Ohjelmistokehitys NASA:n miehitetyissä avaruusohjelmissa

Mikko Hämäläinen
mikko.hamalainen@helsinki.fi

Helsinki 20.3.2005

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

NASA:n miehitettyyn avaruusohjelmaan kuuluu Mercury, Gemini, Apollo, Skylab ja sukkula projektit vuosien 1960 ja nykypäivän välillä. Ensimmäistä vaihetta lukuun ottamatta aluksissa on ollut mukana tietokoneet, joilla on ollut merkittäviä tehtäviä navigoinnista elinympäristön ylläpitoon. Miehitettyjen avaruuslentojen alkuvaiheessa NASA:lla oli hyvin vähän kokemusta monimutkaisten tosiaikajärjestelmien kehityksestä. Ohjelmistojen kehitys insinööritieteenä ja isojen ohjelmistoprojektien johtaminen olivat vielä lapsenkengissään, mikä näkyi kehityksen vaikeutena ja venyneinä aikatauluina.

Tässä seminaarityössä käsitellään ohjelmistokehitystä NASA:n ensimmäisillä miehitetyillä avaruuslennoilla. Painopisteenä ovat erityisesti Apollo-projektin kuulennot, mutta lisäksi käsitellään myös aikaisempaa Gemini sekä myöhempää Skylab-projektia. Alustus käsittelee aikajärjestyksessä projektien ohjelmistokehitystä, minkälaisia ongelmia niissä käsiteltiin, mitä niistä opittiin ja miten ne kehittivät tietojenkäsittelytiedettä.

2 Gemini

Mercury-projektin aikana kävi nopeasti selville, että tulevia avaruuslentoja ei voitaisi tehdä ilman aluksella olevaa omaa tietokonetta, joka kykenisi itsenäiseen prosessointiin. Alusta tulisi voida ohjata ilman lennonjohdon kontrollia, koska jo Gemini-projektissa aluksen lentorata kattoi alueita, jotka eivät olleet lennonjohdon kontrollisäteen sisällä. Erityisen tärkeitä tehtäviä olivat itsenäinen paluu ilmakehään, varajärjestelmänä toimiminen kantoraketille sekä käynnistysvaiheen automatisointi. Kaikki tämä oli tarkoitus tehdä aluksen oman digitaalisen tietokoneen avulla ja ohjelmiston valvonnassa [Tom88].

2.1 Ohjelmistokehitys

Vaikka ohjelmistojen tuottaminen insinööritieteenä oli tunnettu jo 50-luvulta lähtien, oli ohjelmistokehitys vielä toisarvoisessa asemassa laitteiston kehitykseen nähden. Laitteistot olivat kalliita ja niiden toteuttamisen prosessit olivat yleisemmin tunnettuja kuin ohjelmistojen, jotka kirjoitettiin vielä erikoistuneilla symbolisilla

konekielillä. Ohjelmistoprosessien, dokumentoinnin ja strukturoitujen ohjelmistojen kulttuuri ei ollut vielä syntynyt [Tom88].

NASA:ssa selvisi kuitenkin nopeasti, että ohjelmiston kehittäminen Gemini-projektin puitteissa vaati prosessin tarkemman määrittelyn ja paremman kommunikaation eri ryhmien välillä. Tässä vaiheessa nähtiin tarpeelliseksi määritellä miten tarkat spesifikaatiot, versionhallinta, virheiden käsittely ja ohjelmiston jakaminen pienemmiksi moduuleiksi hoidetaan.

Moduulien tärkeys kävi erityisesti selväksi, koska ei ollut järkevää toteuttaa jokaista tehtävää varten täysin uutta ohjelmistoa. Suuri osa ohjelmakoodista käsitteli tehtävän perusosuuksia, kuten nousu ja laskeutuminen, jotka toistuisivat tehtävästä toiseen, joten oli järkevämpää pakata nämä uudelleenkäytettäviin moduuleihin. Näin pystyttiin myös vähentämään ongelmia rajallisen muistikapasiteetin kanssa ja ryhmien tehtävät jakautuivat selkeämmiksi kokonaisuuksiksi.

Ohjelmistoa pyrittiin selkeyttämään rakentamalla se moduuleista, mutta varsinainen ohjelmointi oli vielä hyvin monimutkaista käytettyjen työkalujen takia. Vaikka esimerkiksi FORTRAN oli jo keksitty vuosia aikaisemmin, ei NASA:lla uskottu, että koneen kääntämä koodi voisi olla yhtä tehokasta kuin ihmisen kirjoittama. Tästä syystä Geminin ohjelmissa käytettiin 16-käskyn symbolista konekieltä, mikä johti siihen, että ohjelmat olivat monimutkaisia ja kaavat jouduttiin avaamaan hyvin pieniin osiin [Tom88]. Tämä vahvisti mielipidettä ohjelmoinnista mystisenä taitona, hyvin järjestellyn tieteen sijasta.

Gemini-projekti antoi NASA:lle tärkeää tietoa ohjelmistojen vaatimuksista miehitetyillä avaruuslennoilla. Sen aikana muodostettavia projekti-, spesifiointi- ja verifiointimalleja käytettiin erityisesti Apollo-projektissa parantamaan ohjelmistojen laatua ja projekti toimi merkittävänä alkusykäyksenä ohjelmistojen tuottamiselle insinööritieteenä.

3 Apollo

Apollo-projekti käynnistyi alustavasti vuonna 1961 Mercury-projektin ollessa vielä käynnissä [Com89]. Vuosien 1965 ja 1966 Gemini lennot toimivat tärkeinä opetteluvaiheina NASA:lle ja niiden perusteella saatettiin määritellä entistä paremmin alukselle asetettavia vaatimuksia. Myös tietokonelaitteistolla ja ohjelmistolla piti pystyä

ratkaisemaan täysin uudenlaisia ongelmia.

Matka kuuhun vaati aluksen tietokoneilta huomattavasti enemmän autonomisuutta verrattuna kiertoradalla lentävään Geminiin. Syitä aluksen omalle tietokoneelle olivat maasta tulevan häirinnän estäminen, pitkien tehtävien mahdollistaminen ja raskaan laskennan poistaminen lennonjohdosta, jotta voitaisiin pitää mahdollisena useampi yhtäaikainen lento. Tärkein syistä oli kuitenkin 1.5 sekunnin viive maan ja kuun välillä, joka teki mahdottomaksi ohjata laskeutumista kuuhun maasta käsin [Tom88]. Loppujen lopuksi Apollon tietokoneesta ei tullut niin autonominen kuin alun perin suunniteltiin, mutta esimerkiksi kuuhun laskeutuminen hoidettiin oman tietokoneen varassa.

NASA:lla ei ollut kokemusta niin monimutkaisista ohjelmistoprojekteista, mitä vaadittiin Apollon ohjaus- ja navigointijärjestelmien kehittämiseen. Tästä syystä laitteiston ja ohjelmiston kehitys annettiin MIT:lle (Massachusetts Institute of Technology), jolla oli aikaisempaa kokemusta rakettiprojekteista. MIT:llä ei ollut kuitenkaan aikaisempaa kokemusta ohjausjärjestelmän laajuisista tosiaikajärjestelmistä, joten ohjelmiston kehittäminen oli hidasta ja siinä oli suuria ongelmia aina ensimmäisiin Apollo-lentoihin asti.

3.1 Ohjelmistoprojekti

Apollo-projektin kannalta merkittävin ohjelmistojärjestelmä oli ohjaus- ja navigointijärjestelmä. Sen sijaan, että Apollo olisi laukaistu suoraan maasta kuuhun, mikä olisi tehnyt navigointijärjestelmästä hyvin monimutkaisen, matka jaettiin kolmeen osaan: nousuun kiertoradalle, matkaan kuun kiertoradalle ja laskeutumiseen kuuhun [Tom88]. Tämän ansiosta myös ohjelmisto voitiin jakaa helpommin hallittaviin osiin. Alijärjestelmille perustettiin ryhmiä sekä NASA:lla ja MIT:llä ja varsinkin MIT:llä osa kehitystyöstä myös ulkoistettiin kun työn määrä kasvoi odotettua suuremmaksi.

Ohjelmiston systemaattinen kehitys ja ohjelmistoprojektin määrittely ja hallinta olivat vielä suhteellisen tuntemattomia käsitteitä Apollo-projektin alkuvaiheessa. Gemini-projekti ei ollut tässä vaiheessa vielä niin pitkällä, että sieltä olisi saatu kokemusta isompien ohjelmistojen kehityksestä ja sekä NASA että MIT pitivät ohjelmiston kehitystä edelleen toisarvoisena laitteiston kehityksen rinnalla.

NASA ja MIT olivat kuitenkin hyvin tietoisia kuinka ohjelmiston virheettömyys

ja luotettava toiminta tulisivat olemaan tärkeä osa Apollon-menestyksestä. Tämän takia NASA:lla perustettiin kolme valvontalautakuntaa ja määriteltiin ohjelmiston hyväksymisproseduurit. Näiden tarkoituksena oli kontrolloida ohjelmiston kehitystä ja varmistaa, että määritelty spesifikaatio tulisi täytettyä ilman, että vaatimukset muuttuisivat matkalla [Tom88]. Lautakunta varsinaisen ohjelmiston kehityksen valvomista varten perustettiin kuitenkin vasta 1967 navigointiohjelmiston kehityksessä syntyneisiin ongelmiin.

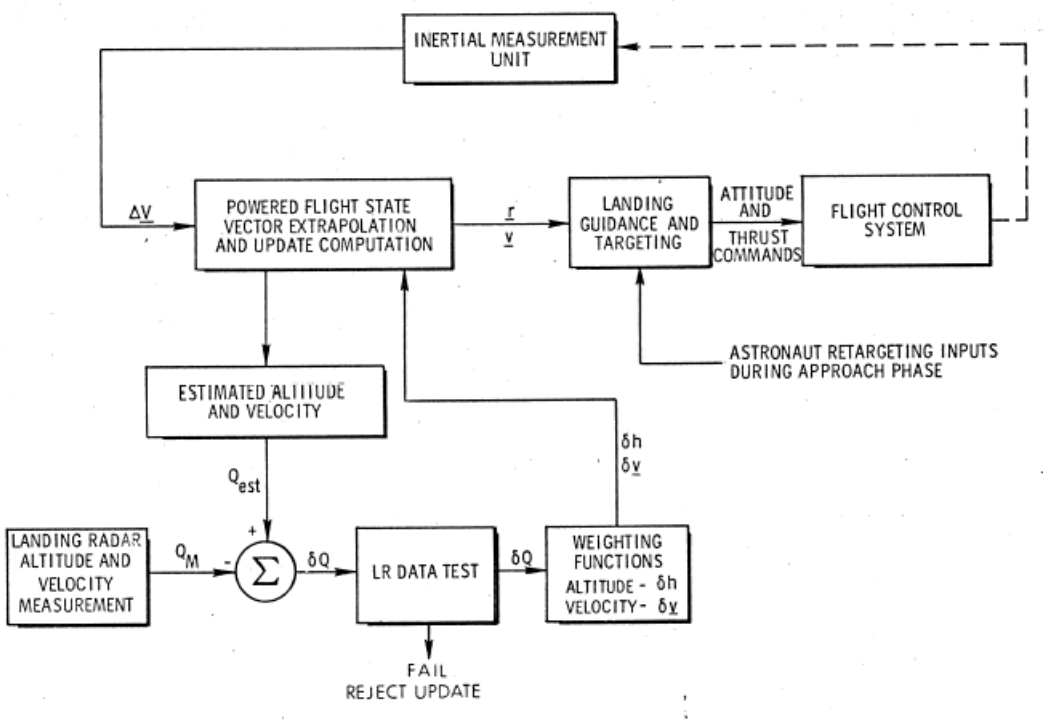
3.2 Spesifiointi ja verifiointi

Apollo-projektin aikainen *Apollo guidance software development and verification plan* asetti tiukat vaatimukset ohjelmiston tuottamiselle. Sen mukaan tarkasti määritelty spesifikaatio oli avain menestyksekkäälle ohjelmistolle. Spesifikaation tuli määritellä ohjelmiston *"vaatimukset, funktiot, rajapinnat, kaavat ja ohjelmoitava logiikka"* [WKG67]. Spesifikaatio voitaisiin tämän jälkeen jakaa eri alihankkijoille ja he pystyisivät sen mukaan toteuttamaan oman osansa ohjelmistosta.

Tämä suunnitelma esitettiin kuitenkin ohjelmiston kehitystä valvovalle komitealle vasta marraskuussa 1967, jolloin ensimmäisen Apollo lennon olisi pitänyt jo tapahtua, jos olisi seurattu alkuperäistä aikataulua. Sitä ennen spesifioinnissa ja verifioinnissa oli ollut suuria ongelmia.

Ohjaus- ja navigointijärjestelmän vaatimuksia toimitettiin NASA:lta MIT:lle jatkuvana virtana 60-luvun alusta, läpi Gemini-projektin aina lähestyvään ensimmäiseen lähtöön asti. Vaatimukset vaihtelivat kaavoista funktiodigrammeihin (kuva 1) ja sanallisiin selvityksiin. Tämä muodostui ongelmaksi, koska NASA ei pystynyt tarkalleen määrittelemään kaikkia vaatimuksia ja määritellyt vaatimukset vaihtelivat yksityiskohtien tasoltaan. Tämä pakotti MIT:n ohjelmoijat otaksumaan liikaa ohjaus- ja navigointijärjestelmän toiminnasta, mikä johti siihen, että valmiita ohjelmistokomponentteja oli hyvin vaikea verifioida spesifikaatiota vastaan [Tom88].

Avaruusaluksissa suoritettavien ohjelmistojen verifiointi on yleisesti hyvin vaikeaa johtuen siitä, että ympäristöä kuten nollapainovoima, jossa ohjelmistot toimivat on vaikea simuloida. Apollo ei ollut tästä poikkeus, mutta sille suoritettiin hyvin kattava joukko yksikkö, integraatio ja simulaattorilla suoritettuja funktionaalisuustestejä. Simulaatiotestejä suoritettiin tietokonelaiteiden toimittajilla ja NASA:lla Houstonissa [WKG67].



Kuva 1: Funktiodiagrammi kuuhun laskeutumisesta [JG71, s. 188]

Yksikkö ja integraatiotestejä suoritettiin MIT:llä ja NASA:lla, mutta myös tässä näkyi, ettei MIT:llä ollut kokemusta ja kiinnostusta formaaliin ohjelmistokehitykseen. Vasta NASA:n painostuksen jälkeen MIT suostui toimittamaan testaussuunnitelmia NASA:lle joiden perusteella tehtiin neljävaiheinen testaussuunnitelma. Ohjelmistoilta vaadittiin 1000-1200 testiajoa ennen kuin ne olivat täysin virhekorjattuja [Tom88].

Vuonna 1966 oltiin kuitenkin tilanteessa, jossa kaikissa MIT:ltä saaduissa moduuleissa oli virheitä ja moduuleita ei ollut yksikkötestattu ennen integrointia, joten virheiden sijaintia oli mahdotonta tietää. Tämän lisäksi virheiden paikannusta hidasti testausraporttien puute, joita MIT ei toimittanut ilman eri pyyntöä. Ajan käydessä vähiin vuoden 1967 ensimmäisen lennon lähestyessä, projektissa hyväksyttiin, että ohjelmistoon jäisi virheitä, jotka astronautit ja lennonjohto joutuisivat kiertämään. Vuonna 1967 suoritetuissa paluupolttosimulaatioissa oli 138 sekunnin ero peräkkäisten ajojen välillä. Tämä tarkoitti, että ohjaustietokoneen tuloksiin ei voitu luottaa ja astronauttien tulisi syöttää lennonjohdon antamat tulokset käsin koneelle. Tähän ei kuitenkaan koskaan päädytty, koska harjoituksessa tapahtunut onnettomuus siir-

si ensimmäisen lennon päivää, missä ajassa ohjaus- ja navigointiohjelmistot saatiin korjattua [Tom88].

Tämän jälkeen otettiin käyttöön tarkemman verifiointiproseduurit, joissa ohjelmisto kävi läpi ensin asiakkaan hyväksymiskatselmuksen (*Customer Acceptance Readiness Review*), jonka jälkeen lopulliset spesifikaatiot, dokumentit, diagrammit ja kaavat julkistettiin ja ohjelmiston testaus saatettiin loppuun. Lopullinen ohjelmiston hyväksyminen tehtiin loppukatselmuksessa (*Flight Readiness Review*), jonka jälkeen se oli valmis aluksen koneille asennettavaksi [WKG67]. Verifiointi ja testaussyklit toistettiin jokaisen ohjelmistoon tehtävän muutoksen jälkeen ja lopulta myös NASA:ssa voitiin olla varmoja ohjelmiston oikeasta toiminnallisuudesta.

3.3 Ohjelmisto

Apollo-projektille tuotettiin ohjelmistoja useisiin tehtäviin. Tietokonejärjestelmien tärkeimpänä tehtävänä oli huolehtia ohjauksesta ja navigoinnista, ja muita vastuualueita olivat Saturn-raketin varajärjestelmänä toimiminen, lennon vaiheiden ohjaaminen ja virheiden käsittely.

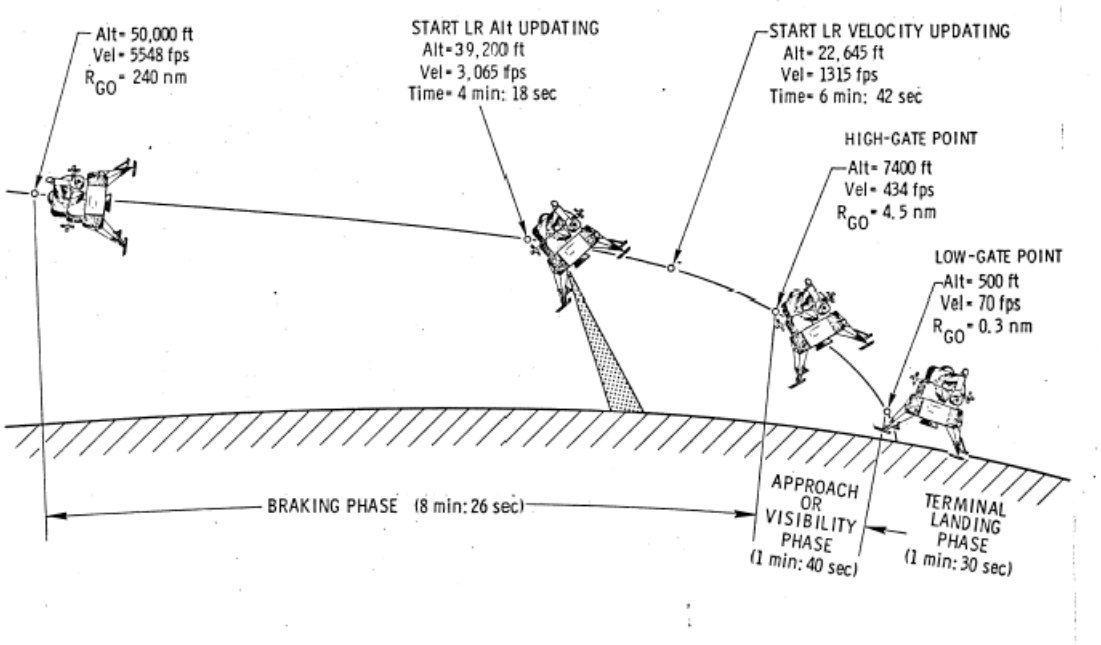
Toisin kuin NASA Gemini-projektissa, MIT ei enää käyttänyt Apollo-projektissa symbolista konekieltä ohjelmistojen toteuttamiseen vaan kehitti sitä varten korkeamman tason kielen. Kieli oli virtuaalikoneessa tulkattu ja sisälsi 128 käskyä. Tämä oli suuri parannus Gemini-projektin 16 käskyyn verrattuna ja helpotti selvästi ohjelmiston kehittämistä, koska kaavoja ei tarvinnut jakaa niin pieniin osiin ja ohjelmoijilla oli käytössää ilmaisuvoimaisempi kieli. Ohjelmat olivat tulkatun kielen takia hitaampia suorittaa, mutta vaativat vähemmän tilaa muistissa.

Ohjaus- ja navigointi oli Apollon ohjaustietokoneen (AGC, *Apollo Guidance Computer* vastuulla. Sen käyttöjärjestelmä toimi prioriteettikäyttöjärjestelmä, joka valitsi töitä ajoon niiden prioriteetin perusteella. Tätä varten käyttöjärjestelmällä oli kaksi jonoa: seitsemän paikan Executive-jono ja 9 paikan Waitlist-jono. Waitlist-jono oli töille, jotka veivät enintään 4 ms ja sitä pidemmät työt laitettiin Executive-jonoon. Käyttöjärjestelmä ei käyttänyt keskeytyksiä vaan tarkisti 20 ms sekunnin välein oliko jonoihin tullut korkeamman prioriteetin töitä [Tom88].

Kyseessä oli monimutkainen tosiaikajärjestelmä, joka hallitsi aluksen nopeuden muutoksia, aluksen ja kuumoduulin kääntämistä, seurasi astronauttien syötteitä, virheil-

moituksia, tietoliikennettä ja päivitti käyttöliittymää [MB67]. Ohjaus- ja navigointi toimivat yhteistyössä siten, että ohjaus sai navigoinnilta parametreina nopeuden ja suunnan muutoksia. Näiden yhdistelmänä toimiva autopilotti saatettiin ohjelmoida suorittamaan monimutkaisiakin operaatioita.

Yksi tärkeimmistä ohjelmistoista oli alijärjestelmä, joka ohjasi kuuhun laskeutumista. Se toimi yhteistyössä astronautin kanssa tuoden kuumoduulin kuun pinnalle niin, että lopullisessa kosketuksessa ei ole enää kuin nimellinen nopeus. Jotta tämä kaikki saataisiin aikaan, oli ohjelmistolle asetettu tiukkoja vaatimuksia. Niitä olivat tehokas polttoaineen käyttö, moduulin kääntäminen niin, että astronautilla oli näköhavainto laskeutumisalueeseen vähintään 75 sekunnin ajaksi ja laskeutumisvaiheen hallinta niin, että astronautti saattoi milloin vaan siirtyä käsiohjaukseen [JG71].



Kuva 2: Kuuhun laskeutumisen vaiheet [JG71, s. 182]

Laskeutuminen suoritettiin vaiheissa (kuva 2), joista ensimmäinen oli jarrutus. Tässä vaiheessa ohjaustietokone toi moduulin oikeaan kohtaan kiertoradalla ja suoritti hidastuspolton, jolla vauhti tiputettiin nopeudesta 1691 ms/s nopeuteen 132 ms/s. Tämän jälkeen tuli vaihe, jossa laskeutumisaluetta pidettiin ohjelmiston ohjauksessa näkyvänä siten, että kuumoduulin ikkunaan piirretty tähtäin osoitti koko ajan laskeutumisalueelle. Lopuksi suoritettiin varsinainen laskeutuminen, jonka astronautti saattoi siis vielä keskeyttää [JG71]. Kaikki laskeutumisen vaatimukset toteutettiin

spesifikaation mukaan ja ne testattiin käytännössä heti Apollo 11 lennolla, jolla Neil Armstrongin piti vaihtaa viime hetkessä laskeutumispaikkaa ja ohjata kuumoduuli alas manuaalisesti alkuperäisen laskeutumispaikan oltua täynnä kraattereita.

3.4 Apollo-projektista opittua

Apollo-projektin alusta lähtien oli selvää, että sekä MIT ja NASA olivat hyvin tottumattomia projektissa vaadittavien ohjelmistojen kehityksessä. Tämä näkyi erityisesti MIT:n kehittämässä ohjaus- ja navigointijärjestelmässä, joka oli NASA:n mukaan liian monimutkainen ja vaikeasti verifioitavissa.

Ongelmia tuottivat erityisesti muuttuvien vaatimusten ja sitä myötä kasvaneen koodin aiheuttama muistin vähyys sekä kommunikaatio-ongelmat Houstonin ja Bostonin välillä. Ohjelmistoa pyrittiin yksinkertaistamaan niin, että se veisi vähemmän muistia ja olisi helpommin toteutettavissa. Tämä tarkoitti kuitenkin polttoaineen kulutuksen kasvua, koska monet tehtävistä jätettiin astronautin tuntuman varaan [Tom88].

Gemini-projektin valmistuttua siitä vapautui ohjelmistokehittäjiä, joilla oli nyt kokemusta menestyksekkäästä avaruusalusten ohjelmistokehityksestä [Hoa76]. Ohjelmistojen laadun parantamiseksi perustettiin lautakuntia, jotka selkeyttivät sekä vaatimuksia että ohjelmiston kehitysprosesseja.

Merkittävin uudistus oli, että ohjelmistokehitys nostettiin samalle tasolle laitteistokehityksen kanssa ja sille päätettiin antaa tulevaisuudessa vaadittavat resurssit. Ohjelmistoja ei enää pidettäisi valmiina vain koska aikataulu määräsi niin vaan niiden laatu olisi ensisijalla. Muita projektista opittuja asioita olivat dokumentaation ja hyvän kommunikaation tärkeys, vaatimusten ja spesifioinnin hallinta sekä kattava testaus.

Apollo-projektin ohjelmistojen toteutus oli valtava tehtävä ja loppujen lopuksi pelkän ohjaus- ja navigointiohjelmiston tekemiseen meni yli 1400 miestyövuotta [Hoa76].

4 Skylab

Gemini ja Apollo-projekteja oli leimannut hyvin vahvasti uuden opettelu ja ihmisen vieminen kuuhun. Näiden projektien johdosta NASA:lla oli opittu hyvin paljon lait-

teiston ja ohjelmiston kehityksestä ja opitut asiat saatettiin ottaa käyttöön Apollon jälkeisessä Skylab-projektissa.

Skylab-projekti erosi tehtävänasettelultaan vahvasti aikaisemmista projekteista, koska sen tarkoituksena oli perustaa tutkimusasema maata kiertävälle radalle. Lisäksi NASA päätti ottaa käyttöön IBM:n System 360 sarjaan perustuvan tietokoneen erikoiskoneen sijaan [Tom88].

Alusta lähtien oli ohjelmistoprojekti päätetty hoitaa oikein, mikä tarkoitti, että kehitysryhmät ryhmät pidettiin pieninä ja kommunikaatioon panostettiin. Tämän lisäksi ohjelmistomoduulit määriteltiin tarkasti ja prosessi jaettiin neljään selkeään osaan vuosien 1970 ja 1973 välillä. Projektissa ei kuitenkaan käytetty korkeamman tason kieltä kuten Apollo-projektissa vaan symbolista konekieltä, jolle laadittiin laaja makrokirjasto.

Ohjelmistosta laadittiin kaksi versiota: 16K pääohjelmisto ja siitä johdettu pienempi 8K:n varajärjestelmä. Pääjärjestelmä oli prioriteettipohjainen, keskeytyksiin kykenevä käyttöjärjestelmä, joka ajoi varsinaisia ohjelmistomoduuleita. Se pystyi keskeyttämään minkä tahansa tehtävän, jos korkeamman prioriteetin työ tuli jonoon. Tämä oli selvää edistystä Apollon työlistojen tarkkailusta.

Järjestelmän testaukseen ja verifiointiin panostettiin. Apollo-projektin tavoin ohjelmistolle suoritettiin tarkastukset eri lautakunnissa sekä tiukat yksikkö-, integraatio- ja simulaatiotestit kolmessa eri simulaattorissa. Digitaaliset simulaattorit olivat jo hyvin edistyneitä niin, että virheen löytyessä uusi koodi saatettiin siirtää simulaattoriin ja jatkaa suoritusta keskeytyneestä kohdasta. Verifiointiprosessin päätteeksi NASA:lla voitiin olla hyvin varmoja ohjelmiston oikeasta toiminnallisuudesta [Tom88].

Skylab-ohjelmistoprojektin ensimmäisenä tavoitteena oli luonnollisesti luoda luotettava ja tehokas ohjelmisto avaruusaseman tietokoneisiin. Toissijaisena tavoitteena oli kuitenkin toteuttaa ohjelmistoprojekti kaikkien sääntöjen mukaan ilman ongelmia. Kummatkin tavoitteet saavutettiin hyvin ja Apollo-projektin opit oli saatu toteutettua käytäntöön.

5 Yhteenveto

NASA on suorittanut miehitettyjä avaruuslentoja 60-luvun alusta nykypäivään. Gemini-projektista lähtien aluksilla on ollut mukana tietokoneet, jotka ovat hallin-

neet yhä enemmän lennon eri vaiheita. Gemini, Apollo ja Skylab tarjosivat NASA:lle ainutlaatuisen oppimiskokemuksen laajojen tosiaikajärjestelmien laatimisesta.

Gemini-projekti toi esille ohjelmistoprosessin tärkeyden. Vaatimusten määrittely, perinpohjainen testaus ja koodin modularisointi helpottivat laajojen ohjelmistojen rakentamista. Ohjelmistot nähtiin kuitenkin vielä koneiden sivutuotteina, eikä niiden toteuttamista nähty varsinaisena insinööritieteenä.

Apollo-projekti aloitettiin monella tavoin yhtä tyhjältä pöydältä kuin Gemini-projektikin, koska Gemini-projekti kulki sen kanssa pitkään rinnakkain, eikä sen tuloksia saatu käyttöön vielä pitkään aikaan. Lisäksi suuri osa Apollo-projektin ohjelmistotuotannosta oli annettu MIT:lle, jolla ei myöskään ollut kokemusta näin laajojen ohjelmistojen tuotannosta. NASA:n tiedot päivittyivät Gemini-projektin kuluessa, mutta MIT kärsi pitkään vaikeuksista ohjelmiston tuotannossa.

Apollo-projekti opetti kuinka tärkeää tehokas kommunikointi eri ryhmien välillä on. Kun Gemini-projektin tieto alkoi levitä MIT:n projekteihin ja formaalit ohjelmistoprosessit saatiin käyttöön, ohjelmiston taso parani ja lopulta aluksissa toimi hyvin testatut ja virheettömät ohjelmistot. Apollo-projektin tuotti myös teknisesti ansiokkaita uudistuksia. Korkeamman tason kieliä ei enää pidetty liian hitaina kriittisten järjestelmien ohjelmointiin ja tosiaikajärjestelmien ymmärrys kehittyi ohjaus- ja navigointijärjestelmien myötä.

Erityisesti Apollo-projektissa voidaan nähdä miten isojen ohjelmistoprosessien hallitsemista opeteltiin käsi kädessä projektin etenemisen kanssa. Ohjelmiston tekeminen jaettiin tarkasti elinkaarimalliin, jota käytetään vielä tänäkin päivänä. Vaatimusten määrittely, ohjelmiston tarkka spesifiointi, hallittu toteutus, testaus, ylläpito ja dokumentointi nousivat keskeisiksi tekijöiksi. Ohjelmisto itsessään nousi myös omaksi alueekseen, jonka tärkeys ymmärrettiin projektitasolla.

Skylab-projektissa NASA saattoi pistää käytäntöön Gemini ja Apollo-projekteissa opittuja asioita. Edelleen ongelmana oli vaatimusten muuttumisesta johtunut muistin vähyys, mutta projektitasolla ohjelmistonkehitys oli huomattavasti edistyneempää.

Ohjelmistojen merkitys ei vähentynyt Skylab-projektin jälkeenkään vaan sukkulaprojektissa ohjelmistoilla oli entistä suurempi valta aluksen kaikissa järjestelmissä. Miehitetyt avaruuslennot edistivät tiedettä monilla tavoilla ja niihin käytetyt resurssit olivat valtavat. Apollo-projekti edusti 60-luvulla noin 30 prosenttia NASA:n

määrärahoista. Koko projektin budjetti oli aikanaan 2.8 prosenttia yhden vuoden bruttokansantuotteesta, mikä vastaa 170 miljardia dollaria nykyrahassa [Con04]. Vertailun vuoksi NASA:n tämän vuoden budjetti on 16.3 miljardia dollaria. Projektien tarjoamat opit edistivät tietojenkäsittelytiedettä ja nostivat käytännön ohjelmistotuotannon yhdeksi sen tärkeäksi osa-alueeksi.

Lähteet

Com89 Compton, W. D., Where No Man Has Gone Before: A History of Apollo Lunar Exploration Missions. *NASA History Series*, NASA Special Publication-4214. URL `http://www.hq.nasa.gov/office/pao/History/SP-4214/contents.ht%ml`.

Con04 Congressional Budget Office, toimittaja, *A Budgetary Analysis of NASA's New Vision for Space Exploration.* The Congress of the United States, syyskuu 2004. URL `http://www.cbo.gov/ftpdocs/57xx/doc5772/09-02--NASA.pdf`.

Hoa76 Hoag, D. G., *The History of Apollo On-Board Guidance, Navigation and Control.* Charles Stark Draper Laboratory Massachusetts Institute of Technology, syyskuu 1976. URL `http://hrst.mit.edu/hrs/apollo/public/archive/1711.pdf`.

JG71 Johnson, M. ja Giller, D., *MIT's role in project Apollo, Vol V The software effort.* Charles Stark Draper Laboratory Massachusetts Institute of Technology, maaliskuu 1971. URL `http://hrst.mit.edu/hrs/apollo/public/archive/1137.pdf`.

MB67 Martin, F. H. ja Battin, R. H., Computer-Controlled Steering of the Apollo Spacecraft. *AIAA, Guidance, Control and Flight Confrence*, Vol 4., sivut 400–407. URL `http://www.hq.nasa.gov/office/pao/History/SP-4214/contents.ht%ml`.

Tom88 Tomayko, J. E., Computers in Spaceflight – The NASA Experience. *NASA History Office*, NASA Contractor Report 182505. URL `http://www.hq.nasa.gov/office/pao/History/computers/Compspace%.html`.

WKG67 Wilson, R., Kayton, M. ja Gilbert, D., *Apollo Guidance Software Development and Verification Plan.* NASA Manned Spacecraft Center, lokakuu 1967. URL `http://hrst.mit.edu/hrs/apollo/public/archive/1695.pdf`.