Touko Hjelt

Tietoturvan parhaat käytänteet ohjelmistokehityksessä

Tietotekniikan kandidaatintutkielma

3. kesäkuuta 2015

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Touko Hjelt

Yhteystiedot: touko.j.j.hjelt@student.jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: Tietoturvan parhaat käytänteet ohjelmistokehityksessä

Title in English: Best practices and security in software developement

Työ: Kandidaatintutkielma

Sivumäärä: 23+0

Tiivistelmä: Parhaat käytänteet tarjoavat tehokkaan tavan välittää tärkeää tietotaitoa tietoteknisiä sovelluksia kehitettäessä. Ne soveltuvat hyvin myös tietoturvaratkaisujen periaatteiden välittämiseen. Tässä tutkielmassa tarkastellaan tietoturvan parhaita käytänteitä sovelluskehityksessä, ja muodostetaan yksi mahdollinen kokoelma parhaita käytänteitä, joilla pyritään takaamaan tietoturvan onnistunut implementointi ohjelmistokehityksen elinkaaren jokaisessa vaiheessa. Sen jälkeen perehdytään myös parhaiden käytänteiden mahdollisiin sudenkuoppiin, sekä parhaiden käytänteiden ja innovatiivisuuden väliseen rajankäyntiin.

Avainsanat: Parhaat käytänteet, tietoturva, sovelluskehitys

Abstract: Best practices are an effective way to deliver vital knowledge in IT domain. They can also restore proper security practices and procedures. This study will analyze best practices of information security in software developement, and produce one possible collection of best practices to ensure successfull implementation of information security in any software developement project. Some disadvantages will also be covered, and some thoughts shared with the skirmish between innovation and best practices.

Keywords: Best practices, security, software development

Kuviot

Taulukot

Sisältö

1 Johdanto

Tietotekniikassa parhaat käytännöt (engl. Best Practices) kuvaavat niitä vakiintuneita käytänteitä, joilla on tarkoitus vastata tieteenalan yleisiin haasteisiin tarjoamalla jonkinlainen yleistetty ratkaisumalli (Goodman & Goldman, 2007, s. 308). Parhaista käytännöistä on muodostunut alalla jonkinlainen standardi: Niin suuret kuin pienet yritykset muodostavat yleisiä käytänteitä koskien monenlaisia arkisia haasteita - koskien kysymyksiä, kuten ”millaista on hyvä koodi, sen lisäksi että se toimii ja on tehokasta”, tai ”Miten tietoturva saadaan integroitua sovelluksiin parhaiten”? Esimerkiksi Microsoft on luonut monia, eri tilanteisiin soveltuvia ja hyvin käytännönläheisiä artikkeleita tai koonteja siitä, miten heidän tarjoamillaan työkaluilla tulisi ohjelmoida ”oikein”. Kuten Rob Caron (2000) kirjoittaa, niiden päätarkoitus on parantaa koodin luettavuutta ja ylläpidettävyyttä. Parhaat käytänteet ovat käytössä myös tietoturvan saralla.

Tähän kirjallisuuteen pohjautuvassa kandidaatin tutkielmassa on tarkoitus tarkastella, millaisia parhaita käytäntöjä sovelluskehityksessä tietoturvan osalta on käytössä ja suositellaan käytettävän, millaisia haittoja ja hyötyjä parhaiden käytänteiden käyttöön liittyy, ja tarkastellaan lähemmin, millaisia parhaita käytänteitä AJAX-ohjelmointitekniikassa tietoturvan kannalta käytetään tai voidaan käyttää.

Oman kokemukseni mukaan kaikissa demotehtäviä laajemmissa ohjelmistoprojekteissa tulee vastaan vaihe, jossa ohjelmoija tai ohjelmoijat rupeavat miettimään yhtenäisiä käytänteitä koodin ylläpidettävyyden ja luettavuuden sekä tietoturvan takaamiseksi. Tähän haasteeseen parhaat käytänteet yrittävät perinteisesti vastata luomalla jonkinlaisen normiston - joko tiukan ja kattavan tai väljän - joka ohjaa ohjelmoijia tuottamaan yhdenmukaista koodia. Kuitenkaan ei ole aina selvää, miten tällainen normisto tulisi tehdä, jotta se palvelisi tehtäväänsä, eikä muodostuisi byrokratiaviidakoksi tai unohdetuksi opukseksi jota kukaan ei koskaan lue syystä tai toisesta.

Tietoturva on aina ajankohtainen ja yhä enemmän painoarvoa saava tutkimus- ja

käytännön ala. Tietotekniikassa eräänlainen viattomuuden ala on ohitse, eivätkä ohjelmistosuunnittelijat ja -kehittäjät voi enää ajatella tekevänsä pelkästään sovelluksia, joita käytetään johonkin hyvään tai, että käyttäjät olisivat vain hyväntahtoisia sovellusta kohtaan. Jokainen tehty ohjelmistosovellus on vaarassa ja elinkaarensa aikana alttiina lukuisille pahantahtoisille hyökkäyksille.

Parhaat käytänteet ovat oikein käytettynä vahva työkalu ja apu ohjelmistokehityksen parissa työskenteleville. Ne tarjoavat yhdenmukaisen ja käytännössä koetellun toimintamallin helpottamaan ohjelmistokehitystä auttamalla välttämään sudenkuoppia, tiedostamaan riskejä, sekä tekemään koodista parempaa ja ylläpidettävämpää. Kuitenkin parhaisiin käytäntöihinkin liittyy ongelmia. Uudelle sovellusalueelle saatetaan esimerkiksi kopioida sen enempää ajattelematta jossain muualla toimivia parhaita käytäntöjä ilman, että pyritään mitenkään varmistamaan niiden soveltuvuus tälle uudelle alueelle.

Tässä tutkielmassa pyritään löytämään ja kriittisesti tarkastelemaan parhaita käytäntöjä turvallisempien ja vakaampien sovellusten kehittämiseksi. Pyrkimys on vastata kysymyksiin, kuten ”millä parhailla käytänteillä pystytään takaamaan sovellusten parempi tietoturva” ja ”mitä rajoituksia ja puutteita parhaat käytännöt sisältävät”.

On olemassa viitteitä siitä, että parhaita käytänteitä käytetään ”väärin”, ymmärtämättä niitä kysymyksiä ja ongelmia mihin kullakin parhaalla käytänteellä pyritään vastaamaan. Parhaiden käytänteiden määrittely ei myöskään ole ongelmatonta (Quayzin, 2011). Oxfordin sanakirja määrittelee parhaiden käytänteiden tarkoittavan niitä ”kaupallisia tai ammatillisia proseduureja, joiden on hyväksytty tai säädetty olevan oikeita tai tehokkaimpia” (Oxford dictionary, 2015). Entä millä perusteella jonkin käytänteen voidaan sanoa kuuluvan parhaisiin käytänteisiin tai milloin paras käytänne lakkaa olemasta paras? Miten sen parhaus testataan?

2 Kuinka parhaiden käytänteiden malli sovelluskehityksen elinkaaressa voidaan muodostaa

Artikkelissaan ”Software Security” McGraw (2004) kuvailee kahta erilaista tapaa toteuttaa tietoturvaratkaisuja: sovellustason (*Application security*) ja ohjelmistotason (*Software security*) tietoturvaa. Hän argumentoi, että sovellustason tietoturva ei ole riittävä, vaan tietoturva täytyy saada integroitua kiinteäksi osaksi itse sovelluksia, sekä esittää yhden mallin jonka avulla tämä voisi onnistua. Tämän periaatteen päälle Ansar-Ul-Haque Yasar et. al. rakentavat sovelluskehityksen elinkaarelle parhaiden käytänteiden mallin. He käyvät läpi artikkelissaan ”Best Practices for Software Security: An Overview” (Yasar & Preuveneers & Berbers & Bhatti, 2008) joukon tietoturvaa parantavia parhaita käytäntöjä, ja ehdottavat, että tietoturvan parantamiseksi näitä käytäntöjä tulisi soveltaa sovelluskehityksen elinkaaren (Sofware Development Life Cycle, tai SDLC) kaikissa vaiheissa parhaan hyödyn saamiseksi.

Kumar, Pandey ja Ahson keskittyvät artikkelissaan ”Security in Coding Phase of SDLC” (Kumar & Pandey & Ahson, 2007) yhteen elinkaaren vaiheeseen ja ehdottavat muutamia käytännön ohjeita, joilla ohjelmiston elinkaaren ohjelmointivaiheessa pystyttäisiin parhaiten takaamaan ohjelmiston laatu ja sitä kautta vakaa tietoturva. He painottavat, että tietoturvan kannalta on välttämätöntä, että siihen panostetaan ohjelmistokehityksen jokaisessa vaiheessa, ja että tietoturva on jokaisen kehitykseen osallistuvan asia.

Parhaisiin käytänteisiin liittyy kuitenkin myös omat riskinsä ja näitä riskejä tai heikkouksia Quyazin lähtee käsittelemään artikkelissaan ”Are best practices really best practice?” (Quayzin, 2011). Hän esittää huolen siitä, että parhaiden käytäntöjen käyttämisestä on tullut automaatio. Hänen mukaansa olisikin tärkeä tapauskohtaisesti miettiä, ovatko suunnitellut parhaat käytännöt todella parhaita siihen tarkoitukseen johon niitä kulloinkin halutaan soveltaa. Hänen pohdintansa ei rajoitu varsinaisesti tietotekniikan tai tietoturvan alueelle, vaan hän pyrkii kriittisesti tarkastelemaan koko parhaiden käytäntöjen konseptia.

Lopuksi Kozlova, Hadenkamp ja Kopanakis ehdottavat artikkelissaan ”Use of IT Best Practices for Non-IT Services” (Kozlova & Hasenkamp & Kopanakis, 2012), että tietotekniikassa kehittyneet parhaat käytänteet voisivat hyödyttää muita teollisuuden aloja. Tämän mahdollistaa se, että jo nyt monet teollisuuden osa-alueet ja käytänteet ovat osin limittäisiä tietotekniikan vastinpariensa kanssa. Esimerkiksi tietoteknisten alojen *tapahtumien hallinnointi* (Inscident management) ja yleisemmin teollisuuden alojen *korjaus ja kunnostus* (Repair and restoration) ovat abstraktilla tasolla hyvin samankaltaisia toimintoja. Tästä syystä ne voisivat hyötyä tietotekniikan parhaiden käytänteiden omaksumisesta.

2.1 Käsitteitä

Termillä *paras käytänne* (tai *parhaat käytänteet*) tässä tutkielmassa tarkoitetaan, jos muuta ei mainita, ”kaupallisia tai ammatillisia proseduureja, joiden on hyväksytty tai säädetty olevan oikeita tai tehokkaimpia” (Oxford dictionary, 2015).

Sovelluskehityksen elinkaari (SDLC) kuvaa sovelluskehityksen eri vaiheita sovelluksen vaatimusten määrittelystä valmiin sovelluksen ylläpitoon. Sovelluskehityksen elinkaaresta on olemassa useita variaatioita, ja tässä tutkielmassa sovelluskehityksen elinkaaresta puhuttaessa viitataan Yasar & Preuveneers & Berbers & Bhatti (2008) käyttämään versioon, jonka vaiheet ovat

1. vaatimusten analysointi,
2. suunnittelu,
3. toteutus,
4. testaus ja
5. ylläpito

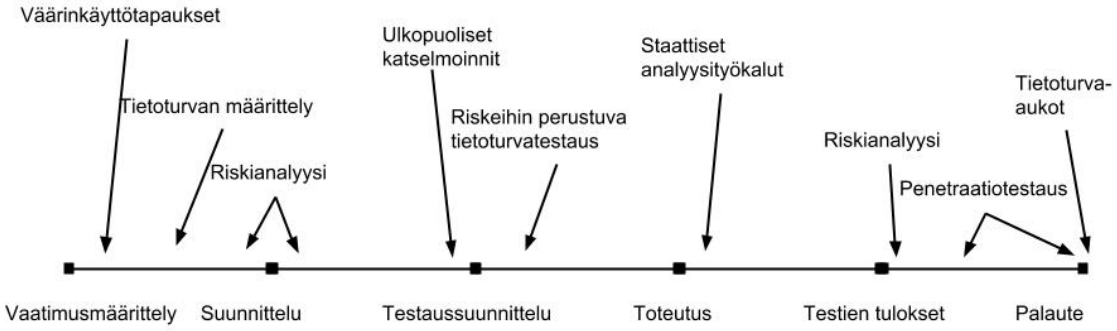
3 Tietoturvan parhaat käytänteet sovelluskehityksen elinkaaressa

Sovellusten tietoturva on mahdollista jakaa McGraw (2004) mukaan kahdeksi osaksi: sovellustason tietoturvaksi ja ohjelmistotason tietoturvaksi. Sovellustason tietoturvalla hän tarkoittaa jälkeenpäin implementoituja tietoturvaratkaisuja, kuten virustentorjuntaohjelmistoja ja palomuureja sekä haavoittuvuuksien etsimistä ja korjaamista, ja ohjelmistotason tietoturvalla proaktiivista lähestymistapaa, jossa tietoturva pyritään implementoimaan ohjelmistoon tai sovellukseen itseensä. Tämä edellyttää McGrawn mukaan sitä, että tietoturva huomioidaan sovelluskehityksen elinkaaren jokaisessa vaiheessa, ja että tietoturva on jokaisen ohjelmiston kehitykseen kuuluvan henkilön vastuulla.

Kumar & Pandey & Ahson (2007) argumentoivat, että tietoturvan implementoiminen jälkeenpäin valmiiseen tuotteeseen on kaikkein kallein ja tehottomin tapa rakentaa tietoturvaa. Heidän mukaansa kulut ohjelmiston haavoittuvuuden havaitsemisesta ohjelmiston kehitysvaiheessa ovat vain kaksi prosenttia siitä, mitä saman haavoittuvuuden korjaaminen maksaa valmiista tuotteesta. Sekä kulujen että tehokkuuden näkökulmasta tietoturvan integroiminen sovellukseen jo kehitysvaiheessa on paras keino, riippumatta ohjelmistokehityksen malleista: Niin vesiputousmalli, kuin ketterät menetelmät hyötyvät tietoturvakäytänteisiin panostamisesta. Myös he kannattavat tietoturvan ottamista sovelluskehityksen keskiöön koko sovelluskehityksen elinkaaren ajan.

3.1 Tietoturvaa alhaalta ylöspäin

McGraw (2004) selittää sovellustason tietoturvan levinneisyyttä ja yleisyyttä sillä luonnollisella syyllä, että useissa suurissa yrityksissä tietoturvasta ovat vastuussa järjestelmien ylläpitäjät, joilla ei ole näkemystä (eikä usein mahdollisuuksiakaan) sovellusten tekemisestä vaan käyttämisestä. Tästä syystä he luonnollisesti ovat rakentaneet tietoturvaa sovellusten, kuten palomuurien ja virusohjelmistojen avulla.



Kuvio 1. McGrawn esitys parhaista käytänteistä tietoturvan implementoimiseen ohjelmistokehityksen elinkaaren eri vaiheisiin vesiputousmallin mukaisesti

Taustalla oleva ajatusmalli on, että haavoittuvia ja herkkiä ohjelmistoja pitää pyrkiä suojelemaan.

Edellä esitetty ajatusmalli tarjoaa tietoturvaa ylhäältä alaspäin. McGraw (2004) kuitenkin esittää, että toimintamallin pitäisi olla päinvastainen. Sovellusten itsensä pitäisi pystyä puolustamaan itseään hyökkäyksiä vastaan. Kumar & Pandey & Ahson (2007) esittävät, että kyseessä on paradigman muutos. Aikaisemmin ajateltiin, että ohjelmiston tehtävä on tarjota ratkaisu johonkin ongelmaan, ja ohjelmistoa käytettäisiin ”hyvään”. Sen sijaan pitäisi ajatella, että ohjelmistojen pitää aikaisemman lisäksi pystyä toimimaan vihamielisessä ympäristössä, missä käyttäjä ei olekaan hyväntahtoinen.

Kuvio 1 esittää McGrawn näkemyksen sovellustason tietoturvan implementoimisesta ohjelmistokehitykseen. Kuviossa on käytetty esimerkkinä vesiputousmallin mukaista ohjelmiston valmistusprosessia, mutta hänen mukaansa samat parhaat käytänteet ovat sovellettavissa myös iteratiiviseen kehitykseen. Tällöin parhaat käytänteet tulevat vain läpikäydyksi useamman kerran ohjelmistokehityksen elinkaaren aikana.

Hänen mukaansa tietoturva kuuluu ehdottomasti vaatimusmäärittelyvaiheeseen (*requirements and use cases*). Hän suosittelee kirjaamaan käyttötapausten lisäksi myös vää-

rinkäyttötapaukset, joiden tehtävä on kuvata kuinka järjestelmä reagoi hyökkäyksiin. Tällä tavoin saadaan määriteltyä tarkalleen mitä, mitä vastaan, ja kuinka kauan järjestelmää pyritään turvaamaan. Kaikki informaatio, mitä järjestelmä pystyy tallentamaan siihen kohdistuvista hyökkäyksistä, on myös arvokasta, sillä sen pohjalta pystytään parantamaan kyseisen järjestelmän, ja yleisestikin järjestelmien tietoturvaa tulevaisuudessa.

Suunnitteluvaiheessa (*design*) suunnittelijoiden täytyy huomioida ja dokumentoida tarkkaan käytetyt tietoturvaperiaatteet ja olettamukset. McGraw suosittelee käytettäväksi riskianalyysiä ja ulkopuolisten tekemiä auditointeja. Toteutusvaiheessa (*coding*) huomio tulisi kiinnittää lähdekoodin staattiseen analysointiin siihen tarkoitettujen valmiiden työkalujen avulla. Näillä pyritään löytämään tyypillisiä haavoittuvuuksia lähdekoodista. Toteutusvaiheen jälkeen suoritetaan riskianalyysin pohjalta testaaminen. McGraw pitää myös penetraatiotestausta suositeltavana lisänä tässä vaiheessa sovelluskehityksen elinkaarta, mutta muistuttaa, että pelkkä musta laatikko -testaus ei ole riittävää mahdollisten haavoittuvuuksien löytämiseksi.

McGraw nostaa myös ohjelmistokehittäjien kouluttamisen parhaiden käytänteiden joukkoon.

3.2 Ehdotuksia sovelluskehityksen parhaiksi käytänteiksi

Seuraavaksi tarkastellaan kolmen eri tahon tekemiä ehdotuksia sovelluskehityksen parhaiksi käytänteiksi tarkastelemalla Yasar & Preuveneers & Berbers & Bhatti (2008) niistä luomia listauksia. Lopuksi listaukset yhdistetään ja sijoitetaan sovelluskehityksen elinkaaren eri vaiheisiin vesiputousmallin mukaan.

3.2.1 Redwinen ja Davisin listaus

McGrawn kanssa hyvin pitkälti samankaltaisiin suosituksiin päätyivät myös Redwine & Davis (2004). Yasar & Preuveneers & Berbers & Bhatti (2008) kokosivat heidän artikkelinsa pohjalta yhdeksän kohdan listan sovelluskehityksen elinkaaren aikana käytettävistä parhaista käytänteistä. Se on kirjattu vesiputousmallin mukaises-

ti, mutta on sovellettavissa muihinkin ohjelmistotuotantoprosesseihin. Parhaat käytänteet sijoittuvat sovelluskehityksen elinkaaren eri vaiheisiin pitkälti kuvion 1 mukaisesti.

1. Väärinkäyttötapaukset (Abuse Cases),
2. Tietoturvan vaatimusmäärittely (Security Requirement),
3. Riskianalyysi (Risk Analysis),
4. Ulkopuoliset auditoinnit (External review),
5. Riskeihin perustuva testaus (Risk-based security tests),
6. Staattinen analyysi (Static analysis),
7. Riskianalyysi (Risk analysis),
8. Läpäisytestaus (Penetration testing),
9. Tietoturva-aukot (Security breaks)

Väärinkäyttötapaukset ja tietoturvan vaatimusmäärittely kuuluvat sovelluskehityksen vaatimusmäärittelyvaiheeseen, ja niiden avulla on tarkoitus kartoittaa kaikki mahdolliset tilanteet joissa sovellus voisi käyttäytyä epänormaalisti. Näiden tilanteiden pohjalta on mahdollista seuraavaksi määritellä minkälaisia tietoturvaratkaisuja tarvitaan sovelluksen turvallisen käytön takaamiseksi (Yasar & Preuveneers & Berbers & Bhatti, 2008). Riskianalyysi kuuluu olennaisena osana suunnittelu-, toteutus- ja testausvaiheisiin, mahdollistaen näin realistisemman riskien hallinnan, joka ei perustu pelkästään tavoitteisiin vaan myös itse sovelluksen toimintavalmiuteen ja kehityksessä havaittuihin uusiin riskitekijöihin. Sen tavoitteena on muodostaa suunnitelma jokaisen tunnetun riskin varalle, sekä kartoittaa riskien todennäköisyyttä ja vaikutuksia.

Staattinen analyysi auttaa paljastamaan ohjelmointivaiheessa tapahtuvia tyypillisimpiä tietoturvariskejä, kuten tietotyyppien ja puskurien ylivuotoja, ja sitä varten on olemassa useita valmiita työkaluja (McGraw, 2004), (Yasar & Preuveneers & Berbers & Bhatti, 2008).

Yasar & Preuveneers & Berbers & Bhatti (2008) suosittelevat myös palautteen keräämistä kentältä toimituksen jälkeen.

3.2.2 Razvanin listaus

Yasar & Preuveneers & Berbers & Bhatti (2008) jatkavat listausta parhaista käytänteistä listaamalla Razvan (2001) ehdotuksen pahraista käytänteistä (myöhemmin PK2). Se on yhteensovitettavissa Redwine & Davis (2004) listauksen kanssa lisäten siihen joukon projektiin osallistujilta vaadittavia ominaisuuksia, kuten luovuus.

1. Suojele oikeita asioita ja heikointa lenkkiä (Relevancy and weakest links),
2. Selkeys (Clarity),
3. Laatu (Quality),
4. Kaikkien asia (Involve all Stakeholders),
5. Teknologiset prosessit (Technology processes),
6. Toipumiskyky (Fail Safe Operation),
7. Puolustuksen syvys (Defense in Depth),
8. Tarvittavat oikeudet (Principle of appropriate privileges),
9. Asiakkaiden kuunteleminen (Interacting with users),
10. Luotettavat prosessit (Trust boundaries),
11. Kolmannen osapuolen sovellukset (Third-party software),
12. Arkaluonteisen datan käsittely (Manipulation of sensitive data),
13. Altista hyökkäyksille jo varhain (Attack first. On paper!),
14. Ole luova (Think outside of the box),
15. Ole nöyrä (Be humble),
16. Deklaratiivinen vastaan programmatiivinen (Declarative vs. programmatic),
17. Katselmoinnit ovat ystäviäsi (Rewiews are your best allies)

Listauksesta on luettavissa, että ohjelmistoja suunnittelevat ja toteuttavat ihmiset, joilla on taipumus tehdä virheitä. Oikeanlaisilla käytänteillä virheiden määrää saadaan pienennettyä ja seurauksia lievennettyä. Kyse on pohjimmiltaan riskien hallinnasta (Yasar & Preuveneers & Berbers & Bhatti, 2008), (Kumar & Pandey & Ahson, 2007).

Jatkossa käsiteltäviä parhaita käytänteitä on syytä avata hieman. Tässä laadulla (PK2.3) tarkoitetaan sekä sovelluksen että tietoturvaratkaisujen laatua. Nöyryydellä viita-

taan ehkä hieman epäselvästi sovelluksen jatkokehitykseen ja paranteluun ajan kuluessa. Nöyryys sovelluskehittäjän ominaisuutena ei varmasti ole ainakaan negatiivinen asia.

3.2.3 Wheelerin listaus

Viimeisen Yasar & Preuveneers & Berbers & Bhatti (2008) esittämän listauksen tekijä on Wheeler (2003) (myöhemmin PK3). Se on kahta muuta spesifimpi ehdottaen selkeitä käytänteitä tietoturva-aukkojen ehkäisemiseksi.

1. Validoi syötteet (Validat all input),
2. Vältä puskurien ylivuotoja (Avoid buffer overflow),
3. Varmista käyttöliittymä (Secure the interface),
4. Erota data ja ohjain toisistaan (Separate Data and Control),
5. Minimoi oikeudet (Minimize Privileges),
6. Yksinkertaista toiminnallisuudet (Minimize the functionality of a component),
7. Käytä turvallisia lähtöarvoja ja muokkaa varovasti (Configure safely and use safe defaults),
8. Alusta turvallisesti (Load Initialization Values Safely),
9. Takaa toipumiskyky (Fail Safe),
10. Vältä kilpailevia säikeitä (Avoid race conditions),
11. Luota vain luotettaviin kanaviin (Trust only trustworthy channels),
12. Luo turvallinen tiedonkulku (Setup a trusted path),
13. Ole yhtenäinen (Use internal consistency),
14. Rajoita resursseja (Self-limit resources),
15. Estä XSS (Prevent cross site malicious content),
16. Ehkäise semanttiset hyökkäykset (Foil semantic attacks),
17. Ole tarkkana tietotyyppien kanssa (Be careful with data types),
18. Käytä kolmansia osapuolia harkiten (Carefully callout to other resources),
19. Näytä informaatiota ulospäin älykkäästi (Send information back judiciously)

3.3 Sovelluskehityksen parhaat käytänteet koostettuna

Tämän jälkeen Yasar & Preuveneers & Berbers & Bhatti (2008) sijoittivat edellä mainitut kolme listausta sovelluskehityksen elinkaaren eri vaiheisiin taulukko 1:n mukaisesti. Taulukko on suuntaa antava ja sovellettavissa. Monet parhaista käytänteistä ovat yleishyödyllisiä ja järkeviä periaatteita ja riippumattomia sovelluskehityksen vaiheista. Esimerkiksi paras käytänne ”Ole nöyrä” (PK2.15) on sijoitettu vain ylläpitovaiheeseen, vaikka se on varmasti hyödyllinen kaikissa vaiheissa, joskin tarpeen mukaan eri tavalla tulkittuna. Sama koskee myös parasta käytännettä ”Kaikkien asia” (PK2.4), joka on sijoitettu em. listauksessa pelkästään testausvaiheeseen. Pelkästä toiston välttämisestä ei ole kyse, sillä esimerkiksi ”Laatu” (PK 2.3) on sijoitettu sekä vaatimusmäärittely-, suunnittelu-, että toteutusvaiheisiin.

3.3.1 Toteutusvaiheen parhaat käytänteet

Edellisessä luvussa käsiteltyjen käytänteiden lisäksi Kumar & Pandey & Ahson (2007) listaavat neljä toteutusvaiheen parasta käytännettä, jotka ovat

1. Ohjelmointistandardit (Coding standards),
2. Katselmoinnit (Code reviews),
3. Tietoturvan yksikkötestaus (Unit testing for security) ja
4. Vikojen hallinta (Defect management)

Ohjelmointistandardeilla tarkoitetaan yhdessä sovittuja käytänteitä kuinka ohjelmistossa käsitellään turvallisesti käyttäjän antamia syöteitä, merkkijonoja, tilapäisiä tiedostoja, ja virhetilanteita. Näitä standardeja on tärkeä ylläpitää, jotta potentiaalisia vikoja saadaan karsittua pois jo tässä vaiheessa.

Katselmoinnit ovat tilanteita joissa joku muu kuin koodin kirjoittaja käy läpi kirjoitettua koodia. On olennaista, että katselmoinneissa tarkastellaan ohjelmistokoodia myös tietoturvan näkökulmasta. Kumar & Pandey & Ahson (2007) suosittelevat, että tietoturvaan liittyvät katselmoinnit ovat omia itsenäisiä tapahtumia.

Taulukko 1. Yasar et. al. tekemä kooste parhaista käytänteistä ja niiden sijoittumisesta sovelluskehityksen elinkaaren eri vaiheisiin. (Yasar & Preuveneers & Berbers & Bhatti, 2008)

Sovelluskehityksen elinkaaren vaihe	Parhaat käytänteet
Vaatimusmäärittely	PK1.1, PK1.2, PK1.3 PK2.1, PK2.2 PK2.3 PK3.3, PK3.17
Suunnittelu	PK1.3 PK1.4, PK1.5 PK2.3, PK2.5, PK2.6, PK2.8 PK2.10, PK2.12, PK2.13, PK2.14 PK3.3, PK3.4, PK3.5, PK3.6 PK3.9, PK3.10, PK3.11, PK3.13 PK3.18
Toteutus	PK1.6 PK2.3, PK2.5, PK2.6, PK2.7 PK2.8, PK2.11, PK2.16 PK3.1, PK3.2, PK3.5, PK3.7 PK3.8, PK3.9, PK3.12, PK3.14 PK3.15
Testaus	PK1.7, PK1.8 PK2.4, PK2.17 PK3.10, PK3.15, PK3.19
Ylläpito	PK1.8, PK1.9 PK2.9, PK2.15, PK2.17 PK3.7, PK3.10, PK3.16

Yksikkötestauksessa suositellaan kiinnitettävän erityistä huomiota komponentteihin, jotka käsittelevät käyttäjältä tai muilta järjestelmiltä tulevia syötteitä, tiedostomuotoja, enkryptattuja verkkotoimintoja ja käyttäjien autentikointia.

Vikojen hallinnan tavoite on, että jokaiselle havaitulle vialle pystytään määrittämään prioriteetti ja laajuus sekä vastuuhenkilö(t), joka tietyn aikarajan sisällä korjaa vian. Vioittuneet komponentit pitää korjaamisen jälkeen testata uudestaan sellaisilla yksikkötesteillä jotka pystyvät havaitsemaan korjatun vian.

Sekä Yasar & Preuveneers & Berbers & Bhatti (2008) että Kumar & Pandey & Ahson (2007) painottavat artikkeleissaan syötteiden varmentamista, virhetilanteiden käsittelyä ja staattisten analyysityökalujen käyttöä ohjelmistokehityksen toteutusvaiheessa parhaina käytänteinä tietoturvan takaamiseksi.

4 Parhaiden käytänteiden kritiikkiä

Parhaat käytänteet ilmiönä ovat saaneet osakseen myös kritiikkiä. Quayzin (2011) huomauttaa, että parhaiden käytänteiden käytöstä on joissain tapauksissa tullut itsessään paras käytänne. Tämä on johtanut parhaiden käytänteiden valta-asemaan, jossa niitä käytetään ymmärtämättä riskejä joita niihin väistämättä liittyy. Hän listaa viisi eri kritiikin aihetta: standardisaation (*standardisation*), rajoittuneen rationaalisuuden (*bounded rationality*), rajallisen todistettavuuden (*limited validity*), rajallisen elinkaaren (*limited life*) sekä innovoinnin hidastavuuden (*hindrance to innovation*).

Parhaiden käytänteiden painottaminen johtaa Quayzin (2011) mukaan standardoituihin ratkaisumalleihin, ja näiden ongelma taas on sovellettavuus uudelle ongelma-alueelle. Standardi ratkaisu toimii vain silloin kun sitä sovelletaan ongelmaan, joka on tarpeeksi samanlainen kuin alkuperäinen ongelma. Parhaiden käytänteiden kohdalla tämä tarkoittaa sitä, että paras käytänne, joka toimii hyvin joissain tapauksissa, ei välttämättä toimi ollenkaan niin hyvin samankaltaisessa, mutta jostain kohtaa erilaisessa tapauksessa. Tarvitaankin harkintaa ja ongelma-alueen tuntemusta parhaiden käytänteiden määrittelyssä, jotta valitaan käytänteitä, jotka ovat parhaita, eivät vain hyviä tai melko hyviä.

Toinen Quayzin (2011) mainitsema ongelma parhaissa käytänteissä liittyy ihmisluonnon taipumukseen suosia tuttuja toimintamalleja jotka tiedämme ennestään toimiviksi (siis nimenomaan parhaita käytänteitä). Tätä taipumusta kutsutaan rajoittuneeksi rationalisuudeksi. Rajoittunut rationaalisuus voi johtaa vääristyneeseen näkemykseen aihealueen riskeistä. Voimme myös ihastua hypoteeseihimme siinä määrin että muokkaamme riskien kuvaa hypoteesejamme suosiviksi. (Lunn, 2008) Meillä on siis mahdollisesti sisäsyntyinen taipumus etsiä ja keksiä parhaita käytänteitä ongelmia kohdatessamme, mutta samalla saatamme muokata käsitystämme ongelmasta siten että se tukee keksimiämme parhaita käytänteitä.

Parhaita käytänteitä on harvoin testattu kovin suurella tilastollisella otannalla. Ne ovat ennemminkin syntyneet arkipäivän huomioista siitä mikä toimii ja mikä ei. Ne

siis usein sisältävät epävarmuustekijöitä ja induktiivista päättelyä varsin pienestäkin osajoukosta kohti suurempaa. Ne eivät esimerkiksi aina kykene tunnistamaan äärimmäisen harvinaisia riskejä joiden realisoitumistodennäköisyys on hyvin pieni. Parhaat käytänteet ovat siis usein rajallisesti todistettuja.

Parhaiden käytänteiden rajallinen elinkaari viittaa ongelmaan, milloin paras käytänne lakkaa olemasta paras käytänne. Tietysti silloin kun löydämme jonkin paremman. Tämä tapahtuu useimmiten kahdella eri tavalla. Joko kun saamme kerättyä tarpeeksi paljon todistusaineistoa osoittaaksemme, että käyttämämme käytänne ei itseasiassa olekaan paras, tai kun kehittyvä teknologia ajaa jonkin parhaan käytänteen yli tehden siitä auttamattomasti vanhentuneen ja hyödyttömän uusissa olosuhteissa. Pelkkä tukeutuminen parhaisiin käytänteisiin siksi että ne ovat parhaita käytänteitä voi estää meitä löytämästä parempia käytänteitä.

Tästä on johdettavissa suoraan parhaiden käytänteiden viides kritiikki: innovoinnin hidastavuus. Parhaat käytänteet ovat olemukseltaan innovaation vastakohta. Ne keräävät ja säilyttävät tietoa siitä mikä on toiminut aikaisemmin, ja muokkaamaan niistä toimintamalleja, kun taas innovointi pyrkii haastamaan vanhat toimintamallit ja lähestymään ongelmaa kokonaan uudesta näkökulmasta ja tätä kautta parantamaan tuotettavuutta tai luotettavuutta. Parhaista käytänteistä voi tulla tekosyy olla kehittämättä parempia toimintamalleja.

Edellä mainitsemistaan puutteista huolimatta Quayzin (2011) pitää parhaiden käytänteiden konseptia hyödyllisenä sekä olennaisena osana innovointia. Parhaat käytänteet auttavat välttämään ”pyörän keksimistä uudelleen”. Parhaat käytänteet ovat hänen mukaansa innovaation lähde. Ne haastamalla voi löytää uutta.

5 Yhteenveto

Parhaat käytänteet ovat olleet olemassa jossain muodossa jo paljon ennen viime vuosisadan lopun tietoteknistä-, ja1700-luvun länsimaista teollista vallankumousta. Se heijastaa vanhoja mestari - kisälli perinteitä, jossa toimivat tekniikat ja tieto/taito pyrittiin välittämään tuleville sukupolville. Parhaat käytänteet onkin mahdollista nähdä tämän perinteen jatkumona ja sovelluksena ensin teolliseen, ja sitten informaatioteknologiseen aikakauteen siirryttäessä.

Parhaille käytänteille on nähtävästi olemassa tilausta. Siksi ne ovat pysyneet keskuudessamme tähän asti, ja siksi ne tulevat keskuudessamme olemaan vielä pitkään. Tästä syystä niihin onkin syytä allokoida tieteellisiä resursseja jotta niistä saataisiin irti paras hyöty. Parhaat käytänteet ovat ehkä nyt tietotekniikkaan siirryttyään kokeneet jonkinlaista renesanssia, ja niitä kohtaan osoitetaankin tervettä mielenkiintoa niin tutkimuskohteena kuin osana ohjelmistokehitystä. Kozlova & Hasenkamp & Kopanakis (2012) esittävätkin, että siinä missä tietotekniikan ohjelmistokehitys on suuresti hyötynyt konventionaalisen teollisuuden prosesseista ja periaatteissa, on mahdollista että jotkin alat voisivat vastaavasti hyötyä tietotekniikan parissa kehitetyistä parhaista käytänteistä, sillä ne jakavat joitain yhteisiä ydinosaalueita.

Parhaita käytänteitä on mahdollista kerätä ja lajitella, kuten luvussa 3 huomasimme. Niiden toimivuutta on myös mahdollisesti tieteellisesti jossain määrin mitata ja arvioida. Niihin liittyy kuitenkin myös rajoitteita ja heikkouksia, ja puhuttaessa parhaista käytänteistä tulisikin olla varovainen, sillä raja hyvän ja parhaan käytänteen välillä on vaikea vetää. Quayzin (2011) ehdottaakin, että paras käytänne tarkoittaa parhaita hyviä käytänteitä, ja parhaat käytänteet ovat parhaita vain, kunnes löytyy uusi, parempi käytänne. Ne kuitenkin luonteensa puolesta pyrkivät säilyttämään tietoa prosesseista jotka ovat tuottaneet hyviä tuloksia, tai joiden avulla pystytään välttämään joitain sudenkuoppia.

Tietoturvasta puhuttaessa löydetyt parhaat käytänteet olivat toisaalta yleistyksiä ja

universaaleja periaatteita, kuten ”ole nöyrä” tai ”luota vain luotettaviin kanaviin”, ja toisaalta hyvinkin spesifejä, tarjoten jonkin hyväksi havaitun ratkaisuus johonkin nimenomaiseen ongelmaan, kuten ”vältä puskurien ylivuotoja”. Tietoturvan osalta yleinen konsensus oli, että tietoturvaa ei kannata lisätä ohjelmistoihin vasta jälkikäteen, eikä kannata luottaa siihen, että ohjelmistoa käyttää hyväntahtoinen käyttäjä. Sen sijaan jo ohjelmistoja suunnitellessa tulee ottaa huomioon mahdollisimman tarkasti kaikki tavat joilla ohjelmistoa voi käyttää väärin, ja taata, että ohjelmisto pystyy toimimaan turvallisesti vihamielisessäkin ympäristössä. Tietoturva kuuluu sovelluskehityksen elinkaaren kaikkiin vaiheisiin, ja oikeilla parhailla käytänteillä pystytään edesauttamaan turvallisten sovellusten tekemistä.

Kirjallisuutta

Caron, R. 2000. *Coding Techniques and Programming Practices*. Saatavilla WWW-muodossa <URL: https://msdn.microsoft.com/en-us/library/aa260844(v=vs.60).aspx>. Viitattu 3.2.2015.

Goodman, B.D. 2007. & Goldman S.N. *Freeing creativity by understanding the role of best practices*. Engineering Management Conference, 2007, s. 308-311. IEEE International.

Kozlova, A. 2012. & Hasenkamp, U & Kopanakis, E *Use of IT Best Practices for Non-IT Services*. Service Research and Innovation Institute Global Conference, 2012, s. 725-734. IEEE

Kumar, R. 2007. & Pandey, S.K. & Ahson S.I. *Security in Coding Phase of SDLC*. Wireless Communication and Sensor Networks, WCSN '07. Third International Conference on, 2007, s. 118-120. IEEE.

Lunn, P. 2008. *Basic Instincts, Human Nature and the New Economics* Lontoo, Marshal Cavendish, 2008

McGraw, G. 2004. *Software Security* Building Security In, 2004, s. 80-83. IEE Security & Privacy

Oxford dictionary. 2015. *Coding Techniques and Programming Practices*. Saatavilla WWW-muodossa <URL: http://www.oxforddictionaries.com/definition/english/best-practice?c Viitattu 13.2.2015.

Quayzin, X. 2011. *Are best practices really best practice?*. System Safety, 6th IET International Conference on, 2011, s. 1-4. IET.

Razvan, P. 2001. *Best practices for secure development* Lokakuu 2001

Redwine, S.T. Jr. 2004. & Davis, N. *Process to Produce Secure Software* Nation Cyber Security Summit, Maaliskuu 2004.

Wheeler, D.A. 2003. *Secure Programming for Linux and Unix HOWTO* GNU Free Documentation License, Maaliskuu 2003.

Yasar, A. 2008. & Preuveneers, D & Berbers, Y & Bhatti, G *Best practices for software*

security: An overview. Multitopic Conference, INMIC 2008., 2008, s. 169-173. IEEE International.