява на софтуерните разработчици да пишат и структурират своя код по начин, който подобрява четливостта и разбираемостта му и намалява когнитивния товар, който може да наложи. Намалената когнитивна сложност стимулира високото разбиране и по-лесната поддръжка, водейки до ефективно отстраняване на грешки и по-малко грешки.

Когнитивната сложност се фокусира върху разбираемостта на кода, идентифицирайки трудностите, с които хората могат да се сблъскат при интерпретиране на кодовата база.

Факторите, които влияят на когнитивната сложност, включват управлението на алгоритмичната сложност, анализа на кода, създаването на оптимални нива на абстракция на кода и подобряването на структурата на кода.

- Алгоритмичната сложност измерва изчислителните ресурси, необходими за изпълнение на алгоритъм.

- Абстракция на кода е идеята да скрием сложните детайли зад прости интерфейси, като направим кода по-лесен за разбиране и поддържане.

- Структурата на кода трябва да бъде интуитивна и лесна за навигация, за да се намали когнитивният товар за разработчиците и направи софтуера по-лесен за разбиране, отстраняване на грешки и поддръжка.

Намаляването на когнитивната сложност може окончателно да подобри ефективността и продуктивността на процеса на софтуерната разработка.

7 Заключение

Компаниите трябва да балансират между голям обем доставки на чести интервали и подсигуряването на съответствие, минимизиране на рисковете и поддържането на високо качество. За това е критично да съставят гъвкави механизми, с максимално ниво на автоматизация, на база които да се придобива увереност, че продукта или негова версия не нарушават някоя от множеството цели и съображения пред организацията.

За това е важно съставянето на система изисквания на ниво компания, с ясни предписания за удовлетворяването им, допринасяйки значително за бърз, ефективен и изчистен от грешки цикъл на доставка.