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. Meta-tasolla 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, datatyppeihin 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.

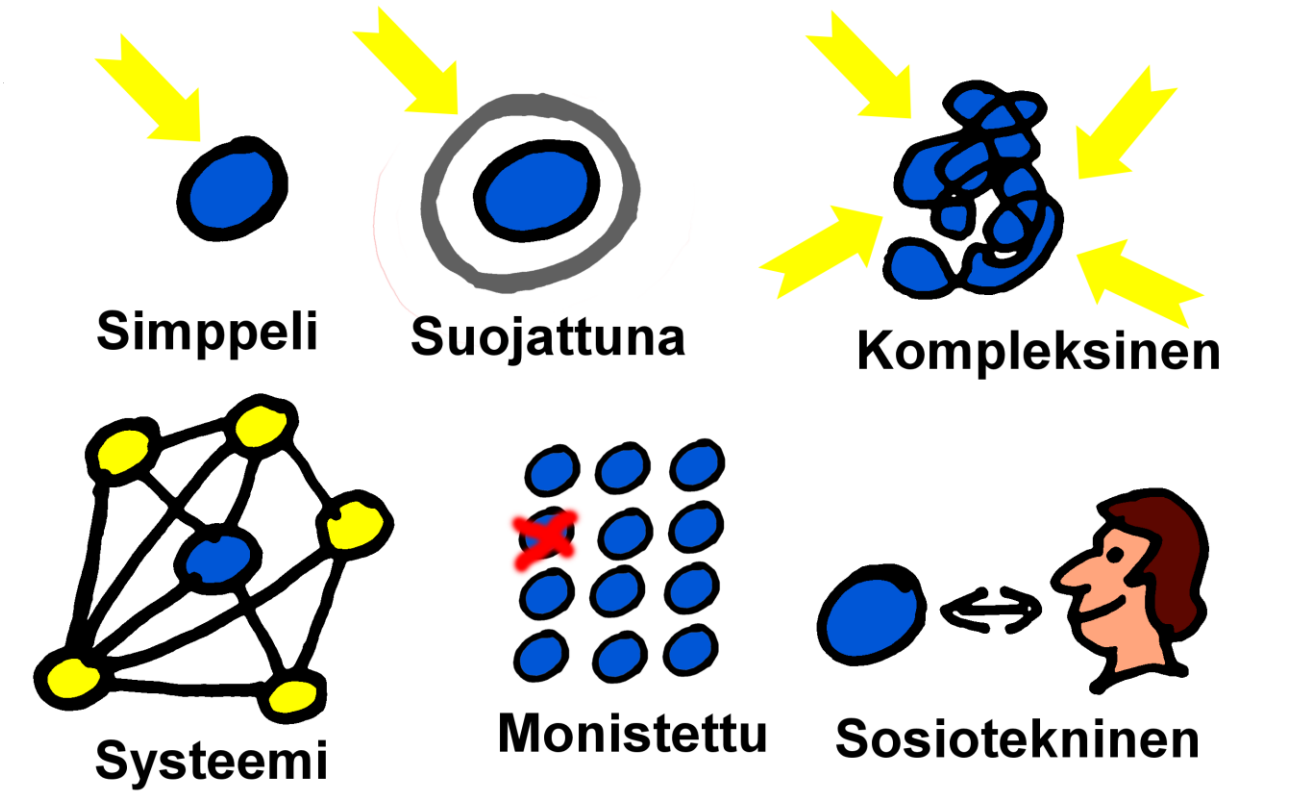
Aiemmin on viitattu luotettavuusanalyyseihin. Arkkitehtuurin arviointi on myös elinkaarta käsittelevä luotettavuusanalyysi, koska siinä yleensä tarkastellaan erilaisia systeemiin vaikuttavia skenaarioita – käyttäjien lisääntyminen, systeemin elementtien vaihtaminen, uusien ominaisuuksien kehittäminen 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