Laatu ja testaus

Robustiudesta. Testauskirja lapsille. Minä en tee bugiraportteja

2/2020

TestausOSY
Finnish Association of Software Testing

Julkaisija Testauksen osaamisyhteisö TestausOSY

Julkaisupaikka Tampere

Lehden arkisto:

https://www.sytyke.org/testausosy.fi/magazine

Lehti on laadittu vapaaehtoisvoimin.
Artikkelien oikeudet kuuluvat kirjoittajille.
Taitto: Matti Vuori
Kansikuva: Matti Vuori

TestausOSY on Sytykkeen osaamisyhteisö, mutta avoin kaikille

Sisällysluettelo

3. Lukijalle

4. Ajattelumalleja, joilla saadaan robustimpia systeemejä. *Matti Vuori*

11. Lapset oppivat testaamaan ohjelmistoja. *Kari Kakkonen*

19. Minä en tee bugiraportteja. *Anssi Lehtelä*

Lukijalle

Matti Vuori

Hyvää syksyä, edelleen pandemian maailmaan.

Tässä numerossa on nykyään normaaliin tapaan kolme juttua. Moni on varmasti kuullut Kari Kakkosen testauskirjasta lapsille. Kari kertoo artikkelissaan sen synnystä. Anssi Lehtelän aiheena on kommunikointi tuotteen testauksessa tehdyistä monenlaisista vainnoista. Itse kirjoitin systeemien robustiuden taustalla olevista ajattelumalleista.

Lisääkin juttuja lehteen mahtuisi, mutta laatu on tunnetusti aina määrää tärkeämpi asia.

Hyviä lukuhetkiä.

Suomessa 28.10.2020

Ajattelumalleja, joilla saadaan robustimpia systeemejä

Matti Vuori

Me kaikki olemme riippuvaisempia tekniikasta kuin koskaan ja se riippuvaisuus vain lisääntyy kaikilla elämänalueilla. Samaan aikaan sosiotekninen ympäristömme muuttuu monimutkaisemmaksi kaikilla abstraktiotasoilla. Tarvitaan entistä luotettavampaa tekniikkaa, jota uhkaavat entistä useammat ja haastavammat häiriöt, poikkeamat ja odottamattomat tapahtumat. Tarvitaan siis laitteiden, ohjelmistojen, systeemien ja infrastruktuurin vahvaa robustiutta. Kysymys kuuluukin, että miten se saadaan aikaan tai ainakin millaiset ajattelumallit edistävät tätä?

Hyvä tekniikka on robustia, eli kestää luotettavasti millaista käyttöä tahansa ja toimii millaisissa olosuhteissa tahansa. Eli ei häiriinny tapahtuipa käytössä tai ympäristössä melkein mitä tahansa. Ja jos vähän häiriintyykin, osaa toipua siitä tyylikkäästi.

Ja tietenkin tämän pitää onnistua systeemin tai tuotteen koko elinkaaren ajan. Mutta robustiuden haasteet muuttuvat ajan myötä. Hajautetussa ATK:ssa on aina se haaste, että toimivatko kaikki systeemin komponentit kunnolla – tai toimiiko kukin komponentti edes itse. Ja ennen kaikkea se, että käyttäjä tietäisi, että systeemi ei toimi. Ns. kaikille on tuttua se, että pilven levytilapalvelu ei ole päivittänyt tiedostoja, sähköpostiohjelma ei ole hakenut uusia meilejä tai koronatutka ei olekaan toiminut kännykän tausta-ajossa. Ja mikään niistä softista ei kerro tästä, vaan teeskentelee, että kaikki on hyvin.

Tällaisten softien suunnittelijoiden mentaalisista suunnittelumalleista puuttuu selvästi ajatus häiriöiden hallinnasta, systeemin paranoidista epäilystä ja aktiivisesta toiminnan valvonnan tarpeesta. Pohditaanpa asiaa hahmotellen robustiudelle otollista mindsettiä ja käytäntöjä systeemien kehittämisessä. Eli millaiselle ajattelulle olisi tarvetta (toinen haaste on sitten se, miten sitä ajattelua saataisiin synnytettyä).

Ymmärrys ongelmien vaikutuksista

Ketä kiinnostaisi robustius, ellei ole tieto- ja tunnetasolla ymmärrystä siitä, mihin häiriöt ja ongelmat vaikuttavat. Kyse on siis riskien ymmärtämisestä. Liiketoiminnan tasolla on kyse vahinko- ja keskeytysriskeistä, vastuuriskeistä ja muusta. Yksilön tasolla vähän muunlaisista asioista. Tämä ymmärrys kokonaissysteemien roolista liiketoimintojen ja ihmisten maailmoissa ohjaa suunnittelustrategioita ja saa tekemään riskianalyysiä potentiaalisten ongelmien tunnistamisen lähtökohdaksi. Asiakaslähtöinen riskianalyysi on riskienhallinnan lähtökohta.

Konkreettinen, realistinen mentaalimalli ja ymmärrys tekniikasta

Systeemien ja niiden ajoympäristön tekniikassa on aina rajoituksia, variaatioita ja virheitä. Kun sen ymmärtää, voi suunnitella systeemejä, jotka pärjäävät rajoituksissa (esim. nopeus, muisti), variaatioissa (data, ajurit, kirjastot, toiminta) ja virheissä (rikkinäinen data, häiriöherkkä tietoliikenne). Tietoliikenne on erityinen rajoitusten alue. Systeemien kehittäjillä on aina huippunopea verkko, jossa ei ole katkoja, ei viiveitä, ei kytkeytymisongelmia. Käyttäjillä on tilanne aivan toinen.

Kuormitusrealismi on tärkeää. Silloillakin pitää olla varmuuskerroin ja digimaailmassa mitään ei pidä käyttää maksimikapasiteetilla, jos siltä toivotaan kykyä reagoida muutoksiin. Tämä pätee niin tekniikkaan kuin ihmisiinkin. Tietenkään se ei riitä vaikkapa nettisaittien palvelunestohyökkäyksissä, mutta arkipäivän ongelmia on paljon enemmän.

Tarkkuuden tarve on vaarallista. Systeemit hyötyvät aina toleransseista. Jos mekaaninen tekniikka täytyy tehdä kovin pienillä toleransseilla, joskus ne menevät umpeen ja systeemi ei toimi. Tietotekniikassa tiukat aikatoleranssit ja jopa reaaliaikavaatimukset ovat haastavia – aina niitä ei voida saavuttaa.

Systeemit ovat likaisia. Suunnittelijat elävät joskus ideaalisessa maailmassa, jossa kaikki on uutta ja kaikki on tarjolla vain heidän tuotteelleen. Reaalimaailmassa vanhassa tietokoneessa on myös vanha käyttöjärjestelmä, kolmannen osapuolen virustorjunta ja kymmeniä muita ohjelmia, joiden kaikkien kanssa pitäisi toimia hyvässä sovussa. Jopa testaajillekin on perinteisesti annettu tuorein tehomylly, joka vieläpä tyhjennetään aina ennen uusia testauksia.

Ymmärrys, että kaikki muuttuu

Jatkuvan muutoksen hyväksyminen on oleellista. Systeemit muuttuvat jatkuvasti. Jolleivat niiden valmistajat muuta keksi, niin muutetaan edes käyttöliittymää. Silloin pitää joko olla muutosta edellä tai tehdä ratkaisuja, jotka sietävät muutosta. Käyttöliittymäpohjainen ohjelmistorobotiikka on tunnetusti kovin fragiilia ja robustiutta toivova välttää sellaisia ratkaisuja.

Systeemin elementti ei aina kestä edes omia muutoksiaan. Robustiuden isoin uhka voi olla se rikkoutuminen hyvää tarkoittavissa muutoksissa, siis regressioriski. Siksi systeemin pitäminen jatkuvasti hyvässä kunnossa ilman "laatuvelkaa" on olennaista.

Konfigurointi tuottaa muutoksia. Ihmiset tykkäävät säätää asioita. Konfiguroinnilla voidaan sovittaa systeemi ympäristön piirteisiin, mutta myös saada se huonoon kuntoon. Joskus systeemejä pitää voida konfiguroida, joskus on parempi että ei. Joskus se, että asioita ei voi konfiguroida, johtuu siitä, että suunnittelijoilla on väärä käsitys käyttäjien ympäristöistä, niiden homogeenisuudesta, kyvyistä yms.

Muutosta pitää hallita. Muutosten vaikutuksia on analysoitava ennakolta ja selvitettävä muutoksen tekemisen jälkeen empiirisesti. Missä muutos ei tuota hyötyä, sitä ei pidä tehdä.

Uusi tekniikka on riski

Tekniikka kypsyy vähitellen. Uusi tekniikka on aina aluksi raakile ja ennen kuin sillä voidaan oikeasti saavuttaa robustiutta, tarvitaan uusia versioita ja aikaa – kenties vuosia. Uusi tekniikka on valitettavasti myös monimutkaista ja huonosti dokumentoitua, joten vaikka tekniikka sinänsä toimisikin hyvin, sitä ei osata käyttää parhaalla mahdollisella tavalla. Tutoriaalit ja tietokirjatkin esittävät naiiveja suunnittelu- ja ohjelmointimalleja jättäen tuotantokelpoiseen robustiuteen vaadittavat asiat lukijan selvitettäväksi. Ja uuden elinkaari saattaa olla lyhyt. JavaScript-kirjastojakin syntyy varmaan joka tunti useita uusia... Teknologiavalinnat on syytä tehdä hyvin huolella, hypeen uskomatta.

Ongelmia on aina tiedossa

Sopiva vainoharhaisuus tervettä. Isoin osa elementtejä missä tahansa systeemissä on jonkun toisen tekemiä. Niiden toiminnasta ei ole varmuutta ja ne saattavat päivityksissä muuttuakin koska tahansa – vaikkapa viallisiksi tai vihamielisiksi. Siksi niitä pitää epäillä ja varmistaa kaikkien operaatioiden todellinen ongelma ja sivuvaikutukset.

Osa vainoharhaisuutta on systeemin abstraktiotasojenkin erottaminen ja suhtautuminen epäillen kaikkeen, mitä tapahtuu muilla tasoilla (ja toki samallakin) – sama pätee kaikkiin muihinkin systeemin käsitteellisiin jäsennyksiin. Epäily merkitsee mm. sitä, että kaikkien funktiokutsujen onnistuminen tarkistetaan, olipa tilanne miten triviaalilta vaikuttava tahansa.

Ihmiset tekevät paljon virheitä. Hyvä suunnittelu vähentää virheiden määrää, mutta systeemin näkökulmasta pitää lähteä siitä, että ihminen voi tehdä mitä tahansa ja jättää tekemättä mitä tahansa. Silloin pitää havaittavissa olevat virheet käsitellä ja/tai tehdä mahdolliseksi perua virheelliset operaatiot. Työssä on tietysti automatisointi tapa vähentää ihmisen tekemiä virheitä. Ihmiset tekevät myös omituisia asioita. Tuotteen tai systeemin käyttö tavalla tai tarkoitukseen, johon sitä ei ole suunniteltu, on aina asia, johon pitää mahdollisuuksien mukaan varautua. Ja kyberrikollisuus on aina mietittävä asia. Nämä kaikki edellyttävät omanlaistaan riskianalyysiä.

Ymmärrys volyymien variaatiosta – keskiarvo ei ole suunnitteluperusta. Isot systeemien käyttövolyymit kampanjoissa ja kriiseissä tuottavat aina haasteita systeemien toiminnalle, koska niihin ei ole varauduttu. On mietitty vain keskimääräisiä käyttäjämääriä normaalissa arjessa. Samoin voi olla yllätys ihan arkinen tilanne, että sisäisiä tietojärjestelmiä käytetään aina vaikkapa kaikki samaan aikaan päivän päätteeksi.

Meta-epäonnistumisen taso

Onnistuminen ja epäonnistuminen ovat robustiuden yhteydessä tärkeimpiä konseptien suunnittelussa ja valinnassa. Se on taso, jolla tehdään kokeiluja ja jolla ideat saavat epäonnistua, jotta voidaan löytää se vaihtoehto, joka toimii parhaiten käytännössä.

Lokakuussa oi taas kansallinen epäonnistumisen päivä (https://fi.wikipedia.org/wiki/Kansallinen_ep%C3%A4onnistumisen_p%C3%A4iv%C3%A4). Keskustelu aiheesta on vähitellen saanut myös onnistumisia ja ymmärretään yhä useammin ja jäsentyneemmin, että epäonnistuminen ei aina ole kehumisen paikka. Jokainen päivä onkin meta-epännistumisen päivä. Metatasolla reflektoidaan toimintaa ja epäonnistumisia. Sillä tasolla tapahtuu mm. tällaisia:

- Kontekstin ja tilanteen asemointi ja tolkun saaminen. Millainen juttu on kyseessä: ihan rutiini vai hallitsematon kaaos? Onko epäonnistuminen iso riski vaiko vain kokeilun tietoa tuottava tulos?
- Toiminnan suunnittelu, toimintamallien ja tukisysteemien rakentaminen.
- Riskienhallinta – riskien tunnistaminen, seuranta, korjausliikkeet, toipumissuunnittelu.
- Oppimisprosessi – kaikenlaisista onnistumisista ja epäonnistumisista.
- Kulttuurinen prosessi, suhde onnistumiseen, toiminnan laatuun, virheisiin, kehittymiseen ja ihmisiin.

Jämäköissä organisaatiokonteksteissa tämän nimi voisi olla vaikka holistic failure management.

You may run fast, but don't break things

Oma työ vaikuttaa aina muihin ja potentiaalisesti vaarantaa aina kokonaisuuden robustiuden. Jos rajapintoja tai toimintalogiikkaa muutetaan, muu systeemi ei kenties toimikaan kuin pitäisi. Sama pätee tietorakenteisiin, tietomalleihin, datatyyppeihin ja muihin. Yhden tietorakenne on usein toisen API. Jo vikojen korjaaminen on riskialtista, koska koko muu maailma saattaa olla rakennettu oletukselle niistä vioista.

Niinpä robustius ei olekaan systeemin elementin ominaisuus, vaan sen suhteen toisiin ominaisuus, joskus vuorovaikutuksen ominaisuus ja kokonaissysteemin ominaisuus.

Joskus testauksenkin tärkein tavoite ei ole löytää vikoja niiden korjausta varten, vaan ymmärtää systeemin käyttäytyminen kaikissa tilanteissa.

Esteettinen silmä voi katsoa väärin

Klassinen robustiuden estetiikka on peräisin maataloudesta – kirves on hyvä kapine ja sen päätä ei juuri saa rikki. Sillä silmällä katsottuna on monoliittinen systeemi viehättävät. Mutta silloin silmä voi nähdä väärin. Oleellista on miten monta ongelmavektoria kohdistuu systeemin elementtiin. Silloin on yhden palvelimen iso tietojärjestelmä huono ajatus. Kun se palvelin on missä tahansa ongelmissa, mikään ei toimi. Mutta kun systeemi jaetaan useille palvelimille ja hajautetaan, yksi palvelin on vähemmissä haasteissa ja jos se on poissa pelistä, kokonaisuus toimii vielä. Geneettinen mentaalinen malli pitää silloin kiven ja kirveen sijaan etsiä sienirihmastosta...

Oleellista on tekniikan luontainen robustius – sellaisen perustekniikan löytäminen, joka sopii kohteeseen parhaiten.

Matalan tason ATK-asioissa yksinkertaisessa lähdekoodissa algoritmit näkyvät kirkkaina, elegantisti, mutta jokainen parametrien ja funktiokutsujen tarkistus lisää monimutkaisuutta, tuo "kohinaa" koodin luettavuuteen ja ei ole "kaunista", mutta sen sijaan lisää varmistamisen estetiikkaa.

Ihmisten maailmassa estetiikka on joskus robustiuden esteenä. Silloin, kun muotoiluidea perustuu minimalismiin, voivat hallintalaitteet olla hyvän kokoisia, käyttäjän mentaalimallin mukaisia jne., mutta jos esteettinen idea lähtee määrästä ja siihen liitetään vähävärinen painonappien merkkaus, on tulos virhealtis. Niinpä "senioripuhelinten" viehättävyys on nuorelle teknologiajohtajalle usein vähäinen.

Sosiotekninen systeeminen näkemys

Kun ihmiset toimivat tekniikan kanssa yhdessä, ei robustius synny tarkastelemalla kumpaakin elementtiä erikseen, vaan pitää katsoa kokonaisuutta, vuorovaikutusta. Sosioteknisyyttä on tietysti monella tasolla laitteiden käytöstä isojen systeemien hallintaan. Kaikilla tasoilla on tärkeää

ymmärtää ihmisen ja tekniikan oikea työnjako. Jos ihminen tekee koneen töitä, se on tunnetusti usein virhealtista. Joskus automaatio jääkin puolitiehen ja lopputulos on surkea.

Uusi tekniikka on usein avuksi. Älykkäät toimilaitteet, älykkäät turvasysteemit ja ihmiselle tarjottava tukiäly voivat olla tärkeä etu monessa tilanteessa.

Saavutettavuuskin on robustiutta tuottava osa-alue, sillä systeemin pitää toimia luotettavasti kaikenlaisten ihmisten käyttämänä. Tietojärjestelmillä taas ihmisen rooli on edelleen seurata systeemin toimintaa ja reagoida tarvittaessa havaintoihin. Silloin on systeemin havainnoitavuus oleellista ja kokonaisuuden robustiuden tuottajia ovat mittarit, dashboardit, hälytykset ja operaattorin käytössä olevat välineet – vaikkapa kyberhyökkäyksen aikana.

Systeemien systeemejä

Aina on läsnä useita systeemejä. Kokonaisjärjestelmä on aina systeemien systeemi. Siinä on ydinsysteemi, jota hallitaan toisella systeemillä, ja joista kumpikin on suhteessa useisiin muihin systeemeihin. Kokonaisvaltainen systeemien tunnistaminen ja huomioon ottaminen on tärkeä ajattelumalli.

Ratkaisujen löytäminen ja arviointi tarvitsee analyyttisyyttä

Muotoilusilmä on tarpeen. Robustiutta arvostava ymmärtää yksinkertaisuutta ja ymmärrettävyyttä. Jos arkkitehtuurikaaviota ei ymmärrä helposti, ei siitä synny eikä se ole hyvin toimiva. Mutta yksinkertaisessa pitää olla mukana kaikki tarvittava: tarvittavat varmistukset, monitorointi ja kaikki muu.

Suunnitelmien analyyttinen arviointi on usein välttämätöntä. Ongelmia voidaan tiedostaa kokemuspohjaisesti, mutta se ei riitä. Ammattilaiseen ajatteluun kuuluu luotettavuusanalyysien tekeminen tyyliin vika- ja vaikutusanalyysi, jossa mietitään miten jokin systeemin komponentti voi vikaantua ja mitä siitä seuraa. Siitä alkaa sitten kokeellinen osuus, jossa robustius todennetaan testaamalla. Robustiutta hakeva mieli pohtiikin heti, että mitenkäs tätä voisi koetella ja mielellään heti ja helposti.

Determinismi on hyvä asia, koska siinä ei ole sattumia. Sattumiin liittyy aina yllätyksiä ja niitä on vaikea testata. Sama pätee rinnakkaisuuteen, reaaliaikaisuuteen jne. Mitä vähemmän synkronoitavaa, mahdollisuutta että järjestykset menevät solmuun tai jonnekin kohdistuu samaan aikaan operaatioita, sen parempi. Mutta kaikella on puolensa ja puolensa. Synkroniset toiminnot voivat puolestaan jumittaa muutakin kuin itsensä, kun asiat eivät suju. Koneoppivat systeemit eivät siksi ole kivoja, että niissä on kyse todennäköisyyksistä, jotka vielä muuttuvat oppimisen myötä.

Mutta determinismi on joskus harhaa. Luotettavuusanalyysit perustuvat usein deterministiseen ajatteluun tapahtumaketjuista, mutta systeemit ovat täynnä asioita, joita ei ymmärretä tai odoteta ja jotka tapahtuvat satunnaisesti. Niinpä systeemit pitää testattaessa altistaa satunnaisuudelle joko reaalimaailman kaltaisissa oloissa tai satunnaisuutta synnyttäen.

Mainitut koneoppivat systeemit ovat myös siksi haastavia, että niitä on tapana opettaa vähän epätasapainoisesti, vinoutuen kehittäjien lähimaailman piirteiden mukaisesti. Siksi niillä voi olla haasteita muissa ympäristöissä.

Aika on ystävä ja vihollinen. Aika muuttaa asioita, tuottaa uusia ilmiöitä ja paljastaa olemassa

Kivi on varsin robusti esine.

olevaa. Siksi systeemien arviointi testauksessakin tarvitsee myös aikaa, mutta koska sitä ei ole, sitä pitää nopeuttaa sopivilla testausjärjestelyillä (esim. nopeuttamalla systeemin tapahtumia, simuloimalla aikariippuvia asioita pitkälle tulevaisuuteen).

Käytäntö on eri asia kuin abstrakti kuvaus

Abstraktissa tekniikassa kaikki sujuu aina hienosti. Käytännössä prosessit jumittuvat, dataliikenne hidastuu, on muistivuotoja. Siksi pitää valvoa mitä systeemissä tapahtuu. Tämä on tietojärjestelmien hostauksessa tuttua, mutta harvinaisempaa vaikka kännykkäpohjaisissa systeemeissä.

"Mitä jos?" Systeemien elinkaaressa tapahtuu monenlaista ja joskus systeemit vikaantuvat. Systeeminen robustius edellyttää miettimistä mitä silloin tehdään? Voiko vikaantumisen nopeasti korjata, vaihtaa elementin toiseen, rajata ongelman erilleen muusta systeemistä jne... Pitääkö elementti tai koko systeemi jopa kahdentaa? Tässä korostuu hyvän arkkitehtuurin merkitys. Ja varasuunnitelmat ovat tärkeitä.

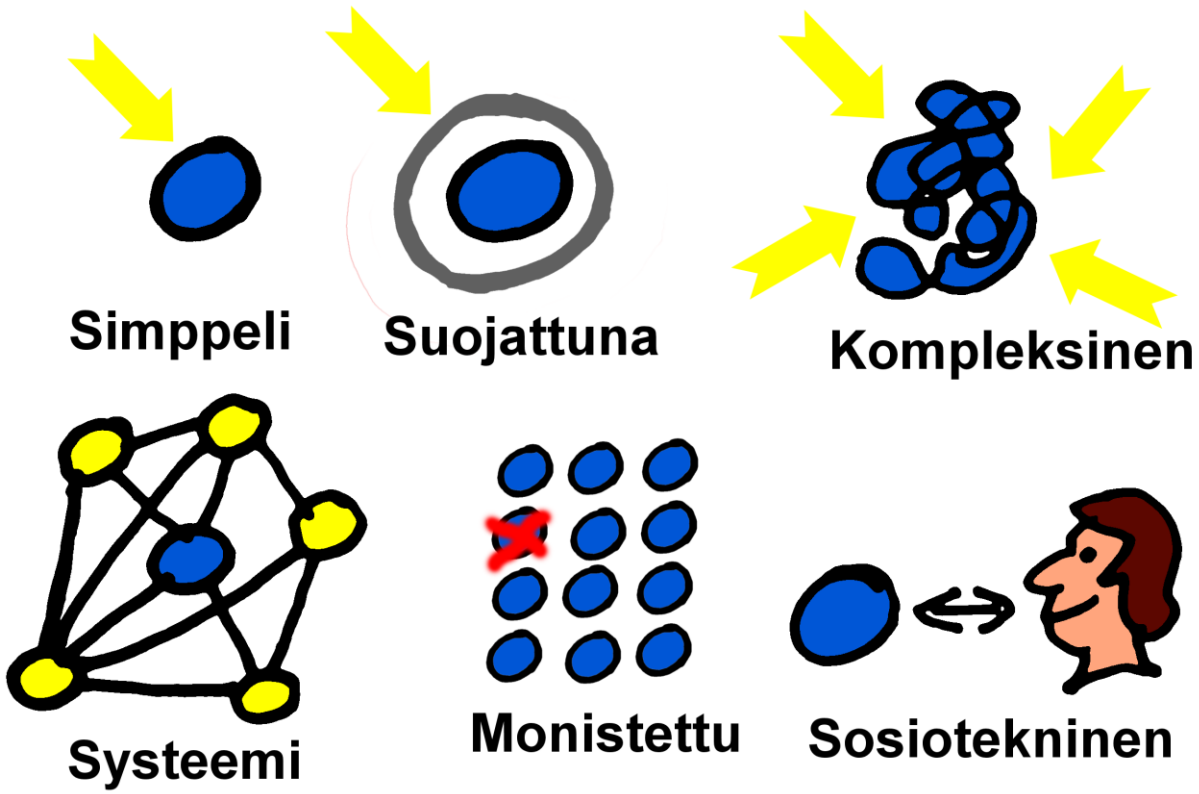
Aiemmin on viitattu luotettavuusanalyyseihin. Arkkitehtuurin arviointi on myös elinkaarta käsittelevä luotettavuusanalyysi, koska siinä yleensä tarkastellaan erilaisia systeemiin vaikuttavia skenaarioita – käyttäjien lisääntyminen, systeemin elementtien vaihtaminen, uusien ominaisuuksien kehittäminen jne... (ja myös vikaantumisia ellei niitä ole analysoitu erikseen).

Testauksen pitää olla monipuolista

Testaus on oppimiskokemus. Systeemien testaus nähdään usein välttämättömänä pahana, jonka keskeisin tehtävä on olla häiritsemättä softan tuotannon työnkulkuja. Silloin testauskin on joko karkeaa tai tai pieniin asioihin lokalisoitua ja ei oikeasti koettele systeemejä. Tässä ajattelussa unohtuu testauksen simulointinäkökulma: laitetaan systeemi pyörimään hallitusti, mutta sitä erilaisilla asioilla tiukasti koetellen. Näin opitaan miten systeemi toimii oikeasti silloinkin, kun olosuhteet eivät ole ideaaliset.

Testauksen robustiuden arviointiin keskittyviä keinoja ovatkin mm.:

- Tutkiva testaus, jossa luova mieli koettelee systeemiä.
- Mallipohjainen testaus, jossa käydään läpi kaikki systeemin tilat satunnaisestikin. Mutta yhtä tärkeää on mallintaa ja simuloida systeemin ympäristöä, jotta voidaan selvittää, miten



Systeemityyppejä

systeemi käyttäytyy suhteessa ympäristön erilaisiin tiloihin ja tapahtumiin.
- Fuzz-testaus, jossa kokeillaan, miten systeemi pärjää rikkinäisenkin datan kanssa.
- Kaikki "negatiivinen" testaus eri abstraktiotasoilla.
- Riskiperusteinen testaus, jossa riskiavaruuden syötteenä on luotettavuusanalyysi.
- Yhteensopivuus ja yhteentoimivuustestaus.
- Testaus erilaisissa ympäristöissä ja niiden konfiguraatioissa (emulaattoritestauksessa ja virtuaalikoneilla voi testiympäristön konfiguraatioita käydä näppärästi läpi ison määrän).
- Käytettävyystestaus.

Hetkinen, piti olla puhe ajattelumalleista ja nyt luetellaan testaustyyppejä! Testauksessa on olennaista tunnistaa softan tyyppi ja siitä kumpuavat vaatimukset ja nähdä heti suunnilleen tarvittava paletti. Sitä alkuarvausta on vaikea muuttaa. Mutta kuitenkin: ketterään maailmaan kuuluu se, että vaikka alustava testaussuunnitelma sanoisi mitä tahansa, ajatuksia sopii säätää, jos uudet asiat vain sopivat projektiin – eli maksajalle eli asiakkaalle.

Vain amatöörit uskovat hopealuoteihin

Lean-ajattelu kai nyt ainakin on varmasti hopealuoti robustiuteenkin? Leanilla on monta olemusta ja kaikki Leaniksi mainittu ei ole Leania nähnytkään. Yksi Leanin kulmakivi on asioiden syvällinen oppiminen yhdessä ja ongelmien ratkaisun systematiikka. Asiakastarpeiden osalta Leanin idea näkyy QFD:ssä (Quality Function Deployment), jossa jäsennetään asiakkaan arvoon johtavia tuoteominaisuuksia, niihin vaikuttavia tekijöitä ja näin ohjataan tuotesuunnittelua. Tuotteen robustius on silloin tasapainoisessa asemassa kaiken muun kanssa ja loppu on kiinni on vain tuotekehityksen osaamisesta.

DevOpsin Ops-puolella ovat robustius-asiat tutumpia ja pitääkin toivoa, että DevOps-muodin levitessä Ops-osaston järjestelmänhallinnan ajattelumallit saavat tilaa ja näkyvät sekä suunnitteluratkaisuissa että niiden muun elinkaaren aikaisissa operaatioissa, eikä DevOps redusoidu kehittäjien lisätöihin. Jo kehittäjien tuominen tiiviisti palvelujen jakelun yhteyteen edistää heidän omaa ymmärrystään systeemien haasteista.

Ketterän kehittämisen pitäisi olla realismipohjaista, mutta käytännössä se on usein ollut "koodauspohjaista" ja sitä on moitittu puutteista arkkitehtuurisuunnittelussa ja vuorovaikutussuunnittelussa. Hyvin sovellettuna ketterä kehittäminen on robustiuden etu, kun systeemejä kehitetään rinnan niiden ymmärtämisen kanssa, empiriaa hyödyntäen.

Tekoälyn yksi ideaali on systeemien tekeminen luotettavammiksi, koska koneoppimiseen perustuvat sellaiset ovat hyviä näkemään kaavoja asioissa ja voivat siksi joustavasti säätyä olosuhteiden mukaan ja ennakoida tulevia haasteita. Haasteena tässä on tietysti se, että miten systeemi pärjää aivan uudenlaisten asioiden kanssa. Yksiteknologisuus on aina innostava asia, mutta käytännössä tarvitaan ratkaisujen diversiteettiä samassakin kohteessa ja asiassa.

Yhteenvetoa ajattelumalleista

Tärkein ajattelumalli on riskien miettiminen, sillä siitä juuri on kyse robustiuden relevanttiudessa. Sitä ei mietitä huvin vuoksi. Riskitietoisuus antaa orientaation ja mahdollistaa teot riskien tunnistamiseksi ja hallitsemiseksi.

Toiseksi olennaisin ajattelumalli on uuden systeemin kohtaaminen avoimin silmin ja sen todellisen luonteen, piirteiden ja haasteiden kokeminen. Ihmisillä on aina kokemuksista kumpuavia mentaalimalleja, jotka muovaavat ajatusta siitä millainen on "mobiilisovellus" tai "tietojärjestelmä". Niitä mentaalimalleja pitää haastaa itse ja yhdessä, koska ne voivat mäpätä uuden kohteen ihan väärään malliin tai ne voivat olla vanhentuneita: robustiuden uhat muuttuvat jatkuvasti ja samoin vaatimukset robustiudelle.

Kyse on aina myös systeemiajattelusta, sillä systeemien elementit eivät koskaan elä eristyksissä ja juuri vasta vuorovaikutuksessa robustius on relevantti käsite. Ja se vuorovaikutus alkaa jo systeemien konseptoinnissa, suunnittelijoiden kesken. Ja systeemi on kaiken päälle aina jossain määrin sosiotekninen.

Robustiuden taustalla on vaihtoehtoisista ammatillisista etiikoista eniten insinöörin etiikka, koska työ tapahtuu tietyn konseptin puitteissa, teknisellä tasolla, tinkimättömästi – tai ainakin tinkien mahdollisimman vähän. Heikolla luotettavuudella

ei haeta kaupallisia pikavoittoja. (Tietysti robusti estetiikkakin on ihan relevantti käsite, mutta sillä saralla robustius liittyy enemmän ilmeen teknisiin toteutuksiin.) Insinöörin etiikan julistuksissa on aina ykkösasiana vastuullisuus tekniikan hyödyntäjiä kohtaan, eli siinä ei ole tekniikka itseisarvo. Jos jollekin on, hän ei ole oikea insinööri.

Tämä on myös robustia ajattelua, joka ei hätkähdä pienistä, vaan osaa jäsentää jokaisen uudenkin tilanteen piirteet hahmottaen konseptin ja sen olennaisuudet, potentiaaliset ongelmat, suunnitelmien heikot lenkit, ja tavat parantaa ratkaisuja.

Edellä on esitelty joitakin robustiuteen liittyvän mindsetin elementtejä. Olennaista tässä on niiden kokoaminen yhteen holistiseksi kokonaisuudeksi, jossa ovat tasapainoisesti mukana systeemin ja sen ympäristön (kokonaisjärjestelmän) ymmärtäminen, hyvä suunnittelu, systeemin arviointi ja testaus ja oppiminen. Tämä kaikki on mielellään työpaikan ja alan kulttuurin piirre ja sitä esiintyykin monessa turvallisuuskriittisten systeemien kehittämisen organisaatiossa. Mutta muualla ovat painopisteet muualla: ohjelmoinnissa, kehittämisvälineissä, automaatiossa.

Matti Vuori katselee robustiutta laatuasioiden ja riskienhallinnan asiantuntijan silmälaseilla ja yrittää suhteuttaa filosofiaa käytännön maailman haasteisiin..Mitä muuta hän on kirjoittanut? Tsekkaa https://www.mattivuori.net/julkaisuluettelo

Lapset oppivat testaamaan ohjelmistoja

Kari Kakkonen

Ohjelmistotestausta lapsille

Yhteiskunta tarvitsee tietotekniikan ja testauksen taitoja. Sekä ohjelmointi- että testaustaitojen puute on jatkuvasti läsnä työmarkkinoilla. Puute pahenee, kun yhteiskunnassamme on yhä enemmän ohjelmistoja. Tarvitsemme yhä enemmän ohjelmoijia ja testaajia. Ohjelmistoja on testattava!

Suosittu ratkaisu tietotekniikan tulevaisuuden tai jopa kaikkien kansalaisten tietokonetaitojen turvaamiseksi on sisällyttää koodausopintoja koulujen opetukseen, jotta lapset voivat oppia ohjelmistokehitystä jo varhaisessa iässä. Tästä lähestymistavasta puuttuu olennainen osa, jonka ohjelmistotestaajat heti huomaavat – uusien, innostuneiden kehittäjien tekemien tuotteiden testaaminen. Painajaisskenaariossa meillä on maailma täynnä surkeaa koodia, ja yritämme löytää kuiville huonon koodin suosta.

Uskon, että meidän on kerrottava lapsille ohjelmistojen testauksen upeudesta. Testaustietojen oppiminen fantasiatarinoiden kautta on tapa, joka sopii suurimmalle osalle lapsia. Joten tavoitteeni on julkaista kirja lohikäärmeistä, ritareista ja ohjelmistojen testauksesta lapsille. Ohjelmistojen testaamisen opettamisella lapsille on pitkäaikainen positiivinen vaikutus testaustaitojen saatavuuteen.

Kirja “Dragons Out” on niin mielenkiintoinen fantasiaseikkailu, että on mahdotonta lopettaa sen lukeminen kesken. Kymmenen vuotta vanhat pilottilukijani ovat kannustaneet minua julkaisemaan kirjan. He sanovat tarinan innostavan ja haluaisivat lukea lisää samanlaista tekstiä. Ohjelmistojen testaus ymmärretään ja se kiinnostaa luonnollisena sivutuotteena kirjan tarinalle. Aikuiset pilottilukijani eri taustoilla (opettajat, ohjelmistojen testaajat, muut taustat) myös ylistävät

ainutlaatuista ja jännittävää lähestymistapaa, joka tulee varmasti olemaan menestys. Heidän mielestään tarina on hyvä, ja ohjelmistojen testaus selitetään hyvin selvästi. Heillä on taipumus nauraa joillekin kirjan analogioille. Hauskaa!

Mitä enemmän tiedät ohjelmistokehityksestä, sitä enemmän voit arvostaa sitä, kuinka tietotekniikkakonseptit muuttuvat fantasiaksi. Olen käyttänyt tarinoissa analogialähestymistapaa, jota olen harjoittanut blogeissani vuosien varrella. Lukijat voivat palata kirjaan yhä uudelleen ymmärtääkseen ohjelmistojen testausta. Esimerkit ovat yleensäkin yksi parhaista tavoista oppia. Kirja on tapa oppia ohjelmistojen testausta tarinankerronnan kautta kaiken ikäisille lukijoille.

Miten lapset oppivat?

Lapset oppivat monin tavoin, kuten aikuiset [1]. Sanotaan, että mitä enemmän oppimisstrategioita lapset käyttävät, sitä parempia he voivat olla ongelmanratkaisussa, lukemisessa, tekstin ymmärtämi-

[1] https://raisingchildren.net.au/school-age/school-learning/learning-ideas/learning-school-years

sessä ja muistamisessa. Olen keskustellut lukemattomien opettajien, testaajien ja lasten kanssa ymmärtääkseni, miten lapset oppivat, voidakseni opettaa ohjelmistojen testausta lapsille. Kerron tässä, mitä olen oppinut.

Tarinat ja esimerkit toimivat yleensä paremmin kuin luetteloiden, termien tai vaikeiden käsitteiden ulkoa muistaminen. Jos pystyt muuttamaan ideasi tarinaksi tai analogiaksi, tee se. Silloin on helpompaa oppia. Jotkut sanovat, että tarinankerronta on vanhin kaikista oppimistekniikoista. Sitä käytettiin jo ennen kuin ihmiset oppivat kirjoittamaan. Tarinankerronta tarjoaa yleensä mahdollisuuden käyttää samalla toista tehokasta oppimistekniikkaa: samaistuminen tarinan hahmoihin. Lapsi voi astua tarinan sankarin kenkiin. Sitten lapsi voi alkaa jäljitellä tarinan sankarilta opittua käyttäytymistä. Olet varmasti huomannut lasten tekevän näin, esim. nähtyään elokuvan. Oma poikani menee aina huoneeseensa elokuvan jälkeen ja alkaa varioida elokuvan tarinaa lego-palikoillaan. Kiehtovaa!

Myös runot ja musiikki ovat tehokkaita oppimiskeinoja. Ne ovat eräänlaista tarinankerrontaa, vain erilaisessa muodossa. Sopivan riimin avulla jonkin asian muistaminen on paljon helpompaa. Vaikka jonkin asian muistaminen ei ole vielä tarkoita tuon asian osaamista, se on oppimisen edellytys. Täytyy muistaa ja toistaa, ja lopulta oppii asian kokonaan. Viime vuosisadalla opettajat ovat korostaneet muistamisen tärkeyttä liikaakin. Silti muistamisella on arvo. Monet ovat kuulleet sanonnan, että "kertaus on opintojen äiti".

Pelaaminen on hyvä tapa kokeilla uusia asioita käytännössä. Kun asetat joitain tavoitteita pelitavoitteiksi ja yrität saavuttaa nämä tavoitteet pelaamalla peliä, opit kuin itsestään. Tämä on pelillistämistä. Abstraktimmalla tasolla voimme puhua tutkimisesta: tutkitaan, kuinka jokin toimii, yritetään tehdä jotain. Tutkiminen on asioiden tekemistä kokeilemalla, oppimalla, miten asiat toimivat ja miten ne eivät toimi. Testaus on oikeastaan myös tutkimista – oppimista tekemällä.

Sitten on yksinkertaistaminen ja selkeys. Vaikka nämä eivät ole omia oppimisstrategioitaan, ne ovat tärkeä osa oppimista. Jos jotain voidaan selittää mahdollisimman yksinkertaisesti ja selvästi, oppiminen voi tapahtua.

Lapset oppivat myös turvallisten rajojen kautta. Pitää kertoa, mikä on oikein ja mikä väärin. Sopivissa rajoissa lapsi voi oppia mitä tahansa turvallisesti, kokeilla eri oppimisstrategioita, etenkin tutkia, kokeilla ja erehtyä. Rajojen asettaminen on olennainen osa lapsen oppimisstrategiaa.

Olen yrittänyt hyödyntää kaikkia näitä oppimistyylejä kirjassani. Yksi asia on kuitenkin tärkein – tarinankerronta. Kaiken lisäksi tarinankerronta on hyödyllistä erityyppisille oppijoille. Jotkut ihmiset oppivat silmien kautta, toiset korvien, toiset liikkeen kautta. Kaikki nämä ihmiset voivat oppia tarinoiden avulla. Visuaaliset oppijat pitävät mielikuvista, joita he saavat tarinankerronnasta. Auditiiviset oppijat yhdistävät sanat ja kertojan äänen. Kinesteettiset oppijat voivat tarttua tarinan emotionaalisiin yhteyksiin ja tunteisiin.[2]

Sitten on kysymys siitä, missä iässä lapset oppivat parhaiten ohjelmistojen testauksen kaltaisia käsitteitä. Milloin he ovat valmiita ottamaan vastaan sellaisia ideoita? On tutkittu, että lapset alkavat oppia abstraktimpia käsitteitä 11 tai 12 vuoden iässä.[3] Siinä iässä aletaan ajatella filosofiaa ja elämän tarkoitusta. Itse ajattelun ymmärtäminen on yksi merkittävimmistä taidoista. Se on melko abstraktia, ja arvaan, että jotkut ihmiset eivät

[2] https://www.harvardbusiness.org/what-makes-storytelling-so-effective-for-learning
[3] https://www.scholastic.com/parents/family-life/creativity-and-critical-thinking/development-milestones/cognitive-development-11-13-year-olds.html

koskaan ymmärrä ajattelun olemusta, vaikka yrittävät. Tarkoituksen jatkuva etsiminen onkin itse asiassa yksi elämän suurimmista asioista. Jos taas ajatellaan ohjelmistojen testausta, huomataan, että siinäkin on kyse ajattelusta. Mietimme, mitä tiedämme ohjelmistoista ja mitä emme. Mietimme, mitä tietoa etsiä ohjelmiston toiminnasta seuraavaksi. Päättelen siis, että testaus opitaan parhaiten noin 11-12-vuotiaana. Tietenkin jokainen on erilainen ja voi oppia asioita omaan tahtiinsa. Voi oppia lisää koko elämän ajan. Mutta milloin aloittaa testaamisen oppiminen? Sanoisin, että 11 vuotta on hyvä ikä.

Kirjaprojekti

Minulla oli ilo käydä James Whittakerin Tarinankerronta ja Luovuus -kurssi muutama vuosi sitten. Hänen tarinankerronta-manifestinsa on erinomainen: "Jokaisella ihmisellä on tarina. Jokainen asia tarvitsee tarinankertojan. Opi olemaan tarinankertoja, koska ellet ole mukana tosi-tv:ssä, kukaan muu ei kerro tarinaasi puolestasi. Joten kerro tarina. Kerro hyvä tarina. Ja olkoon tuo hyvä tarina yksi osa tarinoiden sinfoniaa, joka tekee tästä maailmasta paremman. " Olen samaa mieltä. Minulla oli jo kipinä kirjoittaa kirja jossain vaiheessa, mutta Jamesin tarinankerronta-ajatusten kuuleminen muutti kipinän kirkkaaksi liekiksi. Mutta minkä tyyppinen kirja minun tulisi kirjoittaa? Mikä on tarinani? Haluan kertoa tarinan.

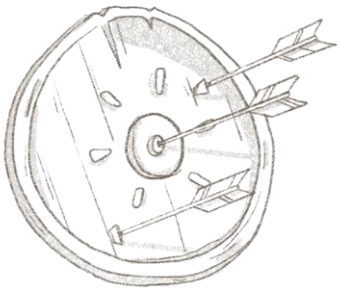
Vuonna 2014 suomalainen kirjailija Linda Liukas julkaisi "Hello Ruby" -kirjan, joka on kirja ohjelmoinnista noin esikouluikäisille. Se kertoo logiikasta, joka on kaiken ohjelmoinnin ydin. Kirjan julkistus sai minut ajattelemaan. Pitäisi olla myös ohjelmistojen testauskirja lapsille. Se voisi olla minun näkökulmani.

Kirjoitin blogeja ohjelmistojen testauksen aiheista, erityisesti työnantajani Knowitin blogissa[4] ja Suomen merkittävimmän IT-lehden Tivin blogissa[5]. Kirjoittaminen toi minulle sujuvuutta kertoa ohjelmistojen testauksesta erilaisten analogioiden kautta. Huomasin, että olen hyvä luomaan analogioita.

Sitten sain tilaisuuden tehdä unelmastani totta. Perheemme päätti ottaa vuoden vapaata töistä ja viettää aikaa Lapissa, nauttia pohjoisen kahdeksasta vuodenajasta, hengittää, unelmoida, ajatella tulevaisuutta ja viettää aikaa perheenä. Asetimme monia tavoitteita tälle vuodelle. Yksi tavoitteistani oli kirjoittaa kirja.

Joten vietin osan sapattivuodestamme 2018-2019 kirjoittamalla kirjan ohjelmistojen testauksesta lapsille. Alku ei ollut helppo. Puoli vuotta oli kulunut ennen kuin päätin, että minun on todella aloitettava kirjan kirjoittaminen. Sitten eräänä pimeänä joulukuun aamuna avasin tietokoneeni ja aloin kirjoittaa.

Kirjoitin ensimmäisen luvun useita kertoja eri näkökulmista ja pyysin palautetta muutamalta ihmiseltä. Kaikki tekstiversiot sisälsivät lohikäärmeitä, ja pian olikin selvää, että tarinan lohikäärmeet edustavat vikoja/virheitä ohjelmistossa. Minulla voisi olla jokaisessa kirjan luvussa aina erityyppinen vika tai vian vakavuusaste. Lohikäärme-analogia oli hyvä ja se antoi samalla kirjalle rakenteen.



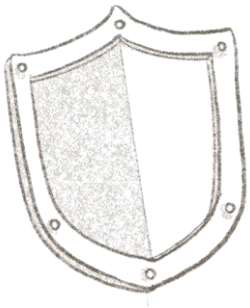
[4] https://we.knowit.fi/

[5] https://www.tivi.fi/blogit/parisuhdetta-ja-ohjelmistoja-ei-pida-testata-vaarin-voi-kayda-huonosti/1acaea00-0163-41cc-a18c-b050a59fe38e

Sitten täytyi päättää, mikä edustaa kirjassa ohjelmistoa. Ensimmäisessä luvussa aloitin lohikäärmeellä, joka ahdisti kylää. Joten kylä, kaikkine ympäristöineen, oli ohjelmisto. Lisäsin myöhempiin lukuihin analogioita ohjelmiston rakentamisesta, ja silloin ohjelmisto tarkentui tarkoittamaan kylän puolustuksia. Loppujen lopuksi käytin erilaisia esimerkkejä siitä, millä tavalla kylä on kulloinkin vaikeuksissa lohikäärmeiden takia. Vastaavasti ohjelmistoja on eri muotoisia ja kokoisia, ja vikoja on isoja ja pieniä.

Ohjelmistojen testausta kuvasi luonnollisesti lohikäärmeen löytäminen, ja vian poistaminen oli lohikäärmeiden tappamista tai voittamista. Tämä sopi hyvin tietotekniikan maailmaan, jossa kaikenlaiset ihmiset voivat löytää vikoja, mutta vian poistaminen vaatii kehitystiimin. Joten kyläläiset, mukaan lukien lapset, löysivät lohikäärmeitä tai ritarit löysivät ja karkottivat lohikäärmeitä. Lasten keskeinen rooli tekisi tarinasta houkuttelevamman nuorille lukijoille – he voisivat samaistua tarinan lapsiin.

Joten, luku toisensa jälkeen, lisäsin erilaisia esi-



merkkejä ohjelmistojen testauksesta. Joissakin tapauksissa tekninen tuki pystyi poistamaan viat, eli tarinassa panin kylien metsästäjät yrittämään lohikäärmeiden karkottamista. Jokainen ohjelmisto ohjelmoidaan ensimmäisen kerran ja uudella ohjelmistolla olisi kaikenlaisia pieniä vikoja. Suurin osa ohjelmistoista päivitettäisiin myöhemmin, ja tämä tarkoittaa erityyppisiä vikoja. Vastaavasti erilaiset kylät eroavat toisistaan. Tietotekniikassa tarvitaan myös erilaisia asiantuntijoita, joten tietäjät istuisivat tarinaan hyvin.

Kävin läpi tätä luetteloani eri näkökulmista ohjelmistotestaukseen, ja ruksin niitä hoidetuksi. Excel-luettelon käytöstä oli paljon apua. Aikajanan käyttäminen auttoi pitämään tarinan yhtenäisenä, koska toisinaan oli tarpeen kertoa tarinaa menneistä ajoista (tarina tarinan sisällä), selittää kuinka kylä rakennettiin ensimmäisen kerran – miten ohjelmisto rakennettiin ensimmäistä kertaa. Kartan käyttö oli myös välttämätöntä, koska fantasiamaailmassani täytyi olla monia kyliä edustamaan erityyppisiä ohjelmistoja, ja toisaalta oli tarpeen käyttää enimmäkseen samoja hahmoja, jotta lukijat samaistuisivat paremmin heihin. Kaiken lisäksi, kaikki fantasiatarinat tarvitsevat kartan, eikö? Muistan, että tutkin Taru Sormusten Herrasta -kirjan karttaa hyvin innostuneesti, kun olin noin 11-vuotias.

Useamman luvun kirjoittamisen jälkeen lähetin aina uudet luvut pilottiluettavaksi uusille ihmisille. Kyselin jatkuvasti, olisiko jollakulla tuttavistani lapsia, jotka voisivat lukea mielenkiintoisen tarinan ja antaa palautetta. Otin yhteyttä kolmen eri taustan aikuisiin, joiden palautetta halusin: opettajiin, testaajiin ja ihmisiin, jotka eivät tiedä mitään testaamisesta tai opettamisesta. Pilottilukijoiden palaute oli erinomaista. Kaikki kannustivat kirjan ideaa paljon. Jotkut sanoivat, että tarina oli hieno, ja he haluaisivat lukea lisää samanlaista. Jotkut sanoivat, että testauskäsitteitä on vähän vaikea ymmärtää. Jotkut kommentoivat lauseiden sanamuotoa ja selkeyttä, jotkut ehdottivat uusia näkökulmia testauksesta kertomiseen. Kaikki tuon puolen vuoden aikana, siis suomenkielisen version kirjoittamisen aikana, annettu palaute oli arvokasta, mukaan lukien testauskollegoideni katselmointi. Minulla oli nyt ehjä kirja, mutta silti pyysin jatkuvasti lisää palautetta, koska vielä olisi aikaa ennen kuin minun tarvitsisi antaa käsikirjoitus kustantajalle.

Haluaisin mainita hahmojen tasa-arvoisuuden. Halusin kirjoittaa hyvin tasa-arvoisesti puhuen tytöistä ja pojista, naisista ja miehistä, isoissa ja pienissä rooleissa. Tarkistin, että minulla on naisia kaikenlaisissa rooleissa, myös valta-asemissa.

Näihin aikoihin etsin myös kuvittajaa kirjalleni. Löysin Adrienn Széllin, ja luonnostelimme erilaisia lohikäärmeitä ja muita hahmoja. Löysimme tyylin, jonka olin unelmoinut ja suunnitellut. Se oli jotain hieman sarjakuvamaista, jotain tarpeeksi monipuolista mihin vain tulevaisuuden tarpeeseen, jota saattaisin keksiä kirjan hahmoille.

Kuvitus loi kaksi haastetta. Ensinnäkin minun pitäisi rahoittaa kuvitus. Toisekseen, minun olisi ratkaistava, miten organisoida tekijänoikeudet, koska kuvittelin jo kokonaisen maailman muita oppimistuotteita kirjan hahmojen ympärille. Ensimmäisen haasteen pystyisin hoitamaan omilla rahoillani, mutta vain jossain määrin. Joten päätin aloittaa joukkorahoituskampanjan. Toisen haasteen ratkaisin perustamalla yrityksen ja neuvottelemalla kuvittajani kanssa kuvitusoikeuksista. Yhtäkkiä minulla oli paljon enemmän työtä kuin kirjan kirjoittamisen perustehtävät. Otin työt kuitenkin haltuun pian. Kun olin samaan aikaan palaamassa kokopäiväiseen työhön sapattivapaalta, tämä ylimääräinen työkuorma aiheutti ongelmia ajan suhteen. Muutaman unettoman yön jälkeen sovin työnantajani Knowitin kanssa, että ottaisin kirjaprojektiin niin monta päivää palkatonta vapaata kuin tarvitsisin. Hyvin ystävällistä heiltä, mutta he ovatkin aina olleet hyvä työnantaja, joka haluaa tukea yhteiskuntaa ja työntekijöitä eikä pelkästään välitä liiketoiminnasta.

Indiegogon joukkorahoituskampanja toi tarpeeksi rahoitusta, jotta kirjan kuvitus olisi sopivan laaja. Se ei luonut tarpeeksi rahoitusta, jotta olisin voinut hoitaa julkaisemisen kokonaan itse, joten tarvitsisin kustantajia avuksi. Toisaalta, halusin muutenkin mukaan suuria kustantajia, joilla oli laaja markkinointiverkosto, joten loppujen lopuksi kaikki meni hienosti. Kampanja antoi minulle myös määräajan englanninkieliselle käännökselle. Osa kampanjaa oli ollut kirjojen lahjoittaminen kouluille, joten sain myös ensimmäiset hyvät lahjoituslupaukset. Lahjoituskampanja jatkuisi lähempänä julkaisupäiviä.

Joulukuussa ja tammikuussa otin lisää vapaata töistäni kääntääkseni kirjan englanniksi. Se oli melko suoraviivainen kokemus – sanoisin hieman nopeampaa kuin kirjan kirjoittaminen. Bonuksena, käännöksen tekeminen ennen suomenkielisen version julkaisemista antoi minulle vielä yhden katselmointikierroksen. Pystyin selventämään myös suomenkielistä sisältöä. Pyysin pilottilukijoiksi useita englanninkielisiä tuttaviani verkostostani ja sain arvokasta palautetta, kommentteja ja rohkaisua. Janet Gregory, monien ketterän testauksen kirjojen kanssakirjoittaja, teki arvostelun

kirjan takakanteen. Palkittu testausguru Paul Gerrard kirjoitti kirjani esipuheen. Olen erittäin kiitollinen molemmille.

Olin alkanut etsiä julkaisijoita kirjaan syksyllä. Kun aloitin etsinnän, tajusin pian, että siitä tulee paljon pidempi matka kuin luulin. Kustantamoiden aikataulut harkita käsikirjoituksia julkaistaviksi ovat aina vähintään useita kuukausia. Joidenkin kanssa voi käydä vuoropuhelua kirjojen sisällöstä, mutta kaikkien kanssa on oltava käytettävissä täydellinen, hiottu käsikirjoitus, ennen kuin he tekevät päätöksen ottaa kirja julkaisuohjelmaan. Kustannusneuvottelujen saaminen päätökseen kesti yli puoli vuotta. Lopuksi sain kuitenkin valita neljästä hyvästä englantilaisesta ja kolmesta hyvästä suomalaisesta tarjouksesta. Valitsin sellaiset, joilla olisi paras mahdollinen laatu tuotantoprosessissa ja markkinointipalveluissa. Molemmat sopimukset vaativat kuitenkin jonkin verran omaa investointiani, mutta olin varautunut siihen mahdollisuuteen säästämällä rahaa. Suomalainen kustantajani on Avain[6], ja englantilainen on Austin Macauley[7].

Joten, nyt kustantajat ovat työskennelleet Dragons Out -kirjan teknisen editoinnin parissa. Suomalaiseksi julkaisupäiväksi on sovittu 1. joulukuuta 2020. Englanninkielinen julkaisupäivä on vielä ilmoittamatta, mutta sen ei pitäisi enää kestää liian pitkään. Kirjaprojektin ollessa näin hyvässä tilassa olen voinut keskittyä markkinoimaan kirjaa ja ajatusta ohjelmistojen testaamisen opettamisesta lapsille. Kirjailijoiden täytyy muutenkin aina jakaa ajatuksiaan kirjan sisällöstä julkaisijoiden markkinoinnin helpottamiseksi, varsinkin jos kohderyhmä

[6] https://avain.net/

[7] https://www.austinmacauley.com/

on erityinen. Kustantajilla on kanavat kirjakauppoihin ja kirjastoihin, ja niitä tarvitaankin. Mutta heiltä puuttuu suora yhteys lukijoihin. Dragons Out -kirjan myötä sana lasten ohjelmistotestauksen tarpeesta on levitettävä testausyhteisön, IT-yhteisön ja kouluyhteisön kautta. Tämä on minun tehtäväni. Joten valmistautuessani julkaisupäivään olen aloittanut lahjoituskampanjan toisen kierroksen. Olen yksinkertaistanut viestin siten, että kauttani voi tilata kirjoja mihin tahansa tarkoitukseen ja lahjoittaa osan näistä kirjoista kouluille. Lisäksi lahjoitan henkilökohtaisesti yhden kirjan jokaista kolmea myytyä kirjaa kohden (tähän minulla on varaa suomalaisten kirjojen osalta ennakkomyynnin aikana).

Kuinka käyttää fantasiaa ohjelmistotestauksen opettamiseen

Kirja kertoo kahden 10-vuotiaan lapsen, Lauran ja Teron, kasvutarinan lapsista ritarioppilaiksi. He oppivat lohikäärmeistä ja ritareista, jotka taistelevat noita lohikäärmeitä vastaan, ja ryhtyvät myös itse toimintaan. He saavat apua mies- ja naisritareilta sekä viisailta tietäjiltä. Tarinat tapahtuvat keskiaikaista Eurooppaa muistuttavassa ympäristössä. Nämä fantasiatarinat avaavat lukijoille ohjelmistokehitystä ja testausmaailmaa analogioina ja vertauksina.

Kirjan rakenne on yksinkertainen. Jokaisessa luvussa kerron ensin fantasiatarinan uudesta lohikäärmeestä ja siitä, kuinka se karkotetaan kylästä. Tarinankerrontaani täydentää Adrienn Széllin upea kuvitus. Selitän tarinan analogian ohjelmistojen testaamiseen ja kehittämiseen tietolaatikoissa, jotka ovat luettavissa tarinan aikana tai sen jälkeen. Kuinka ohjelmistovika löydetään testauksen avulla? Miksi se on haitallista ohjelmistoille? Kuinka se voidaan poistaa? Aina, kun uusi tietokone-, testaus- tai kehitys-termi mainitaan, selitän sen mahdollisimman yksinkertaisesti sanastossa.

Jokaisen luvun lopussa on muutama harjoitus testausaiheiden oppimiseksi hieman paremmin.

Fantasiatarinoiden kerronta ulottuu muihinkin muotoihin kuin kirjaan. Ymmärrän, että kirja ei voi tavoittaa kaikkia maailman opettajia tai kouluja. Mutta ilmainen opetusmateriaali sähköisessä muodossa voi. Joten julkaisen kirjan mukana ilmaisen (Creative Commons -lisensioidun) PowerPointin, joka sisältää yhteenvedon ohjelmistojen testauksesta fantasiatarinan analogioiden avulla, sekä joitain harjoituksia. Materiaali on kaikkien maailman opettajien saatavilla.

Alla on muutamia otteita kirjasta kuvaamaan, kuinka fantasiakonsepti muuttuu ohjelmistojen testauskonseptiksi.

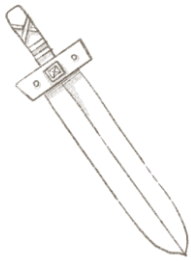
Lohikäärmeet ovat vikoja

Tarinat muuttuivat yhä villimmiksi. Lopulta vanha Aatu sanoi ääneen sen, mitä monet jo ajattelivat. Se on lohikäärme! Se syö lampaitamme. Voisiko tosiaan olla niin? Muistettiin myös, että taruissa urheat ritarit taistelevat lohikäärmeitä vastaan. Voisiko läheisestä Vuorelan linnasta saada avuksi tuollaisen urhean ritarin auttamaan tässä ongelmassa? Päätettiin lähettää kylän vanhin, Amanda-rouva, anomaan ritaria avuksi.

> *Puna-musta lohikäärme on tarinassa vika, josta muistivuoto aiheutuu. Muistivuoto on siis itse asiassa häiriö ohjelmiston toiminnassa. Vika koodissa aiheuttaa häiriön. Koodaaja on tehnyt koodatessaan virheen, josta vika johtuu. Ihminen voi poistaa vian kirjoittamalla koodin paremmin. Lohikäärmeen voi vastaavasti tappaa tai karkottaa.*

Ritaritiimi koostuu ohjelmistokehittäjistä ja testaajista

Keltaparta ja kaksi muuta ritaria olivat suunnitelleet paaluvarustuksen muotoa suurelle pergamentin palaselle. Ritari Joutsenjärvi ja kaksi muuta ritaria auttoivat kyläläisiä paaluttamaan hirret tukevasti pystyyn maahan ja sitomaan ne köysillä kiinni toisiinsa. Paaluvarustuksesta tehtäisiin mahdollisimman tukeva. Joutsenjärven tehtävänä oli lisäksi pitää lohikäärmeet loitolla.



Ohjelmistokehitystiimi on jakanut ohjelmistokehityksen tehtävät niin, että kukin tekee sitä, missä on parhaimmillaan. Keltaparta pystyy hyvin suunnittelemaan ja rakentamaan paaluvarustusta. Hän on tyypillinen koodaaja, joka osaa suunnitella ja koodata ohjelmistoa. Joutsenjärvi osaa myös löytää lohikäärmeitä. Hän on selkeästi eniten tiimin testaaja, joka osaa myös tunnistaa vikoja ohjelmistossa.

Kyläläiset löytävät lohikäärmeitä, käyttäjät ovat hekin testaajia

Joutsenjärvi käänsi hevosensa ympäri ja ratsasti nopeasti takaisin paaluvarustukselle. Hän huikkasi apunaan olleille ritareille ja rakennusmestari Vahva-Einolle, että lohikäärme oli tulossa. Kaikki teroitetut hirret pitäisi kasata äkkiä paaluvarustuksen aukolle. Keihäät ja miekat, kellä niitä oli, pitäisi hakea välittömästi. Vettä pitäisi varata ämpäreihin mahdollisimman paljon. Sitten hän meni etsimään Keltapartaa linnalta. Tero otti tehtäväkseen tiedottaa kyläläisiä. Hän juoksi ympäri kylää huutaen: ”Lohikäärme on tulossa, tarvitaan vettä, keihäitä ja miekkoja!”

Käyttäjät ovat usein kehitystiimin apuna. He voivat löytää vikoja, joita kehittäjät ja testaajat eivät löytäisi. He voivat myös kertoa, mitkä vioista ovat tärkeimpiä korjata ja mitä niille kannattaisi tehdä. Tarinassa Tero varoittaa muita lohikäärmeestä ja korostaa sen karkottamisen tärkeyttä levittämällä sanaa vesiämpäreistä, miekoista ja keihäistä.

Linnanrouvat ja linnanherrat rakennuttavat varustuksia, kuten tuoteomistajat

Kauniina keväisenä iltapäivänä kylään saapui parivaljakko, ritarit Keltahattu ja Vahvajänis. Ritarit saivat linnassa tavallista upeamman kestityksen, jonka jälkeen he saivat vatsat täynnä kuulla Margaretan ehdotuksen. ”Jääkää linnalle leipiini ja rakentakaa meille mahtava muuri!”, Margareta pyysi. ”Meillä on metsässä paljon puuta rakennusaineiksi.” Hän lupasi ritareille: ”Saatte kaksinkertaisen palkkion ja sen lisäksi saatte elellä täällä kylällä vuoden päivät.”

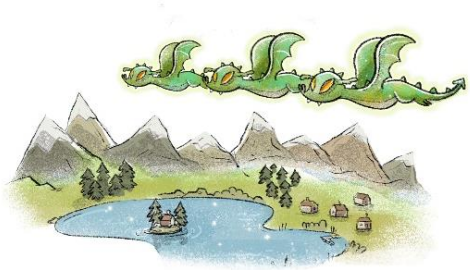
Linnanrouva Margareta on ohjelmiston tilaaja. Hän esittää toiveen muurista. Ohjelmistojen maailmassa tilaajalla on vastaavasti usein varsin ympäripyöreä vaatimus, jota ohjelmistokehitystiimin täytyy tarkentaa.

Erilaisia tapoja voittaa lohikäärme, tai poistaa vika

Kuinka ollakaan, ritarit olivat oikeassa lohikäärmeiden suhteen. Heti kun ensimmäinen kymmenen metriä paaluvarustusta oli saatu pystyyn, ilmestyi pieniä lohikäärmeitä työporukan vaivaksi. Ne olivat vihreitä ja kimaltavia olentoja, suunnilleen saman kokoisia kuin koirat. Niillä oli suuret nahkaiset siivet, joilla ne lentelivät ympäriinsä ketterästi. Ne syöksivät tulta tiuhaan tahtiin. Aaro huomasi ne ensimmäisenä, sillä hän oli ottanut tavakseen tarkkailla rakentamista läheisen talon katolta. Hän ohjasi ritareita huutamalla: ”Lohikäärmeet ulos!” Aaron mielestä olisi parempi karkottaa lohikäärmeet eikä tappaa niitä.

Huomataan, että on olemassa monia eri tapoja kukistaa lohikäärme. Sen toimintaa voi ennakoida. Samoin ohjelmistoissa monia vikoja voi ennakoida ja ehkäistä pohtimalla niitä etukäteen. Aaro huomaa lohikäärmeet

ensimmäisenä. Tätä tilaajan tekemää testausta kutsutaan yleensä hyväksymistestaukseksi. Monesti uusien ohjelmistojen rakentajat ovatkin käyttäjien tekemän testauksen varassa, varsinkin jos ohjelmisto tehdään vain yhtä tilaajaa varten.



Erilaisia lohikäärmeitä, tai vikoja

Näkyi vain liekkejä. Yksi aitta oli ilmiliekeissä ja romahtamaisillaan. Talon seinäkin paloi jo. Sitten seinä alkoi palaa eri kohdasta voimakkaasti roihahtaen. Ja heti perään vielä kolmannesta kohdasta. Roihu voimistui ja he näkivät seinän edessä leijumassa pienen mustan hahmon. Kun se siirtyi lähemmäksi palavaa seinää, erottuivat sen mustat lepakkomaiset siivet ja pitkä kiemurteleva pyrstö selvästi. Sen suusta lähti valkohehkuinen lieska, joka sytytti taas uuden kohdan seinää ilmiliekkeihin. Lohikäärme! Laura oli asiasta varma, vaikka tämä oli eri näköinen kuin Lammaslaakson lohikäärme, jonka hän oli nähnyt surmattuna.

Tietoturvavian vuoksi hyökkääjä pääsee järjestelmään, kuten tarinan lohikäärme pääsee pimeyden ja mustan värinsä turvin puhaltamaan liekkiään seiniin, joiden takana epäilee olevan ruokaa. Vihamieliset hyökkääjät ovat tietoturvatestauksen maailmassa nimeltään mustia hattuja (black hat), ja näin tarinan lohikäärmekin on musta.

Kokemus auttaa löytämään lohikäärmeitä, tai vikoja

Aamiaisella Laura ja Tero alkoivat puhua toistensa päälle kiirehtiessään kertomaan ajatuksestaan: ”Mitä jos lohikäärme…”, ”Se lentää…”, ”Harkot katoavat…”. Tarmo rauhoitti tilanteen: ”Lapset, lapset, yksi kerrallaan”. Tero kertasi, että harkkojen arveltiin katoavan iltapäivällä niiden jäähtyessä sulattamon takana. Laura kertoi, että lohikäärmeet osaavat lentää ja pitävät kullasta. Yhteen ääneen he sanoivat, että lohikäärme on varas! Tarmo-setä oli vähän epäileväinen, mutta Lauran isä Veijo sanoi, että lapset voivat hyvinkin olla oikeassa. Hän muistutti Lammaslaakson ja Kalajärven tapahtumista. Tarmon isä muisti nyt myös Teron aiemmat tarinat Metsälinnan käynniltä.

Tarinassa Laura ja Tero ovat kohdanneet lohikäärmeitä ennenkin ja osaavat sen vuoksi auttaa metalliharkkojen varkauden selvittämisessä. Kokemuksesta on myös tosimaailmassa apua. Suorituskykyviat ovat usein helppoja havaita, kun suorituskykytestauksesta on kokemusta.

Kari Kakkonen on lapsille suunnatun ohjelmistotestauskirjan Dragons Out kirjan kirjoittaja, Knowitin kouluttaja, ISTQB:n sihteeri ja innokas isä. Kari on diplomi-insinööri ja Aalto-yliopiston alumni. Kari on toiminut testauksen parissa yli kaksikymmentä vuotta. Dragons Out -kirjan ja tämän artikkelin kuvitus on Adrienn Széllin tekemä.

Tilaa itsellesi tai lapsellesi ohjelmistotestauksen fantasiakirja Dragons Out https://www.dragonsout.com/p/preorder-dragons-out-book.html

Minä en tee bugiraportteja

Anssi Lehtelä

Minä en raportoi **bugeja**.

Bugi on niin iso ja siksi myös niin helposti väärin ymmärrettävä sana, että käytän mieluummin muita termejä. Selkeämpiä tai kuvaavampia, kuten esimerkiksi havainto, ajatus, yllätys, vaihtoehto tai idea. Nämä eivät ole myöskään yhtään niin syyllistäviä termejä kuin bugi. “Bugin raportointi” kuulostaa vähän samalta kuin haasteen toimittaminen, ja miksi ihmeessä pukea tiedon lahja näin rumiin vaatteisiin?

Ja ehkä kaikkein tärkein syy siihen etten raportoi **bugeja** on se, että todella usein se mitä raportoin on yksinkertaisesti väärin, tai korkeintaankin vain osatotuus.

Jätän usein myös jotain havaintoja kokonaan **raportoimatta**. Miksi tuhlaisin omaa tai kollegojeni aikaa asioilla, jotka eivät tule johtamaan mihinkään lopputuloksiin. Jos asiassa ei ole oikeaa riskiä tai arvoa, saatan vain unohtaa koko asian. Jonkun mielestä tämä saattaa olla vastuutonta, omasta mielestäni ammattimaista.

Minä en myöskään **kirjoita raportteja**. Hyvän, sopivan pituisen, ja oikein ymmärrettävän raportin kirjoittaminen on vaativa ja aikaa vievä homma. Ja vaikka tekisi parhaansa on suuri riski, että raportti silti ymmärretään väärin.

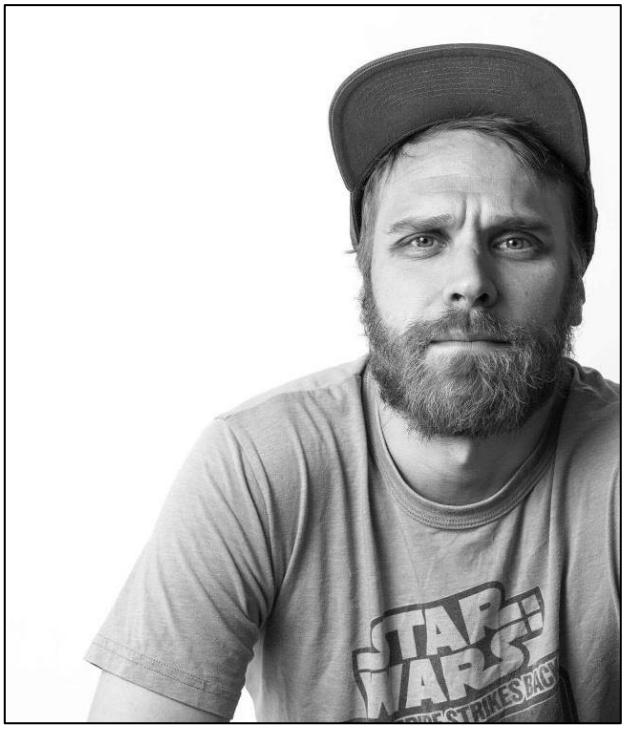
Sen sijaan minä juttelen asiasta jonkun kanssa. Sillä jonkun huomion demoaminen ja suullinen kuvailu on nopeampaa, tehokkaampaa, ja helpommin ymmärrettävää. Tällöin usein myös ongelman syyt sekä joskus mahdollisesti jopa korjaustavat löydetään nopeasti.

Silloin tällöin kyllä kirjoitan yhä jotain. Mutta jos niin teen, niin tämä asia olisi mieluiten vain kutsu keskusteluun, kuten vaikkapa post-it lappu. Taikka lyhyt kuvaus huomiosta slackiin, jossa asiasta on helppo keskustella. Loppuun asti pyrin välttämään bugiraportin tekemistä johonkin backlogiin tai bugiraporttijärjestelmään, minne suurella todennäköisyydellä se ajan saatossa vain unohtuu, muiden kauniiden raporttien tavoin.

Jos voisin tehdä raportin “feilaavana testinä”, saattaisin näin tehdä. Mutta silloinkin tuntuu että käytän liikaa aikaa itsekseni asian kanssa, ilman keskustelua. Ja mikäli näkisin vaivan feilaavan testin tekemiseksi, miksi en sitten samoin tein korjaisi ongelmaa itse, tai yhdessä jonkun kanssa?

On huvittavaa, että ennen järjestin kisoja parhaista bugiraporteista, ja pidin luentoja bugien raportoimisesta, mutta tänä päivänä en tee sitä edes itse.

Ja se on mahtavaa.



Anssin yhteystiedot:

Twitter: @hellofatester

Linkedin: https://www.linkedin.com/in/anssil/

TestausOSY

Finnish Association of Software Testing