# Tarkvara hindamine.

Maksumus, töökindlus, testimise korraldus

## *Mõõdetavad suurused*

Mõõdetakse päris paljusid tarkvaraga seotud suurusi, kuid eesmärgist lähtuvalt võiks eristada järgmisi alamhulki:

* vastavus spetsifikatsioonile
* sobivus kasutajatele
* veakindlus
* kulud loomiseks ja haldamiseks.

Ehkki "hea" programmi loomisel tuleb kõiki neid aspekte arvestada, võib mõnel juhul ühe või teise koha pealt silma veidi kinni pigistada. Nagunii pole enamasti üheski valdkonnas võimalik täielikku täpsust saavutada, tuleb piirduda hägusa loogika ning leiduvate vahenditega. Spetsifikatsioonile pea täpne vastavus on näiteks vajalik, kuid loodav moodul peab ilma muudatusteta sobituma teiste omasuguste hulka. Siis tuleb kasvõi töökiiruses ja kasutusmugavuses järele anda, et loodud tarkvaratükk standardile vastaks. Niigi on näiteid standardi järgi loodud draiveritest mis mõnes kombinatsioonis töötavad ja teises mitte ning vaesel kasutajal pole sageli muud teha kui meelehärmist kukalt kratsida. Parem kui me midagi sama hullu enam juurde ei loo. Selle kontrollimiseks tuleb mõõta loodud tarkvara vastavust standarditele.

Iseseisvalt töötava programmi puhul on pigem tähtis, et kasutaja selle abil oma töö võimalikult hästi ja kiiresti valmis saaks. Nii võib mõnikord sellisel juhul pigem eelnevast detailsest spetsifikatsioonist loobuda ning töö käigus tekkinud häid ideid juurde lisada. Loomulikult tuleb seejärel testida nii tehtud muudatusi kui kogu programmi kasutaja(te) peal ning hoolitseda, et tulemus võimalikult hästi tarvitatud oleks.

Mida raskemad on võimaliku vea tagajärjed, seda hoolikamalt tuleb neist hoiduda. Kui suurest mälust ühekordselt paar baiti rohkem kulub või kasutaja mõne operatsiooni juures sekundi murdosa jagu rohkem ootama peab, siis pole veel asi väga hull. Valesid tulemusi andev funktsioon võib juba rohkem muret valmistada ning kui selline koodilõik juhtub sattuma lennuki juhtimissüsteemi, siis võib seda mõne aja pärast juba ajalehe õnnetusteveerust lugeda. Järelikult sellistel juhtudel on vajalik veakindlust põhjalikult testida.

Tarkvara loomise kulu on võimalik samuti ennustada ning mõõta. Sama ülesande lahendamisel võib konkureerivate võimaluste puhul projekti maksumusega võrreldes väike osa otsustada, millises suunas on mõistlik liikuda.

Valikuline loetelu tarkvara mõõdetavatest suurustest

Maksumus

* Kulud projekteerimisele, kirjutamisele, testimisele ja muule (õpe, sõidud, kohtumised).

* Keskmine koodirea maksumus
* Valmimisjärgse koodihoolduse kulud
* Dokumenteerimisele kulutatud aeg
* Arvutipargi hind
* Ümbertegemiseks kulunud vahendid

Vead

* Tarkvara loomisel ning testimisel ilmnenud vigade ning veatüüpide hulk
* Millal ja kuidas vead leiti
* Spetsifikatsioonist leitud vead
* Moodulite õnnestunud ning vigaste ühendamiste suhe.

Võrdlus eelmise tarkvaraprojektiga

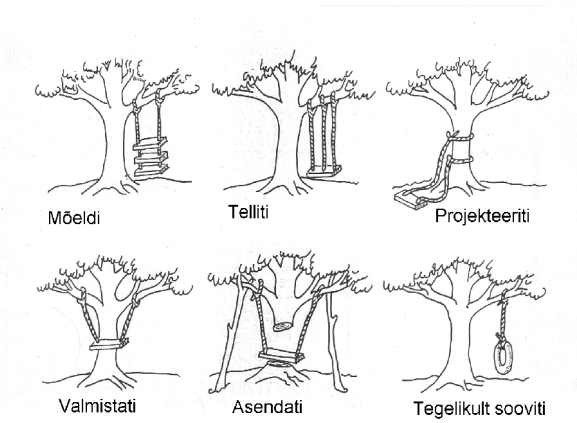
* Lähtekoodi kasvukõver
* Koodimuutuste jaotus arendamise ning haldamise käigus
* Maht koodiridades/keerukuspunktides
* Ajakava sõltuvus mahust
* Hinna sõltuvus mahust
* Töötajate hulk
* Dokumentatsiooni maht

Maksumus jagati vanemates käsitlustes enamjaolt kolme võrdsesse ossa projekteerimise, kirjutamise ning testi vahel muutes mõnikord suhteid sõltuvalt tarkvara eripärast. Nüüd aga on leitud, et teistega vähemalt samaväärne on juurdekuuluv "muu", kuhu kuuluvad nii konteksti uurimine, uute oskuste õppimine kui enese tutvustamine. Ilma selleta võib tehiskeskkonnas mõnda aega hakkama saama, kuid iseseisvana tänapäeva muutuvas maailmas ei tule hästi välja.

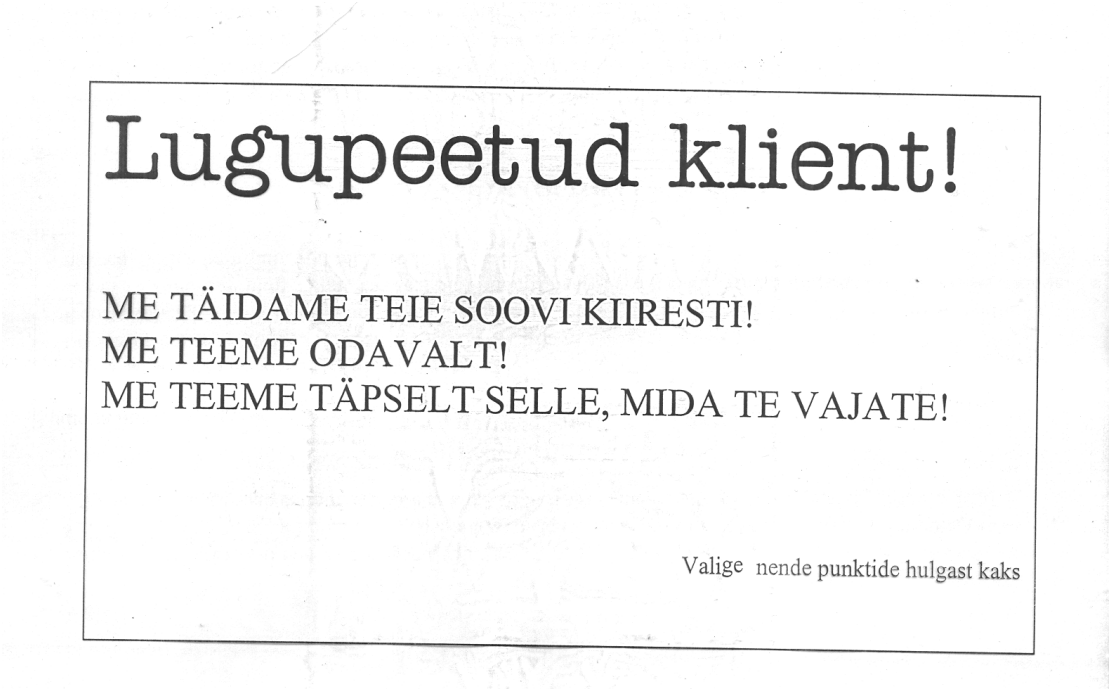
Projekte uurides ja võrreldes on leitud mõnesugused seosed. Küll nõrgad ning eri valdkondade ja meeskondade puhul tublisti erinevad, kuid siiski märgatavad seosed. Näiteks, et sama tüüpi rakenduste puhul on valminud koodiridade arv ning tööjõukulu ligikaudu võrdelised. Ning, et mida mahukam projekt, seda enam on sinna põhjust ja võimalust inimesi tööle võtta. Väiksema ülesande puhul pole suurel inimeste hulgal mõtet, sest toiminguid ei anna nõnda kergelt üksteisest sõltumatuks jagada ning algse ülesannete jagamise ning pärastise tulemust kokku ühendamise peale võib rohkem aega ja jõudu kuluda kui kogu lugu üksinda tehes oleks võtnud.

Kuna mahukamatel projektidel on võimalik rohkem inimesi tööle võtta, siis nende kestus kasvab tunduvalt aeglasemalt kui maht – siin tabelis on pakutu suisa neljandat juurt. Samas dokumentatsioon tuleb suuremate projektide juures ka märgatav – selle hulk on ligikaudu võrdeline töötavate inimeste arvuga.

Kui suuremat tarkvaraprojekti põhjalikumalt ette ei kavanda, siis juhtub kergesti nõnda, nagu näha järgnevas Soome huumorikogumikust pärit koomiksis. Et kui müügimees, arendaja ning klient oma vajadustest selgesti aru ei saa, siis võib kuluda palju aega ja vahendeid  saavutamaks mingilgi moel rahuldavat tulemust – selle asemel, et lihtsate vahenditega soovitud funktsionaalsus saavutada. Tõsi küll – esimesel juhul on põhjust ja võimalust iga sammu jaoks vahendeid küsida ning täiesti uues valdkonnas tegutsedes on harva lootust paljalt oma ideedele tuginedes lihtsale ja toimivale lahendusele jõuda. Kui siinset pilti aga silma ees hoida, siis tasub teada, et peaaegu alati on mõni lihtsam lahendus olemas. Nii nagu malemänguski.



Igal asjal pidada olema oma head ja halvad küljed. Ning kui kusagilt võita, siis tuleb teisest kohast järele anda. Programmeerimisfirmadele olen kuulnud soovitus panna ukse peale järgnev plakat.




Nagu näha, saab kliendi meelitada kohale, sest kõik pakutud punktid on üldjuhul täidetavad. Ainult, et korraga nende täitmiseks peab lihtsalt õnnelik juhus olema. Kahe punkti täitmine on aga palju jõukohasem. Kui soovitakse kiiret ja odavat lahendust ning pole nõnda tähtis, et kõik tehnilised nõuded täpselt täidetud oleks, siis võib mõningase otsimise tulemusena leida sobivad valmiskomponendid olgu siis oma firma seest või võrgu pealt ning nende abil sobiva valdkonna rakendus kokku klopsida. Kui hind pole tellijale probleem, siis võib mitmete tegijate, pikkade tööpäevade ja ostetud komponentidega täiesti kõlbuliku tulemuse saavutada. Ning kui pole kiiret, sel juhul võib mõnigi tarkvaraprojekt valmida küllatki "muu seas", pannes pikema aja jooksul tallele sobivaid koodilõike ja komponente ning siis mõne päevaga lõpptulemuse viimistleda.

## Vead

Traditsiooniline ning kõige selgem ja arvutilähedasem tarkvara kvaliteedi mõõtmise valdkond. Kui pidevalt saadakse sisestatud õigete andmete abil valesid tulemusi, siis ei saa seda tarkvara mitte heaks pidada. NASA-s tehtud 5-aastase uuringu käigus ilmnenud ligi 10000 viga jagati tähtsuse järjekorras järgmistesse gruppidesse:

* Valeandmed
* Vigased liidesed programmide vahel
* Programmi tööjärje suunamisvead
* Väljade algväärtustus, mälu eraldamine ja vabastamine
* Arvutusvead

Liidesed tulevad mängu suuremate programmide puhul, kus väikesest ebaklapist võib järgneda hulk probleeme, sest sama ühendust kasutatakse paljudes kohtades. Algväärtustamise ja mäluga seotud vead sõltuvad mõningal määral kasutatavast

programmeerimiskeelest ning -tehnikast, ülejäänuid aga tuleb kindlasti iga programmi koostamisel arvestada, mõõta ja parandada.

## *Testid*

Eriti algajal kirjutajal, kuid pea alati ka kogenumal koodiloojal ei hakka programm pärast lähtekoodi valmistippimist kohe õigesti tööle. Ikka õnnestub sisse teha mõni süntaksiviga, et selle tulemusena translaator loodud tekstist aru ei saa. Või kui puuduvad semikoolonid või sulud on paika saanud, siis ikka juhtub, et kõik vastused ei tundu inimliku loogikaga järele kontrollides kuidagi mõistlikud olema. Ühele lehele mahtuva käsujada puhul jõuab silmadega koodi üle vaadata, mitmesuguste andmete ja väljatrükkidega katsetada kus viga leidub ning see rida muuta või välja kommenteerida ja paremaga asendada. Nii saab enamasti mõne(kümne) minutiga programmilõigu tööle ning võib seda rahumeeli enese huvides kasutama hakata. Kasutasime intuitiivselt testimise võtteid ning tõenäoliselt edukalt. Kui aga programm on suurem ja mitme inimese hallata või on igal leiduval veal kõrge hind, siis sellisest lähenemisest ei piisa ning tuleb süstemaatiliselt kontrollida, kas masin etteantud käskude peale teeb seda mida vaja ning mitte lihtsalt seda, mis me talle ette kirjutasime. Siinsele lühikokkuvõttele võib pikemaid seletusi kõrvale leida raamatust Tarkvara kvaliteet ja standardid, Tepandi 1997.

## Laused ja harud

Süstemaatiliseks testimiseks on hulk võimalusi mitmesuguse veakindluse ning ressursikuluga. Üks võimalus on võtta ette programmi tekst ning proovida programmil lasta läbida kõik kirjutatud laused, soovitavalt vähemalt kaks korda. Juhuslike andmetega katsetades võib kergesti juhtuda, et läbitakse korduvalt samu teid ning eriolukordadel töötlemiseks loodud programmilõigud (mis korralikult tingimusi kontrollival programmil võivad võtta küllalt suure osa mahust) võivad paljus jääda läbimata. Selline test ei avasta küll samas lauses lähteandmetest tingitud eriolukordi, kuid võimaldab leida mõned suuremad kahe silma vahele jäänud apsakad ning näitab kätte ka lõigud, kuhu polegi võimalik jõuda.

Eelkirjeldatud lauseadekvaatsuseks nimetatud meetodi täiendusena on loodud haru- ning elementaartingimuste adekvaatsuse meetodid. Niimoodi tuleb lisaks kõikidele käsulausetele läbida ka tingimuste tühjad pooled ning liittingimuste puhul proovida iga osa töötamist. Näiteks

```
if (vanus<10) or (vanus>20) then

writeln('pole teismeline');
```

juures piisaks esimesel juhul kontrollida väärtusega nt. 11, teisel 7 ja 11 ning kolmandal 7, 11 ja 25.

Lähtekoodita test, piirjuhud

Ka lähtekoodita saab süstemaatiliselt testida, siis tuleb leida andmevahemikud, mille juures programm peab erinevalt käituma. Alustatakse aga nii, kuidas ka tavainimene võiks programmi tööd proovima hakata, umbes sellises järjekorras:

* Kas programm üldse midagi teeb, tööle läheb
* Kas tähtsamad kohad töötavad.
* Kuidas õnnestub masinalt viga välja meelitada
* Kas tehakse kõike mis tarvis
* Kuidas kõik kättesaadavad vahendid toimivad
* Andmed piirjuhtudel
* Suur koormus (palju andmeid, kasutajaid)

Kui ühtemoodi töödedavad andmed asuvad mingis vahemikus, siis tasub proovida sisestada väärtusi nii vahemiku seest, eest kui tagant. Piiri juures tuleks teha mitu katset. Testi sisendandmete komplekt tuleks koostada nii, et kahtlane väärtus oleks vaid üks, ülejäänud asugu võimalikult probleemidevabas piirkonnas. Nii on kergem vigu üles leida. Testida tuleks nii sisendite kui väljundite piirolukorras. Mõned näited piiride leidmisekst.

Reaalarvu puhul proovida täpselt piirjuhuga, samuti üles- ning allapoole võimalikult vähe, kuid siiski eristatava arvuga. Kui väärtusel pole ülemist piiri tasub proovida väga suure arvuga. Hulkade puhul tuleks võtta arve nii seest kui väljast. Samuti võib piirolukordadena arvestada nii täis- kui reaalarvu lubatud maksimaalset või minimaalset suurust ja massiivide varutud pikkusi.

## Juhuslikud andmed

Lihtsalt juhuslike andmete väljamõtlemisega kaasneb oht, et kasutaja pakub peast samu alateadlikult tuttavaks saanud väärtusi ning mitmed vahemikud jäävad proovimata. Lihtsalt matemaatilisi juhuarve pakkudes läheb skaala laiaks ning vajalikku piirkonda võib suhteliselt vähe katseid langeda. Kui aga eelnevalt kindlaks teha ligikaudsed enam tarvilikud piirkonnad ning siis seal juhuarvudega testida võib tulemus päris tõhus olla. Lisaks on siiski vajalikud piirolukordade testid.

## Koodi analüüs, tüüpvead

Mida selgemini kirjutatud kood, seda lihtsam on vigu leida. Kehtib pea alati, kui teine inimene peab lähteteksti uurima, kuid ka ise on võimalik mõne ajaga loodud käske ja ideoloogiat nii palju unustada, et omakirjutatud koodist läbinärimine võib pea sama raske töö olla kui ise sama koodilõigu uuesti kirjutamine. Eri keelte ja valdkondade jaoks on koostatud loetelud sagedamini esinevate vigade kohta, mõned näited:

* Segased muutujanimed
* Kas pöördumisel muutujal väärtus olemas
* Andmete lugemise järjekord.
* Kas samu andmeid talletatakse/loetakse ühtmoodi
* Lubamatud tüübid, tüübimuundus
* Üle/alatäitumine

* Jagamine nulliga
* Muutujatel kahtlased väärtused (vanus=200)
* Ebatäpsuste kuhjumine (tsüklis liidad 1, lahutad 0.99)
* Sulud
* Kas spetsifikatsioon õieti tõlgitud
* Tsükli kordamine 0 või väga palju kordi
* Alamprogrammi sisendparameetrite muutmine
* Globaalsed ja lokaalsed muutujad

## Moodulite ühendamine

Lihtsamate programmide korral võib valminud eraldi testitud moodulid ühte panna ning seejärel katsetada, kas ja kus vigu leidub. Suure hulga moodulite puhul on aga vea allikat raske üles leida. Vaheetapina saab kasutada üheskoos töötavate/testitavate moodulite kogumeid. Nii saab sellele mooduliplokile andmeid saata ning uurida, kas nad tagastavad selliseid väärtusi, nagu neid kirjeldav liides ette näeb.

Süsteemist terviklikuma pildi saamiseks võib osa mooduleid asendada väiksemate programmilõikudega või suisa inimpoolse vastamisega.

## Vigade hulga hindamine

Üks võimalus vigade hulga mõõtmiseks on jälgida vigade avastamist sama intensiivse töö korral. Esialgu on vigade osakaal koodis suurem ning neid leitakse kergemini. Tasapisi avastamine väheneb ning seda kõverat pikendades võib ligikaudu hinnata allesjäänud vigade arvu.

Alternatiivina saab kasutada lisatud vigu. Kui tegijaid on mitu, siis saab üks lisada teise poolt uuritavasse moodulisse tahtlikke vigu. Uurides leitud tegelike ning lisatud vigade suhet saab aimata allesjäänud vigade arvu.

## Testimise maksumus

Kõiki vigu on lootust harva kätte saada. Kui valida konkreetse programmi jaoks sobivat testi, siis vigade ohtlikkuse ning parandamiseks kuluva hinna suhted oleksid järjekorras:

* proov kasutajalt saadavate andmetega (tõenäolisemad enamkasutatavad koodilõigud)
* läbivaatused (avastavad varaseid kalleid vigu)
* vea oletus (ekspert)
* piirjuhud
* sisendandmete grupid
* topeltprogramm
* lisatud vead
* tõestamine

### *Vigade põhjalikum püüdmine*

Väga veakindlate programmide testimiseks tavalistest meetoditest ei piisa. Ka ressursside põhjalikumal kulutamisel ei õnnestu naljalt viia vigade arvu alla viie ühe tuhande koodirea kohta. Edasi iga järgneva vea avastamiseks tehtavad kulutused suurenevad tohutult. Kalleid seadmeid juhtiv programm peaks aga töötama suhteliselt laitmatult, sest koodiapsaka tõttu tekkinud katastroof või suur varaline kahju joonistab tarkvara tootjale paratamatult suure pleki. Tehiskaaslase juhtimiseks kasutatakse aga ligi pooltteist miljonit koodirida nii et selle töökindlana loomine ning hoidmine ei pruugi kuigi kerge ülesanne olla. Liiatigi kui tegelikes oludes katsetamist kuigi palju lubada ei saa. Mõned vahendid kitsaskohast ülesaamiseks.

Sama ülesannet täitev programm luuakse mitmel moel ning saadud vastuseid võrreldakse. Nii leiab võimalikke vigu testimisel ning käivitades võib häda korral ka eeldada, et kui mitu algoritmi näitavad sama tulemust, üks aga erinevat, siis esimestel on tõenäoliselt õigus. Selliseid lahendusi kasutati paarkümmend aastat tagasi suurarvutite puhul, kus lambi läbipõlemine võis kergesti tulemusi muuta. Tarkvara puhul aga kipub inimene samu vigu tegema ning sellega meetodi töökindlus väheneb.

Kaitsemehhanismid ning veapuu analüüs. Suurte vigade (näiteks kokkupõrke) vältimiseks saab mõnikord piirata nende vigade eeltingimusi. Tulemusena võib muutuda toimimine aeglasemaks kuid turvalisemaks. Eks kõigil ole oma hind.

Matemaatiline tõestamine, et programm vastab spetsifikatsioonile. Töömahukas, kuid kriitilistel puhkudel vajalik.

Lühikesed alamprogrammid, lihtsad moodulid. Mida väiksem on lõik, seda enam on sealt lootust kõik vead üles leida.

Muid töökindluse parandamise vahendeid

## Autori analüüs

... ehk lihtsalt programmi koostaja või koostajate grupp vaatab oma töö pärast enese arvates valmimist veel korra terase pilguga üle. Hea tahte puhul küllalt tulemusrikas meetod, lihtne korraldada ning nõuab vaid koostajalt lisatööaega, kuid muude kulutustega seotud ei ole. Miinuseks on, et autori mõte käib vana rada, ta ei suuda jälgida kasutaja loogikat ning seetõttu jäävad võimalikud vead avastamata. Käsud, mis autorile une pealt selged on, ei pruugi võõrast programmi kasutavale inimesele sugugi mitte tuttavad olla ning seetõttu võib viimane päris kergesti imelikesse olukordadesse sattuda. Samuti on autori eesmärk lõpetada töö, mitte leida vigu ning ta ei soovi oma valmistehtud loogikat lõhkuda olgugi, et sellest võivad mitmed arusaamatused alguse saada.

## Läbivaatus

Tarkvara tootev seltskond istub kokku ning igaüks saab ülevaate ka teiste tegemisest. Teades, et ka teised sinu tööd jälgivad on ka omal alateadlik soov mõnigi asi ilusamini ja selgemalt teha, et see paremini arusaadav oleks ning teistele vigasena ei paistaks. Mitmekesi koos uurides on mitu mõttekäiku ning harvemini jääb mõni tähtis detail

tähelepanuta. Nii suurendatakse inimestevahelisi kontakte ning vajaduse korral on ühte liiget võimalik kergemini asendada. Kasulikuks korraldamiseks peavad sellised läbivaatused olema ettevalmistatud, muul juhul võib koosistumine lihtsalt lõbusaks mokalaadaks kujuneda. Samuti peab leiduma juhataja, kes aitab ajakavast kinni pidada ning hoolitseb, et osavõtjad teaksid ettevõtmise eesmärke ning oma kohustusi. Isiklike vastuolude või temperamentsete isikute puhul võib tekkida probleeme.

## Audit

Väline hinnang valmivale tarkvarale või valmistamise protsessile. Üheks eesmärgiks on välistele osapooltele (näiteks tellijale) näidata tootja usaldatavust. Samuti aitab asutuse juhtkonnal otsida ilmnenud probleemide põhjusi või avastada võimalikke ohuallikaid. Võõra inimese käest on mõnel kriitikat kuulda või lugeda kergem kui omade seast. Samas kui pole sisemist valmisolekut lasta oma vigu välja paista või nendest õppida, siis pole loota ka auditi abil oma töö tulemuste paranemist.

## *Testimise korraldus*

Kui kogu programmi loob ja kasutab sama inimene ning ilmnenud vead ei saa tuua suurt kahju, siis piisab kui testitakse ja parandatakse vigu töö käigus.

Ühe looja ja ühe tellija puhul kirjutab tellija programmist leitud vead üles, arendaja parandab need ning ei tohiks ka muid keerulisi protseduure sinna juurde tekkida. Kui tarkvara täidab vastutusrikast ülesannet ja/või on sellel palju kasutajaid, siis peab teste olema mitu: nii tootja, testijate kui väikse kasutajate grupi juures. Hoolimata sellest kipub märgatav osa vigu siiski lõppkasutajate juurde jõudma, kes ei oska nende parandamiseks suurt midagi ette võtta. Mõnel puhul aga pole võimalik eeltoodud süsteemi kasutada, siis tuleb leida abivahendid.

Keeruka toote puhul tuleb alustada kontrolliga juba algusest ning hallatavate osadena vaadata üle nii spetsifikatsioon kui kood. Siis on lootust pärastise osade ühendamise juures ka tekkivate vigade põhjustele kergemini jälile saada. Sama käib ka lihtsama kuid ülisuure töökindlusvajadusega programmi kohta. Seal peab iga rida olema mitmeid kordi läbi uuritud ja põhjendatud.

Kui projekteerijad ja programmeerijad asuvad lahus, siis peab arvestama vajadust programmi kirjutamisel spetsifikatsiooni täiendada, kui selgub, et midagi olulist on tähelepanuta jäänud. Hajusalt asuva kasutaja puhul on sealtpoolne vigade teada saamine ning pärastine muudatuste tegemine keeruline. Ikka leidub keegi, kelle masinasse on vana versioon sisse jäänud ning kus mõned tehtud parandused ei tööta.

Muutmiste juures tuleb kontrollida, kas endised testid töötavad. Kergesti võib juhtuda, et ühte viga parandades luuakse teine või rohkemgi juurde.

Mõned tarkvarameetrika reeglid
* Tarkvarakontroll pole imevahend

Kontroll ja mõõtmine ilma lähema eesmärgita toob lihtsalt muret ja vaeva. Kasu võib sellest olla vaid ühe abinõuna tarkvaratöö korraldamisel

* Keskendu tulemuste kasutamisele, mitte andmete kogumisele.

Kui liialt palju energiat kulub esimese peale, siis on raske kasuliku tulemuseni jõuda.

* Selgita eesmärgid.

Tarkvaratootja püüab enamasti suurendada toodangu hulka või kvaliteeti, vähendada kulusid, püsida graafikus. Enne, kui asuda mõõtma ja kontrollima, tuleks selgeks teha, millises järjekorras ja valdkonnas oleks mõistlik muudatusi teha. Kus nagunii kõik esialgu samamoodi jääb, seal pole suurt mõtet ka mõõtmisele vahendeid kulutada.

* Uuri, kuidas mõõtmistulemusi rakendada.

Tippjuhid tõenäoliselt loodavad uuringust teavet põhiprintsiipide kohta. Projektijuhid soovivad andmeid jooksva projekti käigushoidmiseks ning uute plaanimiseks. Programmeerijatel on lootust saada teada, millele on mõistlik enam ja millele vähem rõhku panna. Hea meetrika puhul peaks pea samade andmete analüüsist kõik osapooled kasu saama.

* Uuri rakendamisvalmidust.

Mõõtmistulemuste kasuliku tarvitamisega kaasneb paratamatult vajadus inimeste töös muudatusi teha. Enne mõõtmise algust on mõistlik kindlaks teha, kes, kas ja kui palju on valmis oma tööharjumusi muutma ning kui kulukaks selle sisseviimine läheks ettevõttele. Head süsti võib mõõtmiselt oodata vaid siis, kui nii juhtkond kui tehniline personal on valmis uuringu tulemusi oma töös arvesse võtma.

* Kavanda kiire edu

Esimene mõõdetav projekt tuleks valida selline, kus on loota juba lühikese aja jooksul märgatavaid tulemusi. Siis tekib osalejatel usk, et mõõtmise ning tulemuste rakendamise abil on võimalik omale ja teistele kasu saada. Mitmete muidu heade uuringute hädaks kipub olema, et mõõtmine ning tulemuste rakendamine võtavad nii kaua aega, et töötajad on selleks ajaks suurest andmete kogumisest tüdinud ning ei arva mõõtmisest enam kuigi hästi. Sellises olukorras aga soovituste andmine, et tee nii või teisiti, kuigi see võiks pikemas plaanis kasu tuua tekitab pigem trotsi kui valmisolekut uuendustega kaasa minna. Kiiresti võib uurimisest meeldiva laengu saada näiteks vastava ala programmidest tüüpvigu leides. Kui mõne nädalaga või rutemgi on võimalik teha analüüs ning näidata levinumate vigade jaotus, siis saab seda kirjutamisel varsti arvestama hakata ning kirjutaja suhtub uurijasse juba tunduvalt väiksema ettevaatusega, võibolla suisa poolehoiuga. Soovides aga kogu protsessi suunamiseks tarvilikke vihjeid saada, selleks tuleb siiski pikema ajaga ja põhjalikumate arvutustega leppida.

* Keskendu seotud grupile

Kui kavatsetakse tarkvaratöö tulemust mõõtma hakata, siis tuleb paratamatult otsustada, milliseid projekte, millises arengujärgus ning mis tüüpi tööd seal tuleb mõõtma hakata. Edukat uuringut koos tulemuste rakendamisega on tunduvalt lihtsam läbi viia seotud grupis, näiteks ühes hoones asuvas ettevõtteüksuses. Suuremate

maastaapidega saab üldisemaid andmed ning neid võib kasutada teadustöös või muudes teoreetilistes aruteludes, kuid nende abil pole kuigi palju lootust konkreetsete inimeste tööd tegelikult paremaks muuta. Uuringu tulemuseks saab olla olemasoleva töö analüüs ning oma vahendite abil selle paremaks muutmine, mitte võrdlus teiste gruppidega.

* Alusta väikselt

Mõõtmise sisseviimisel alusta väheste inimeste, tarkvaratootmise üksikute selgepiiriliste etapide ning üksikute projektidega. Väiksest õnnestunud mõõtmisest on kergem suurema peale edasi minna.

* Tootjad ning mõõtjad olgu eraldi

Neil on erinevad huvid ning ka hea tahtmise korral kipuvad ühed teisi segama.

* Kindlusta, et mõõtmised vastaksid eesmärkidele

Pole mõtet koguda andmeid, mille analüüsi tulemusi pole plaanis realiseerida. Iseenesest võivad mitmed väärtused tunduda huvitavad, aga kui neid pole kavas selle uuringu käigus kasutada, siis kokkuvõttes ikkagi raisatakse ressursse.

* Kogu võimalikult vähe andmeid.

Kui lisamõõtmised järelduste täpsust oluliselt ei suurenda, pole mõtet ilmaasjata andmeid kirja panna

* Hoidu liigsete tulemuste esitamisest.

Lugemisel parajasti kasutu kraam on müra. Kui uuringu käigus selgus huvitavaid tulemusi, tuleks neid näidata neile, kel selliste tulemustega midagi mõistlikku teha on. Üldraportis sellised tabelid ja graafikud lihtsalt raiskavad kasutajate aega ning riiulipinda. Tüüpilised andmed, mis liiaga raportitesse lisatakse, sest nad on olemas: testide arv, arvutite kasutusgraafikud, kohtumistele kulunud aeg, koodi keerukusaste.

* Arvesta uuringu hinda

Uuringul on mõtet, kui temasse kulutatud vahendid end tasa teenivad. Märkimisväärne osa kuludest tuleb kanda uuringu alustamisel, edaspidine ei pruugigi enam nii hull olla. Uuringu suuremad kulud: andmete kogumine - ankeetide täitmine, andmete saatmine (tarkvaratööliste lisatöö); tehniline personal - raamatupidamine, tulemuste transport õigesse kohta; analüüs, tulemuste esitamine. Uuring võib suures organisatsioonis võtta ~6% selle aja jaoks eraldatud ressurssidest, väiksemas aga kuni viiendiku. Andmete kogumine ei tohiks võtta üle paari protsendi projekti eelarvest, tehniline abi peaks piirduma nelja-viiega ning analüüsile võib veidi rohkem kuluda.

* Kavanda andmete analüüsile vähemalt kolm korda rohkem ressursse kui kogumisele

Maailmas on kogutud juba väga palju andmeid kuid suur osa neist vananeb andmekandjatel täiesti kasutuna või on neist kätte saadud tühine osa võimalikust teabest. Püüa seda viga vältida. Kogu ainult tarvilikke andmeid, võimalikult vähe ning püüa nende põhjal järeldused kätte saada.

* Kogu tööjõukulu andmeid vähemalt kord kuus

Kui vaja siis ka tihemini. Pikema aja peale oskab inimene harva tagantjärele oma pingutusi täpsemalt hinnata. Loodud kogumissüsteemi tihedamini tööle panna on lihtsam kui süsteemi käivitada.

* Uuri, millise tööjõukulu kohta on mõistlik andmeid koguda

Kas näiteks programmeerija suudab eristada, palju tal kulus aega koodi kirjutamiseks palju testimiseks. Mõnikord need kipuvad omavahel nii ühenduses olema, et eristada on küllalt raske.

* Kogu vaid töötavaks kuulutatud tarkvara vigu.

Konfigureerimata ja juhuslikult kokku pandud tükkide juurest leitud vigu ei õnnestu kergesti süstematiseerida. Samuti on selliste vigade ülesmärkimine paratamatult juhuslik ega anna pilti tegelikust vigade jaotusest. Muidu tuleks vigade juurde märkida: leidmise kuupäev; parandamise kuupäev; avastamiseks ning parandamiseks kulunud tööaeg; vea allikas (spetsifikatsioon, kood, eelmine muudatus); vea tüüp.

* Vea parandamiseks minevat aega on raske mõõta

Programmeerija saab tunduvalt kergemini öelda, palju kulus tal aega leidmiseks ja parandamiseks kokku.

* Tarkvaraprotsessi on raske mõõta

Samale tulemusele võib jõuda mitmeti.

* Jaga programmi loomine selgepiirilistesse etappidesse

Siis saab kergemini jälgida ajakavast kinnipidamist ning osalistel on selge hinnata oma edukust.

* Kasuta programmi mahu mõõtmisel ka koodiridu

Kuigi see mõõtühik on küllalt laialivalguv ning sõltuvalt keelest, kirjutajast ja keskkonnast võib samasuguse programmi puhul ligi suurusjärgu võrra erineda, on tegemist siiski lihtsalt edastatava suurusega, mis samades tingimustes aitab programme omavahel võrrelda. Lisaks tavalisele numbrile, kus loetakse kokku kõik tarkvara koosseisu kuuluvad read sõltumata sellest kas seal ka midagi kasulikku tehakse, eristatakse ka käskluste arvu (mõnel real võib neid olla mitu), kommentaaride arvu, korduvkasutamiseks loodud ridu. Kui programmis kasutatakse varem valminud lõike, siis saab programmi või tema osi jagada: uus; tugevasti muudetud (üle 25% ridadest); nõrgalt muudetud ning muutmata. Teisteks programmi mahu mõõtühikuteks

võivad olla näiteks tsüklomaatiline keerukus (erinevate võimalike harude arv) või punktid kus arvestatakse nii projekteerimise, kirjutamise kui testimisega seotud tööd.

* Ära looda automaatsele andmekogumisele liialt

Sellise süsteemi ehitamine võib minna tunduvalt kallimaks kui inimjõul kogumine. Hea korralduse puhul võtab andmete edastamine vaid kuni paar protsenti programmeerijate tööajast.

* Tee andmete saatmine lihtsaks

Kui sellega probleeme ei tule, siis on andmete kogumine palju meeldivam. Hoolitse, et oleks kergesti võimalik kontakti võtta andmete kogujaga ning temalt vajadusel nõu saada.

* Kasuta andmete hoidmiseks olemasolevaid vastavaid programme

Ise selliste välja töötamine läheb liialt kulukaks.

* Andmed on nagunii vigased ja puudulikud

Sest ikka juhtub, et kusagil läks mõni number valesti, võeti ajapuuduses või pettekujutluses huupi või jäeti lihtsalt saatmata. Osalt sellepärast, et andmete kogumist ei pea programmeerijad oma põhitööks ega vaevu saadetavaid teateid üle kontrollima. Ning inimesed teevad ikka ja alati vigu.

## *Ülesandeid*

- Koosta ruutvõrrandit lahendav programm.
- Hoolitse, et see töötaks korralikult ka kõikvõimaliku vigase sisestuse korral.
- Paiguta võimalikult iga toiming eraldi alamprogrammi (kokku neid vähemalt 5)
- Testi iga alamprogrammi tööd nõnda, et iga koodirida saaks vähemalt korra läbitud.
- Leia algandmed võimalike piirjuhtude tarbeks ning kontrolli programmi tööd nende puhul, samuti mõlemal pool piiri.

- Kommenteeri kõik oma funktsiooni, nende ülesanded, sisendparameetrid ja väljundväärtused.
- Lisa oma koodi paar apsakat, märgi need üles.
- Vahetage oma koodid pinginaabritega. Märkige üles kõik koodis leiduvad parandamist vajada võivad kohad koos ettepanekutega.
- Võrrelge koos kummagi koode ja ettepanekuid. Jälgige, kas lisatud vead leiti üles. Hinnake, kui suure osa moodustasid lisatud vead kogu paranduste hulgast.