Tesztautomatizálás alapjai

Utoljára módosítva: 2020. május 14.

Készítette: Ráncsik Áron

Tartalomjegyzék

1. TTCN

2. Tesztelés minőségbiztosítás és adatbiztonság szempontból

2.1. Fogalmak

Etikus hacker

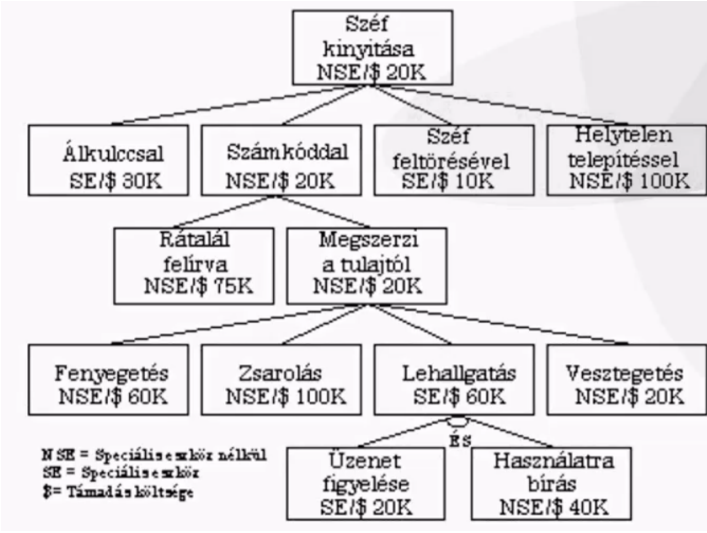
- A vizsgálandó cég által megbízva
- Aláírt szerződés, engedély, titoktartási nyilatkozat és károk alól felmentés
- Az előzetes engedély nélküli hibakeresés és a hiba kihasználása Magyarországon bűncselekmény (de legalábbis szürke zóna)
- Vannak azonban "jó fej" cégek akik díjazzák a hibabejelentést, és "bounty" programokat működtetnek.
- Alapvetően nem szabad erre számítani.

Kockázat

- Hatás (veszteség)
- Valószínűség
- Védekezés
  - Költség
- Támadó
  - Költség
  - Kockázat
  - Nyereség

Kockázatelemzés egy módszere a támadás-fa elemzés

- A támadó szempontjából közelíted meg magad



Vagyon

- Adatok
  - Ügyféladatok
  - Üzleti titkok
  - Forráskód
  - Dokumentáció
  - e-mailek
  - Alkalmazottak adatai
- Eszköz: prototípus
- Egyéb: hírnév bizalom

Vagyon értéke

- Jövőbeni jövedelem
- Érték a konkurenciának
- Újragyártás költsége
- Büntetés a vagyon hiánya miatt (pl. adóellenőrzés)
- Helyreállítási költség adatvesztés miatt
- Büntetés adatvesztés miatt
- Cég értékének csökkenése

2.2. Adatokhoz való hozzáférés és biztonság

- Hozzáférés
  - Helyi hálózat
  - VPN
  - Fizikai (CD, Pendrive)
  - Email
- Biztonság
  - Titkosítás (milyen erősség, ki ismeri a kulcsokat)
  - Azonosítás (jelszó, hw kulcs, kettős autentikáció)
  - Jogosultságok

2.3. Kockázatok

- Vezetőség támogatásának hiánya
- Források hiánya
  - Tudás hiánya
  - Eszközök hiánya
- Szervezet támogatásának hiánya
- Kulcsemberek támogatásának hiánya
- Biztonsági kockázatok lebecsülése
- A biztonsági tesztelés és a tesztelési előírások eltérése
- A rendszer működésének nem ismerete
- A rendszer céljának nem ismerete

2.4. A biztonsági tesztelés szerepe a software életciklusban

Lépései:

- Követelményfelmérés
- Tervezés
- Fejlesztés
- Tesztelés
- Üzemeltetés
- Dokumentáció

2.5. Tesztelés követelményfelmérése

Adatvédelmi igények

- felhasználói csoportokat és azok adatvédelmi igényeit azonosítani és dokumentálni
- adattípusokat azononosítani és a biztonsági szinteket hozzárendelni
- felhasználók hozzáférését meghatározni

Biztonsági követelményeknek való megfelelés

- Biztonsági követelményeket dokumentálni
- Kivételeket felkutatni

Gyakori sérülékenységek

- Az ismert sérülékenységeket dokumentálni és figyelembe venni
- Ismeretlen sérülékenységekre felkészülni

Tesztelhetőség

- A követelmények úgy lettek megfogalmazva, hogy az alapján megírhatóak a tesztek?
- Az általános kifejezések (pl. biztonságos adatátvitel vagy hozzáférés csak megfelelő jogokkal) megfelelően definiálva vannak és tesztelhetőek?

Használhatóság

- Egyensúly a használhatóság és a biztonság között
- A követelmények megfelelően szabályozzák a használhatósághoz tartozó biztonsági szabályokat?
- A biztonsági előírások világosak és érthetőek?
- A felhasználó hozzáférési nehézsége esetén követendő eljárások definiálva és dokumentálva vannak?

Teljesítmény

- Egyensúly a teljesítmény és a biztonság között
- A követelmények megfelelően szabályozzák a teljesítményhez tartozó biztonsági szabályokat?

2.6. Tesztelés tervezése

- Adj vagy kapj bizalmat, de soha ne feltételezz
- Használj biztonságos autentikációt
- Autentikálás után vizsgáld a hozzáférési jogokat
- Tartsd külön az adatokat és az utasításokat (sql injection, buffer overflow)
- Ne hajts végre megbízhatatlan forrásból származó utasítást
- Validáld az adatokat
- Használd helyesen a titkosítást
- Figyelj oda a kényes adatokra
- Legyél tisztában a külső komponensek sebezhetőségeivel
- Mindig vedd figyelembe a felhasználóidat
- Funkcionális biztonsági ellenőrzések(pl. pénztáros nem adhat ki egy határnál nagyobb összeget csak a főpénztáros engedélyével)
    - Jelszavak kezelése(pl. nem szöveges formában vannak tárolva)
    - Jelszavak feltörés ellen védettek(pl. nem lehet korlátlanul próbálkozni)
- Szerkezeti hozzáférés ellenőrzés
    - felhasználói jogok
    - titkosítás
    - autentikáció
- Biztonságos kódolási szokások
    - HTTP GET kérések nem tartalmazhatnak érzékeny adatokat
    - Alkalmazás hibákat az alkalmazás kell hogy lekezelje
    - Adatvalidáció és a hibák logolása (SQL injection)
    - Adatok ideiglenes tárolása is csak biztonságos formában és csak a szükséges ideig
    - Szervizek futtatása csak a minimálisan szükséges jogokkal és sohasem root-ként
    - API-k használata direk OS hívások helyett
    - Adatok útközbeni titkosítása
    - Jövőbiztosság (könnyű tesztelni és karbantartani)
- Hozzáférés az operációs rendszerhez

  Ha egy támadó behatolt akkor semmiben sem lehet bízni
- Nyelv sebezhetőségek

  forráskód fertőzés

  fordító hiba
- OS sebezhetőségek
- Külső fenyegetések

  DoS
- Belső fenyegetések

  Outsourcing

  Elégedetlen alkalmazott
- A tesztelő rendszer sebezhetőségei

2.7. Tesztek fejlesztése

- Buffer overflow
- Adatvalidálás
- Fejlesztő által beépített nem kívánatos kód
- Backdoor
    - tesztelési célból
    - támadási célból
    - felsőbb utasításra

2.8. Komponens fejlesztés tesztelése

- Egységként kezeld (kívül minden veszélyforrás)
- Bemenő adatok validálása
- Fordító figyelmeztetéseket nem szabad figyelmen kívül hagyni
- Kövesd a biztonsági előírásokat
- Ne bonyolítsd túl
- Alapállapot a tiltás, csak szükség esetén adj engedélyt
- A lehető legkevesebb jogot adj
- Csak a feltétlen szükséges adatokat küld tovább
- A minőségbiztosításnak van értelme
- Code review

2.9. Integráció tesztelése

- Az integráció során új hibalehetőségek és biztonsági rések keletkezhetnek
- Ugyanakkor a komponensekben megmaradt biztonsági rések el is tömődhetnek hogy aztán később orvul hátbatámadják a figyelemetlen fejlesztőt
- Smoke testing (olyan teszt gyűjtemény ami jól lefut a rendszeren és módosítás esetén nézzük hogy elfüstöl-e?)

2.10. Rendszer- és elfogadási teszt

- Az átvételi feltételeket még a tervezési fázisban kell definiálni
- Az átvételi teszt valós üzemi feltételek között történik
- Helyes és helytelen adatokat és felhasználói viselkedést, ill. a rendszer azokra adott válaszát is szükséges tesztelni
- Tervezett tesztek meglétének ellenőrzése és futtatása, valamint az eredmények értelmezése

2.11. Üzemeltetés ellenőrzése

- Rendszeres penetrációs tesztelés
- "Bounty program"
- Definiált és dokumentált hibakezelési eljárások
- Kárminimalizálás és kommunikáció
- Rendszeres adatmentés

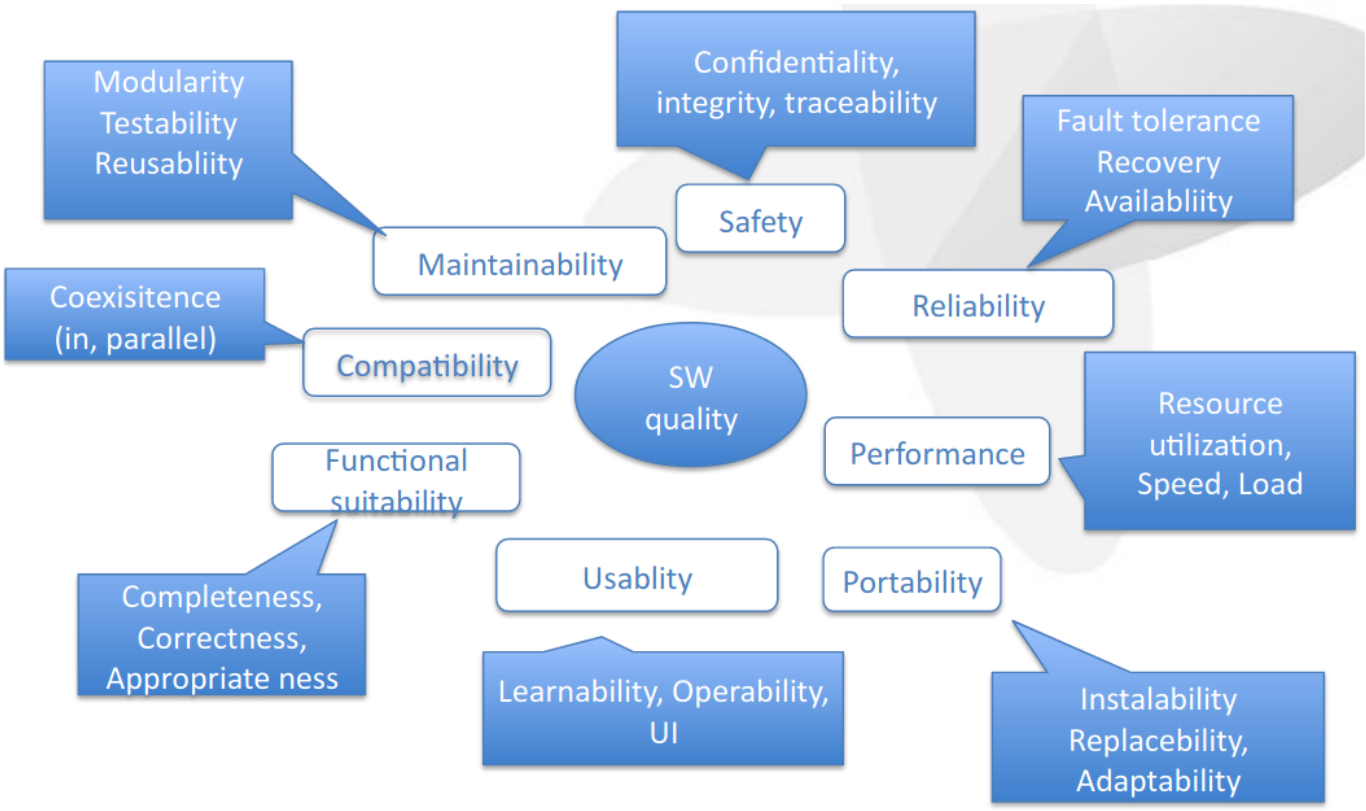
2.12. Minőségbiztosítás, Minőségellenőrzés

Minőségbiztosítás Azok az eljárások amik megpróbálják megakadályozni hogy hibát építsünk be

- fontos része a kompetencia fejlesztés is: Ki, mit, miért? Mire kell figyelni? (code review, milyen változónevek, mit, hol valósítunk meg, milyen dokumentáció, visszakövethetőség)

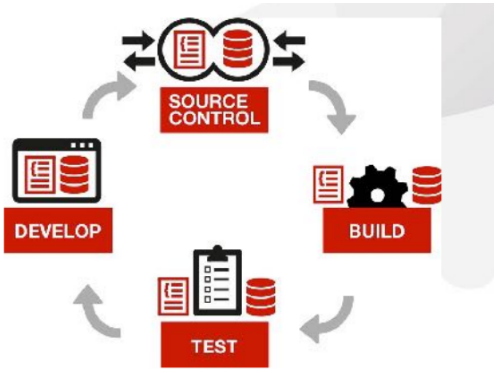
Minőségellenőrzés tesztek futtatása hogy észrevegyük a beépített hibákat mielőtt az ügyfél teszi azt.

2.13. Product quality



3. Continuous Integration

3.1. What's in for the developer?



3.2. Fogalmak

Continuous integration The practice of automatically building and unit testing an entire application frequently, ideally on every source code check-in

- frequent check-ins
- no long-running parallel code branches
- build and test locally before submitting
- shortening the feedback loop for every change

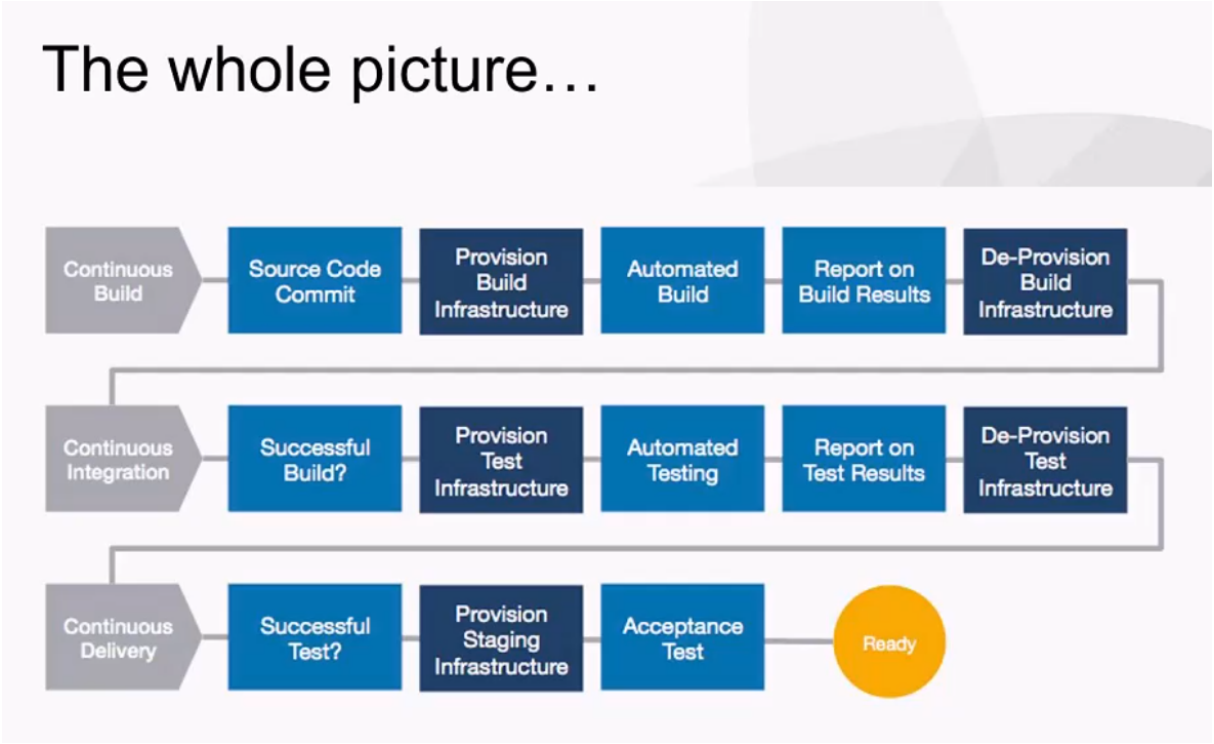
Continuous delivery The practice of deploying every build to a production-like environment and performing automated integration and acceptance testing

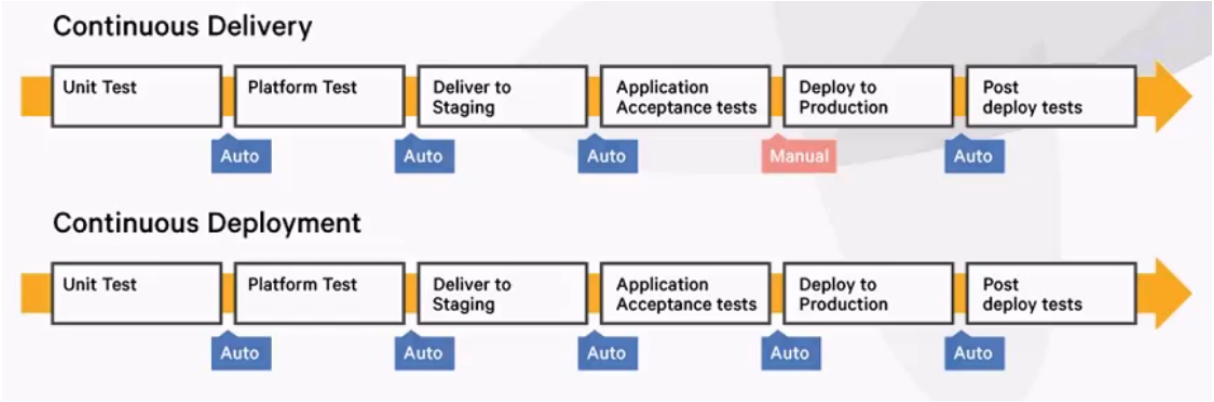
- small-scale deployment on a single server
- mock environments with Docker containers or VM
- mimic production environment
- should end with deployment to a production-like environment
- to see functionality and performance in a real environment

Continuous deployment The practice of automatically deploying every build to production after it passes its automated tests

- every change goes through full automated testing
- deployed automatically to the production environment
- one small change at a time!
- no giant releases with hundreds of changes piled up for months

3.3. Összehasonlítás

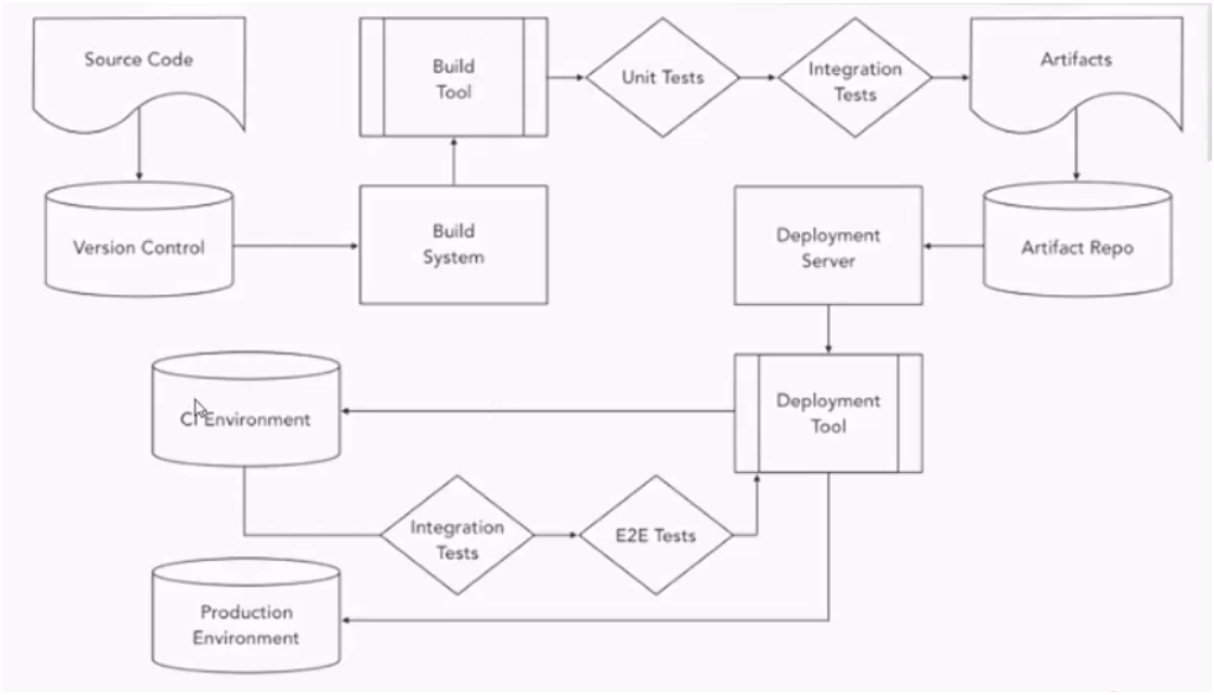




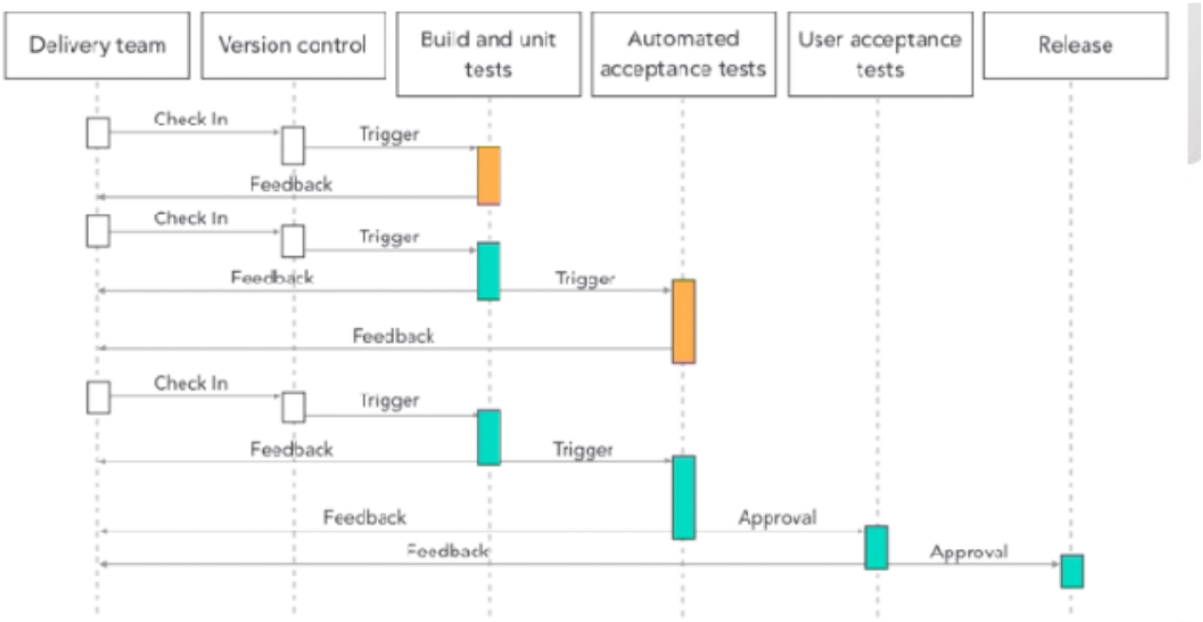
3.4. Benefits CI, CDip

- empowering teams
  - self service system
  - transparent and understandable process of delivery
  - makes the team less reactive
  - reduces the strain between
    - developers and operation
    - QA and security
- Lowered cycle times
  - from weeks or months to hours or even minutes
- Better security
  - less time remediating security issues
  - easier compliance audits
- Rhythm of practice
  - removes stress
  - release date is not a stressful event any more
- More time to be productive
  - less time reworking, more time adding new features

3.5. Build pipeline



3.6. CI Flow diagram



3.7. Version control

practices

- always use version control
- needs to be used by all teams who touch the code
- build and deploy with a single command
- commit often
  - uncompleted features hidden behind a feature flag
- easy to understand commit messages
- don't commit broken code
- commit hooks enforce quality
  - pre-commit hook => fast local test
- careful with secrets

Branching

- consider a master branch approach
- developing off a master gives better performance
  - no need for a lot of branching and pull requests
  - mechanism to handle changes should be small and easy to understand

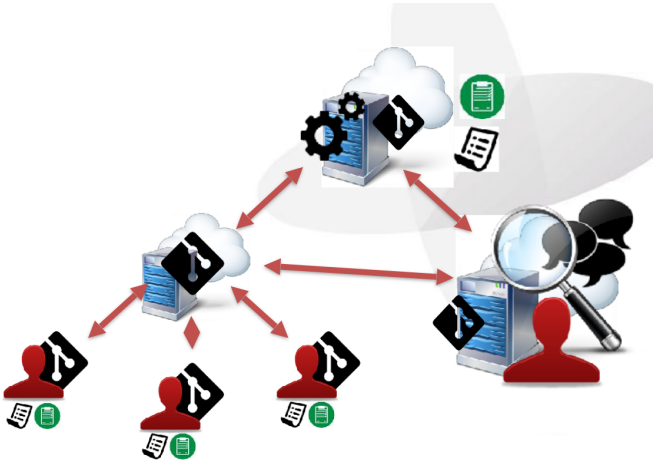
3.8. CI principles and practices

- VC server => CI server
- CI is practise
- Open source tool jenkins
- CI as service
- Start with a clean environment
  - Don't maintain the previous run
- Pass the coffee test
  - Code commit to receiving results less than 5 minutes

CI culture

- Run tests locally before committing
- Do code reviews
- Don't commit new code to broken builds
- Don't leave the build broken No "coat commits"
- Don't remove failing tests

3.9. Review



Review visszajelzés

- 0: nem tudom eldönteni
- -1: szerintem ne
- -2: semmiképp ne
- 1: szerintem igen
- 2: jó, mehet

3.10. Notifications on build progress

- Commit
- Build start
- Build complete
- Deploy to stage
- Deploy to production
- Undeployed code for more than an hour
- Event in the monitoring tool
    - Collerating errors to code changes

3.11. Artifacts

Kész lefordított kódok amik kiszállításra fog kerülni teszt után

Reliability what you tested goes into production

Composability

- language version
- library version
- specific compiler or interpreter
- other dependencies

Security control over what goes into the package

Shareability well understood construct for consuming your code

3.12. Plan ahead

Artifact megtervezése

Packaging formats

- RPM
- debian package
- Python package repo

Dependency management dependency definition in packaging

Artifact repositories

- Nexus
- JFrog Artifactory
- Apache Archiva
- etc

3.13. Testing, Types of testing

Good tests are: fast, reliable, isolate failures

Unit testing
- Performed at build time on a single unit of code and/or artifact without the use of external dependencies or deployment
- JUnit, XUnit, Rspec, etc

Integration testing
- Bringing together pieces of your application as it needs to use external dependencies
- full app, API, running server
- RAML, serverspec

End-to-end (E2E) testing
- exercise the full flow of an application
- Selenium, protractor

Security testing
- looks for flaws in code and runtime to prevent compromises and leaking of data in production
- FindBugs, Fortify, GauntIt

Performance tests
- soak tests (napokig futtatjuk, eláztatjuk)
- spike tests (stressz teszt magas terhelés)
- step tests (lépésenként végrehajtani)

System tests

Acceptance tests

Validation tests

3.14. Terminology

Shift left move testing as early as possible in the dev process

Test fixtures set of objects used to run a test in a well known environment
- dataset, server with known configuration, etc
- artifacts and should be built and managed like one

System under test (SUT) the application and system on which you are running the tests

Cycle time time from work starting on item to delivery into production

Lead time time from requesting an item until it's delivered into production

Mocking or stubing code to stand in for external dependencies to enable unit tests

3.15. Testing philosophy

Test driven development (TDD)

- writing a failing test first, and then writing the code to pass
- refactoring to make it cleaner

Behavior driven development (BDD) writing tests in a simple end user behaviour centric language

Acceptance test driven development (ATDD) agreeing on acceptance tests before development to estabilish what to deliver

3.16. TDD

Never just hope that your code works Prove it, Again...,... and again... , ... and again... , From the first line of code... ,... to deployment. The easiest way is automated testing.

- Write a test
- Watch the test fail
- Write application logic - as simple as possible
- Pass the test
- Refactor, removing duplication
- Pass the test again

3.17. Does unit testing replace all other testing?

There are plenty of other necessary tests beta testing, performance testing, stress testing, integration testing, usability testing ,etc. So, the answer is definitely NOT!

3.18. Does TDD work for everything?

There is much more: Multi-threading, Security testing, UI testing, Game development, etc.

3.19. Unit testing frameworks

For creating, running and managing automated unit tests XUnit, SUnit - Smalltalk, JUnit - Java, NUnit - .NET, PyUnit - Python, CppUnit - C++, OCUnit - Objective-C

3.20. Assertion

Sanity check. Positive: These two strings are equal, This integer is positive. Negative:This object in not NULL. Not the generic flow of code: exceptions, error handling, diagnostic dialog boxes. FAIL NOW!

3.21. Can I change the order of tests?

- The order shouldn't matter
- Controlling suggests dependencies - avoid at all costs
- Unit tests are not to test the whole application
- Unit tests are are only for testing individual units

  In isolation

3.22. Why mocking?

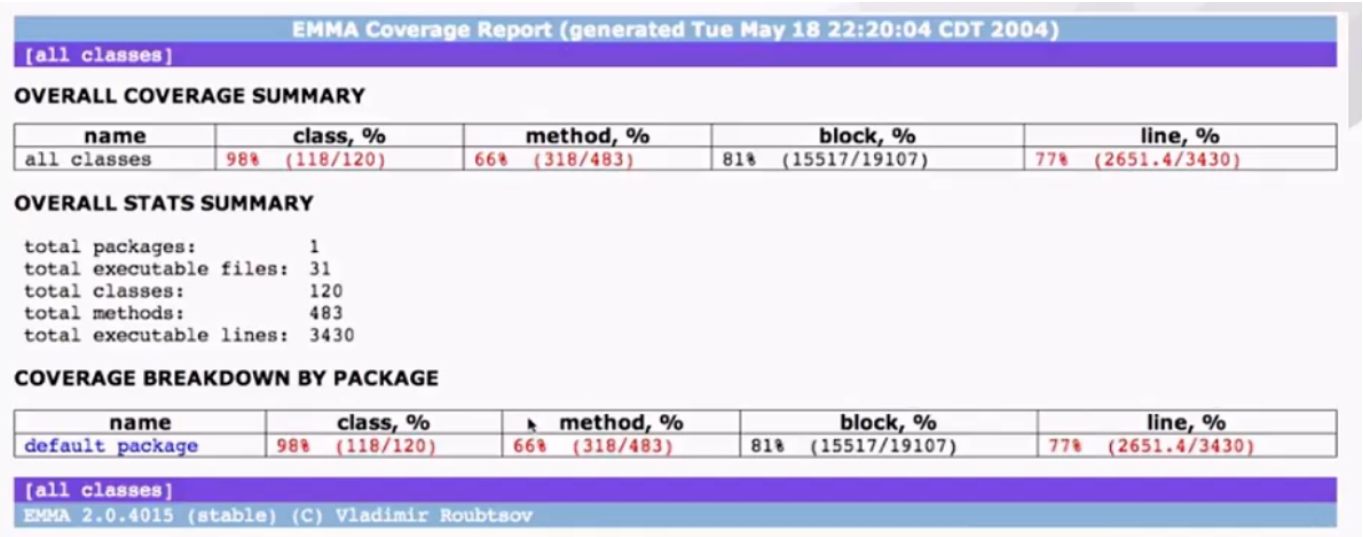
- Why mocking?
- Real object hasn't been written yet
- UI needs human interaction
- Slow or difficult to set up
- External resource (file system, database, network, printer, etc.)
- Non-deterministic behavior

3.23. Mock Object Frameworks

Provide structure for defining mock objects, May remove the need to create a custom class Can help in generating method stubs, Often provide prearranged mock objects.

- File stream
- Console
- Network
- Printer
- etc.

3.24. Measuring code coverage



EMMA Coverage Report (generated Tue May 18 22:20:04 CDT 2004)

[all classes]

OVERALL COVERAGE SUMMARY

name	class, %	method, %	block, %	line, %
all classes	98% (118/120)	66% (318/483)	81% (15517/19107)	77% (2651.4/3430)

OVERALL STATS SUMMARY

total packages: 1
total executable files: 31
total classes: 120
total methods: 483
total executable lines: 3430

COVERAGE BREAKDOWN BY PACKAGE

name	class, %	method, %	block, %	line, %
default package	98% (118/120)	66% (318/483)	81% (15517/19107)	77% (2651.4/3430)

[all classes]

EMMA 2.0.4015 (stable) (C) Vladimir Roubtsov

3.25. What to test?

Test every branch, if, else, and, or, case, for, while, polymorphism. "Test until fear turns to boredom". Use code coverage tools.

3.26. What to avoid

- Interaction with a database or file system
- Require non-trivial network communication
- Require environment changes to run
- Call complex collaborator objects
- Unit tests != other tests

3.27. Testing metrics

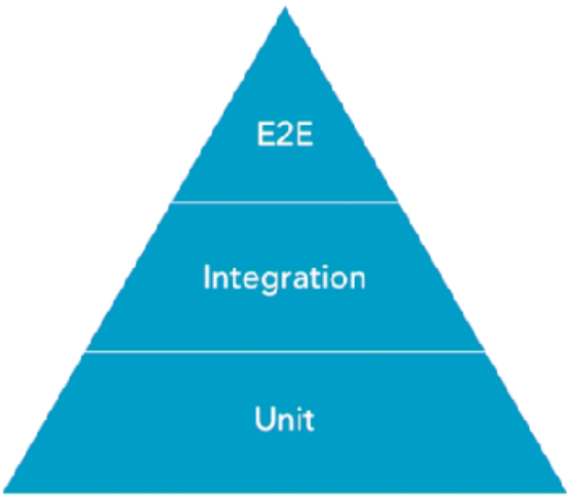
Cycle time time from the start of work to delivery

Velocity Value delivered per unit time

Customer satisfaction How well a product/service met the customer's needs

- NPS (Net Promoter Score)
- Bug reports

3.28. 70-20-10%



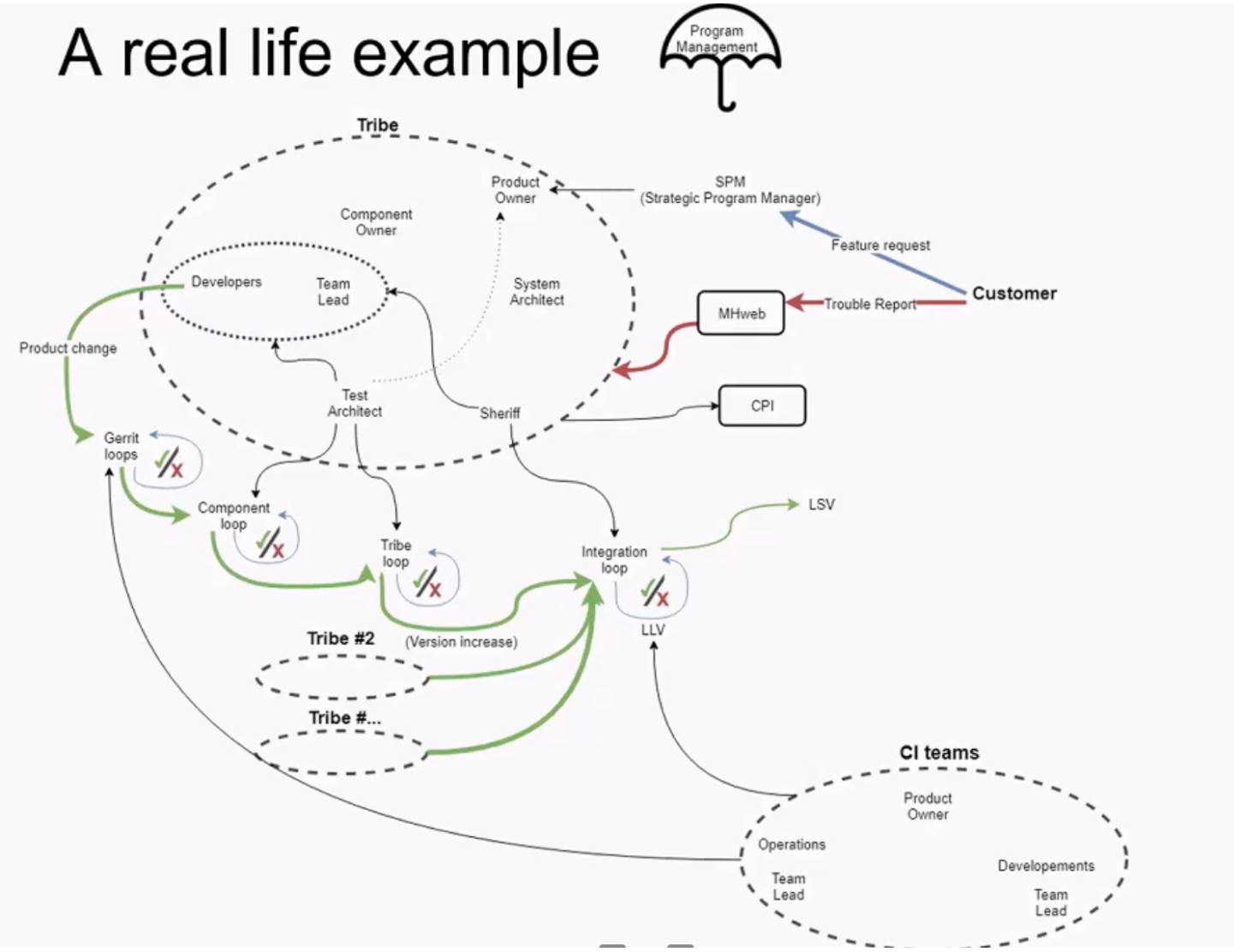
3.29. Deployment

- Same artifact
- Same way
- Same (or at least similar) environment
- Same smoke tests at the end of every deployment
- Small batches
- Changes loosely coupled as much as possible
- Separate deploy and release
  - Blue/green deployment (Zöld élő, kékre új deploy, ezt nem használják az ügyfelek, ha minden teszt sikeres a kékből lesz az új zöld)
  - Feature flags to toggle parts of the code

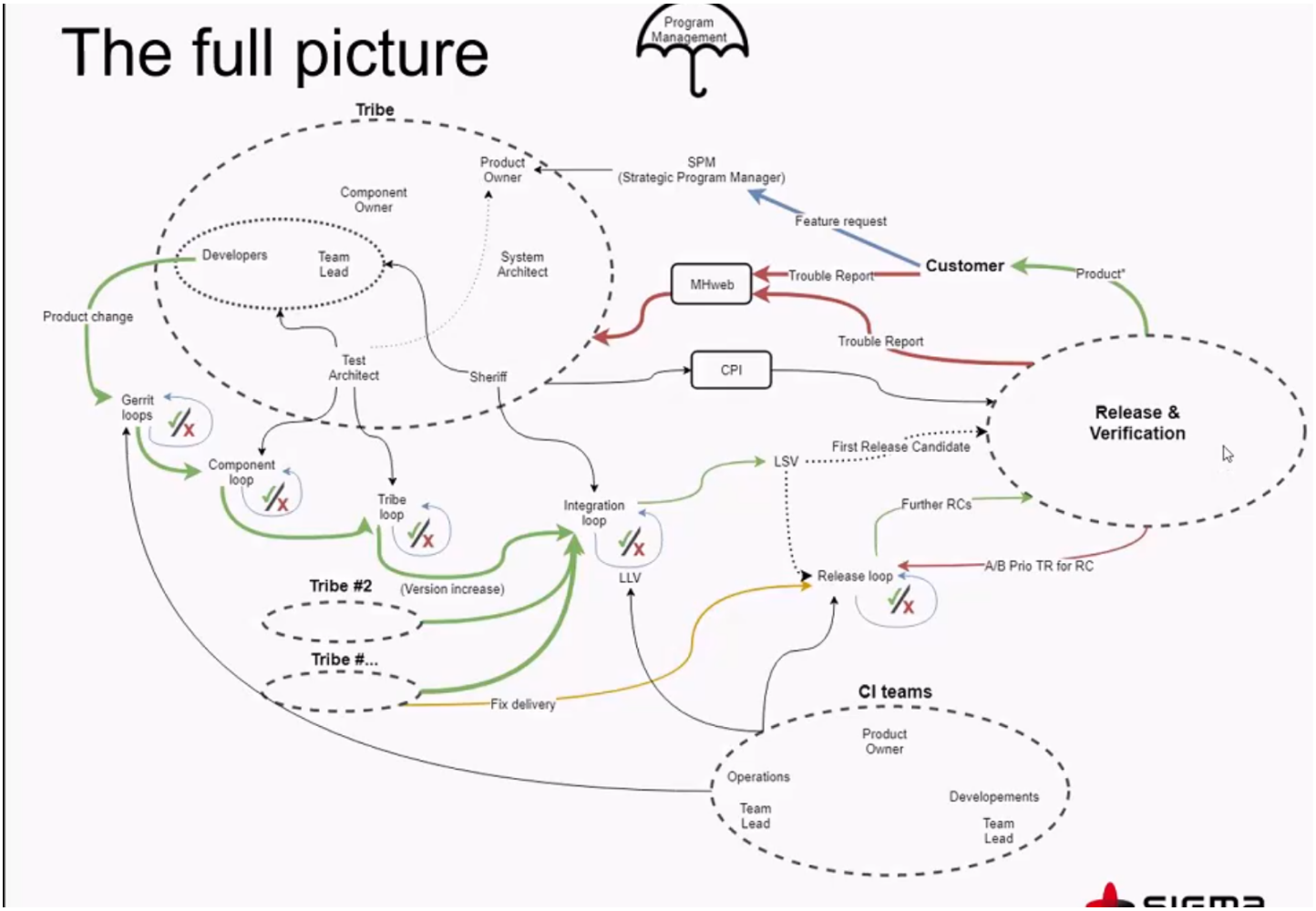
3.30. Best practices

- Each developer is responsible for their check-in
- Small changes
- Build quality in
- Don't check in broken builds
- Automate high quality testing
- Run tests before check in
- Fix inconsistent tests
- Don't ignore or disable tests
- Automate deployment
- Keep the build and deployment fast

3.31. A real life example



3.32. The full picture



4. Selenium