Smidig epistemologi

Hans Georg Schaathun

NTNU — Noregs Teknisk-Naturvitskaplege Universitet
georg@schaathun.net

Samandrag Praktiske fag kan somme tider framstå som ein pariakaste ved universiteta. Vitskapsideala som er utvikla over fleire hundre år inneber at det er like viktig å vita korleis me kan vita, som faktisk å vita. Kunnskap skal vera objektiv og etterprøvbar og helst kontekstfri. Praktiske fag vert dregne i eit dilemma mellom vitskaplege ideal og praktisk relevans. Mange profesjonsfag har etter kvart ein rik litteratur om dette problemet, der ein utviklar epistemologiar som er likeverdige med dei konvensjonelle vitskapane, men som tek omsyn til skilnaden mellom fag som skildrar verda og fag som endrar verda. Dette har i liten grad vore drøfta innanfor programutvikling, og her skal me sjå på korleis eksisterande metode og epistemologi frå design kan vera relevant for programutvikling generelt og smidige metodar spesielt. Vidare skal me drøfta korleis denne innsikta er relevant for undervising og vurdering.

1 Innleiing

Konflikten mellom teori og praksis tek mange formar, og likeeins dilemmaet mellom industri og akademia. Akademia har, gjennom snart 2500 år, lagt stor vekt på *epistemologi*, dvs. læra som skil mellom det me veit og det me trur. I tradisjonelle universitetsfag har det vore like viktig å vita korleis me kan vita, som faktisk å vita. Nye vitskapsdisiplinar har brukt lang tid på utvikla sine eigne epistemologiar og forskingsmetodar.

Dei praktiske faga eller profesjonsfaga har andre forståingar av kva som er rette og gode svar på faglege problem. Når desse faga so vert universitetsfag møter me ei utfordring. Universitetet krev epistemologiar slik at me objektivt kan vita kva som er rett og god kunnskap. Samstundes må me ivareta fagtradisjonen og hensikta bak dei profesjonane som me utdannar til. Herbert Simon drøfta dette dilemmaet for meir enn femti år sidan. Den gamle skulen visste ikkje korleis dei kunne utdanna profesjonsutøvarar på det intellektuelle nivået som sømmer seg for eit universitet. Den nye skulen abdiserte frå ansvaret for utdanning i profesjonelle kjerneferdigheiter, skreiv han (Simon, 1969, s. 57). Her i landet vert utfordringa stadig reaktualisert ved at stadig fleire gamle høgskulemiljø må finna seg ein ny identitet som del av eit universitet.

Dette dilemmaet vert særleg tydeleg i IKT-faga. Dette er ikkje berre fordi IKT er eit ungt fag. Viktigare er det at det ikkje er eitt fag, men ei rad med fag som spring ut av ulike fagtradisjonar og dermed ulike epistemologiar. Me finn IKT-utdanningar innanfor både matematiske, samfunnsvitskaplege, ingeniørfaglege, humanistiske og handelsfaglege miljø, utan at der alltid er klare

skilnader i læringsutbyta eller i yrka som dei utdannar til. Dette gjer at ulike epistemologiar og yrkesidentitetar møtest innanfor eit mikrokosmos (som UDIT) og ikkje berre under den store universitetsparaplyen. Diskusjonen om kva som faktisk er god og sikker kunnskap om IKT kunne med fordel ha gått djupare.

Epistemologi er ikkje berre ein reint akademisk konstruksjon, men sjølve grunnlaget for vurdering og evaluering av løysingar (og produkt). Når me ser på yrket som program- eller systemutviklar, treng me epistemologi i tre ulike roller. Som arbeidstakar må ein kunna vita om løysingane som vert utvikla er gode og korrekte. Som undervisarar treng me objektive kriterium for å vurdera studentane sitt arbeid. Kriteria må vera robuste nok til å tola klagesensur. Sist men ikkje minst må studentane vera i stand til å vurdera sitt eige arbeid. Dette er ikkje berre fordi dei seinare treng å vurdera seg sjølv som arbeidstakarar, men òg fordi eigenvurdering er eit viktig element i aktiv læring (jf. Nicol & Macfarlane-Dick, 2006).

Det er naturleg å sjå epistemologien i samanheng med utviklingsmetodikken. I kva grad kan me seie at metoden vår leier til epistemologisk sikker kunnskap? Innanfor programutvikling er dei tradisjonelle fossefallsmetodane i stor grad erstatta av sokalla *smidige metodar*. Dette er ei sekkenemning for mange metodar som har éin ting til felles. Dei forkastar det gamle, strenge planleggingsregimet, der nitidige planar skulle sikra objektiv vurdering av måloppnåing. Korleis dei smidige metodane kan gje objektiv kunnskap er det diverre skrive lite om.

I denne artikkelen skal me drøfta to forskingspørsmål.

1. Kva epistemologiske innsikter er med på å tvinga oss til å forlata fossefallsmetodikken?
2. Kva grep kan eller bør me ta i smidige metodar for å sikra epistemologisk sikker kunnskap?

Me vil starta med ein generell analyse, der me ser programutvikling som eit spesialtilfelle av design eller profesjonsfag. Utgangspunktet vårt er metodetradisjonane som byggjer på arbeidet åt Herbert Simon og Donald Schön. Deretter, frå avsnitt 4 vil me sjå spesifikt på smidige metodar og samanlikna med den generelle analysen. Avsnitt 5 drøftar kva konsekvensar dei epistemologiske vurderingane bør ha for undervisinga i program- og systemutvikling. Eit lite litteratursamandrag er plassert til slutt for å peika på andre vinklingar enn dei som studien er bygd på.

Objektet for studien er den kognitive prosessen i programutvikling. Dvs. at me primært spør korleis utviklaren som einskildperson kan vita kva som er gode svar innan ein bestemt prosjektkontekst. Dette er ikkje den einaste relevante vinklinga, men i eit utdanningsperspektiv er det ei naudsynt vinkling fordi studenten skal ta eksamen som einskildperson og må kunna vita kva som er godt arbeid. Det er òg relevant som yrkesutøvar, fordi det er ein føremon å kunna skilja godt arbeid frå dårleg før ein delar det med prosjektgruppa. Likevel har me ikkje funne nokon liknande studie i litteraturen

Vonleg tilfører dette noko nytt som lesaren kan ha nytte av. Me skal likevel understreka at me ikkje søkjer eitt endeleg svar som skal vera gyldig for alle pro-

sjekt, utdanningar eller karrierar. Dette vert klart etter kvart, når me definerer *uskikkelege problem*.

2 Design eller vitskap

Vitskapen er gjerne sett som den høgaste formen for kunnskap i vår kultur. Han fortel oss korleis verda *er*. Jobben som programutviklar er derimot å forandra verda. Spørsmålet er då korleis verda *burde vera*. Det same gjeld andre *profesjonar* (jf. Schön, 1983). Simon (1969, s. 55) omtaler profesjonane kollektivt som *design*. Definisjonen hans vert ofte sitert:

> *Everyone designs who devises courses of action aimed at changing existing situations into preferred ones.*

Ein designar legg altso handlingslaup som skal skapa ein ny og føretrekt situasjon. Dette gjeld oss som programutviklarar, som legg handlingslaup for å utvikla eksisterande programvare til meir føretrekt programvare. Det gjeld oss ogso som undervisarar, som legg handlingslaup i form av læringsaktivitetar som skal setja studentane i ein ny situasjon med høgare kompetanse. Til sist gjeld det studentane våre, som sjølv i stor grad kan velja sitt handlingslaup gjennom studiet. Alt som vert forma gjennom slik handlingslaup kaller Simon for *det kunstige*.

Simon bruker nemninga *det kunstige* i ei heilt spesifikk tyding. Kunstige fenomen er som dei er, fordi dei vert forma av mål og hensikter, og tilpassa miljøet rundt (Simon, 1969, s. ix). Naturvitskapen har vunne høg status på grunn av den modne epistemologien som sikrar at vitskapleg kunnskap er objektiv og reproduserbar. Idealet er kontekstfrie resultat, dvs. kunnskap som er gyldig i ein kvar ny samanheng. Når det kunstige derimot stadig vert forma av konteksta, vert kontekstfrie resultat irrelevante. Simon (1969) meiner difor at me treng ein *vitskap om det kunstige*, som ein motsats til naturvitskapen. Naturen er styrt av *nødvendige* samanhengar (*necessities*). Det kunstige, som er tilverka av menneske, er derimot styrt av eventualitetar (*contingencies*).

Både i matematikken og i fysikken kan me tala om at ei løysing eller ein teori er sann eller rett. Løysinga er ei naudsynt fylgje av problemet. Eit dataprogram er i høgda rett *for brukarane* i den situasjonen der dei finn seg i dag. Med løysinga fylgjer ny innsikt som leier til nye behov. Dermed er løysinga som var rett i går, ikkje rett i morgon. Dette er ein vesentleg eventualitet som ein profesjonsepistemologi må ta omsyn til.

Matematikken bruker logisk-deduktiv metode for å koma fram til slutningar som *nødvendigvis* fylgjer frå aksioma. I empirisk forsking er det vanleg å testa moglege teoriar ved å dedusera *naudsynte* konsekvensar av teoriar, og testa om desse konsekvensane oppstår i røynda. Når me skal utvikla ny teknologi er situasjonen ein annan. Me står overfor kundar og brukarar med motstridande behov. Ofte veit dei ikkje kva dei treng eller kva som er mogleg å be om. Då lèt det seg ikkje gjera å dedusera logisk eit programsystem som løyser problemet. Utviklaren må setja seg inn i alle dei eventualitetane og einskildtinga som rår i ein unik situasjon, og vega dei opp mot kvarandre. Resultatet er ofte at utviklaren tyr til individuelt skjønn i staden for objektiv fornuft.

Er det so mogleg å tala om ein objektiv fornuft for einskildting? Dette problemet har vore studert sidan antikken. Aristoteles (1999, Bok 6) delte sinnet, eller fornufta, i ein vitande del (ἐπιστημονικὸν—*epistemonikon*) og ein overvegande del (λογιστικόν—*logistikon*). Den vitande delen kan finna kunnskap om det som *ikkje kunne vore annleis*, dvs. det naudsynte og det universelle. Den overvegande fornufta skjøner seg derimot på einskildting, og omfattar kunst (*techne*—τέχνη) som fortel oss kva som er bra eller rett å lage, og klokskap (*phronesis*—φρόνησῐς) som fortel oss kva som er rett å gjera. Alt dette handlar om fornuft, som let oss vita kva som er sant, og skil seg dermed frå meiningar der me kan ta feil.

Mange seinare tenkjarar i Vesten har rekna den universelle kunnskapen (*episteme*—ἐπιστήμη) som den einast sanne og dermed neglisjert eventualitetsbasert fornuft. Der har likevel vore nokre unntak, særleg i Ciceros humanisme, renessansen og amerikansk pragmatisme (Goldman, 2004).

Vitskapsidealane prega utviklinga av både metode og utdanninga for profesjonane i fleire generasjonar; ei utvikling som på mange måtar toppa seg på 1960-talet (sjå t.d. Schön, 1983). Deretter har stadig fleire forfattarar peikt på korleis profesjonsproblem skil seg frå vitskapsproblem. Horst Rittel talte om *wicked problems* (Rittel & Weber, 1973), Simon (1973) om *ill structured problems*, og Schön (1983) om unike fall. Desse ulike konseptualiseringane peiker på det same behovet for ein ein eventualitetsbasert fornuft. Særleg innanfor designtradisjonen er der ein rik litteratur som utdjupar dette, men det er verd å merkja seg Goldman (2004) som drøftar det i ei rein ingeniørkontekst.

Design er sjølvsagt eit tvitydig omgrep. Det viser til ein fagdisiplin med sin eigen tradisjon og kultur (t.d. industridesign). Det viser til ein aktivitet som gjerne er eitt steg i utviklinga av (t.d.) programvare før sjølve implementasjonen. I definisjonen åt Simon vert design praktisk talt synonymt med profesjonane, og det er denne tydinga me vil bruka her. Samstundes skal me tala om ei snevrare design*tradisjon*, som omfattar arkitektur og industridesign m.m., der ein i større grad allereie har studert korleis designproblem skil seg frå vitskaplege problem.

3 *Wicked Problems* og *Bounded Rationality*

Design og utvikling er eit møte mellom problem og fornuft, og der er visse drag, både ved problema og ved fornufta, som avgjer kva som er hensiktsmessig metode.

Då Herbert Simon fekk nobelprisen i økonomi i 1978, var det i stor grad pga. arbeidet om *avgrensa fornuft* (*bounded rationality*) frå 1950-talet (Carlson, 1978). Han forska på beslutningstaking i større organisasjonar og viste at dei *ikkje* er rasjonelle aktørar som tek optimale avgjersler, slik som klassisk økonomisk teori føreset. Beslutningsprosessane er prega både av ufullstendig informasjon og utilstrekkelege kapasitet til å vurdera eventualitatane. Det er difor ikkje mogleg å treffa optimale eller perfekte avgjerdsler. Simon konkluderte med at fornufta vår er avgrensa, og beslutningstakarane tilpasser seg det. I praksis fører dette òg til at problemløysing vert eit søk etter *tilfredsstillande*, men gjerne suboptimale løysingar. Me kan forsvara slike løysingar rasjonelt ved å visa at dei er konsistente

med tilgjengeleg informasjon (Simon, 1996, s. 169), utan at me utelukkar at ny informasjon eller grundigare vurderingar kan gje betre svar. Optimale løysingar er berre mogleg på enkle problem.

Den avgrensa fornufta har ei kontroversiell fylgje for design og utvikling. Det er ikkje fornuftig (rasjonelt) å ta utgangspunkt i endelege mål. Me har rett og slett ikkje nok kunnskap, i starten av eit komplekst prosjekt, til å binda oss til endelege mål. I dei fleste tilfelle gjev prosjektet ny innsikt som viser at dei opphavlege måla er feil eller ugunstige. Simon (1996, s. 162) føreslår at hensikta med designmål heller er å motivera aktivitet som igjen leier til nye mål.

Fornufta skal, tross grensene sine, handtera problem som er *uskikkelege* (*wicked*) og dermed stadig tøyer dei etablerte grensene for kva me kan løysa. Buchanan (1992) framhevar nettopp korleis designproblem er grenselause. Dei er ikkje berre problem som krev vidare utgreiing før me finn ein veldefinert form; dei lèt seg faktisk ikkje endeleg bestemma. Allereie Rittel og Weber (1973), som innførte omgrepet *wicked* problem, var inne på dette. Dei skriv m.a. at eit uskikkeleg problem alltid er eit symptom på eit anna uskikkeleg problem på høgare nivå. Ein av grunnane til at det er slik er at kvar delløysing gjev oss ny innsikt som gjerne leier til større forventingar. Som Simon (1969, s. 3) skriv, «as man's aims change, so do his artifacts—and vice versa, as well». Løysingane genererer nye problem. Dermed vil uskikkeleg problem heller ikkje ha nokon stoppregel (Rittel & Weber, 1973). Me vil alltid finna nye ynskjer og mål som gjer at eventuelle stoppreglar vert flytta.

Me skal nemna at Simons forfattarskap framstår som tvitydig. Gjennom det omfattande arbeidet sitt på kunstig intelligens står han fram me klokkartru på at all problemløysing essensielt er iterative, heuristiske søk som kan automatiserast. Når han skriv om *ill structured problems* (Simon, 1973), er det noko anna enn uskikkelege problem. Han ser ikkje noko skarpt skilje mellom veldefinerte og vandefinerte problem. Dei vanskelege problema krev berre noko meir fornuft enn det som er tilgjengeleg i augneblinken. Mange forfattarar, ikkje minst Schön (1983), tilskriv difor Simon eit lineært og mekanistisk syn på problemløysing, trass i at Simon meir enn nokon annan har framheva korleis menneskelege mål er i stadig endring og dermed bidreg til kompleksiteten (sjå òg Chua, 2009). Simon gjer det klart at fornufta må kunna identifisera, vurdera og velja mål, men korleis skriv han diverre ikkje om.

Innanfor designdisiplinen er det velkjend at målet ikkje er gjeve på førehand. Likevel har eit lineært syn vore vanleg (Buchanan, 1992), med ein problemdefinisjonsfase og deretter ei problemløysingsfase (sjå td. Norman, 2013). Uskikkelege problem krev derimot at problemet utviklar seg parallelt med løysinga. Schön (1983) er ein av dei fyrste som viste korleis profesjonsutøvarar stadig rammar om (*reframe*) problemet gjennom løysingsprosessen. Maher mfl. (1996) har sidan vidareutvikla og formalisert denne idéen som samevolusjon (*co-evolution*) av problem og løysing. Dorst (2019) (m.a.) har vist empirisk at denne samevolusjonen er vanleg i designprosessar.

Uskikkelege problem er tidslege. Problemet eksisterer i notid, og fornufta må ta stilling til det her og no. Løysinga høyrer derimot til ein framtid som

enno ikkje er. Når me sidan kjem til det punktet der me skal evaluera løysinga, høyrer problemet til ei fortid som ikkje lenger er. Me kan ha eit minne om problemet som var, men me kan ikkje sjå bort frå nye problem i notida når me skal vurdera løysinga. Simon (1996) liknar designprosessen med naturleg evolusjon, i eit kapittel som mangla i 1969-utgåva.

Evolusjonsidéen heng saman med designdefinisjonen. Designaren skal ikkje berre konstruera den føretrekte situasjonen, men heile handlingslaupet som bring oss frå den eksisterande til den føretrekte situasjonen. Handlingsrommet er heile tida avgrensa av situasjonen som ein står i. Mutasjonane i naturleg evolusjon tek aldri store sprang. Kvart designval og kvar ny situasjon i den kunstige verda er forma av dei moglegheitene ein hadde, og treng ikkje framstå som fornuftig i ein seinare situasjon. Difor seier Simon (1996, s. 47) at me berre kan forstå utviklinga og framtida til slike designa system gjennom å kjenna historia.

Evolusjonsmetaforen er sjølvsagt ikkje noko anna enn ein velkjend iterativ utviklingsmetode, men metaforen framhevar eit par viktige moment som ein lyt vera medvitne om. All menneskeleg problemløysing, famlande eller innsiktsfull, er, seier Simon (1996, s. 195), ikkje noko anna enn prøving og feiling og utval i ulike blandingar. Designaren vil vekselvis generera og evaluera moglege trekk (handlingar) (jf. Simon, 1972). Somme trekk leier, gjennom evalueringa, til det som Simon (1996) kaller *stabile formar*, eller levedyktige avkom i den biologiske metaforen. Stabile formar demonstrerer synlege framsteg i utviklinga og dannar byggesteinar til seinare generasjonar. Dette føreset at dei er persistente og dermed vert hugsa til seinare. Nye versjonar av køyrbar programvare er eit døme på slike stabile formar, og målretta bruk av versjonskontroll gjer dei persistente slik at me seinare kan fylgja historia når det trengst.

Avgrensa fornuft gjer at me aldri vil generera alle moglege trekk for so gjera utval gjennom evalueringa. Sjølve trekkgenereringa inneber eit utval (Simon, 1996, s. 195), der designaren byggjer på tidlegare røynsler. Dette opnar for at intuitive og raske tankeprosessar, det som Kahneman (2011) kaller *System 1*, kan stå for mykje av trekkgenereringa. Slike intuitive tankeprosessar er for Simon ikkje noko anna enn informasjonsbehandling i den enorme databanken som inneheld røynslene åt eksperten, men som Kahneman har dokumentert opnar dei raske prosessane for å ta feil. Dette er likevel ikkje noko problem, dersom det sakte og pålitelige tankesystemet (System 2) tek over i trekkevalueringa.

Trekkevaluering handlar ikkje fyrst og fremst om evaluering av fullstendige løysingsforslag. Langt oftare evaluerer me eit trekk for å få innsikt som informerer vidare søk (Simon, 1972). Kvart einskild trekk, kvar handling i handlingslaupet, må evaluerast for å vurdera om og korleis det kan passa inn i ei større løysing. Somme evalueringar fører til ein føretrekt situasjon gjennom betre innsikt, andre til eit betre produkt.

Simon skriv lite om sjølve evalueringsmetodikken. Her er der meir å finna hjå Schön (1983) som er eit av dei aller mest siterte verka om designmetode. Schön vert ofte sett som ein diametral motsats til Simon, der intuisjon skal vera viktigare enn fornuft, men det er i beste fall ei overforenkling. Schön formulerer ein metodikk som han kaller *reflection-in-action*, og her inngår ei grundig drøfting

av den eksperimentelle metoden som skal sikra objektivitet. Eksperimenta har den same rolla som dei har hjå Popper. Kvart eksperiment har ein sjanse til avsløra lyte i det føreslegne trekket. Trekk som overlever mange eksperiment er sannsynlegvis bra. Dette føreset likevel at designaren aktivt freistar å finna lyta, vel vitande om at han kan ta feil. Han må verkeleg setja trekket på prøve, og på varierte prøver, for å bevisa at han sjølv tek feil. Løysinga vert då objektiv i den meininga at det er mogleg å tilbakevisa ho (Schön, 1983, s. 166).

Schön skriv hovudsakleg om tankeeksperiment. Med utgangspunkt i skisser og historier vil designaren *førestilla seg* situasjonen som kan bli skapt. Gjennom førestillinga kan han sjå seg sjølv som brukar — som menneske — og vurdera kva løysinga tyder for oss som menneske. Dermed er eksperimentet ikkje avgrensa til førehandsdefinerte kriterium og mål. Gjennom førestillinga kan ein sjå kva som er viktig *for folk*, og ikkje berre det som prosessen på førehand har definert som viktig. Dette gjev eit brukarperspektiv som manglar i funksjonstesting, og det gjer det mogleg å identifisera nye mål.

Prosessen er skildra som ei samtale med situasjonen. Designaren føreslår trekk — forandringar — og situasjonen svarer tilbake, med ynskte og uynskte konsekvensar av trekket. For at dette skal fungera må skissa (eller prototypen) vera tilstrekkeleg detaljert og presis (Schön, 1987, s. 140). Dvs. eksperimentet må ta utgangspunkt i ein *stabil form*. Dersom formen ikkje er stabil, vil han gje etter og tilpassa seg historia som tankeeksperimentet fortel.

Overrasking er eit viktig fenomen i eksperimenta åt Schön, liksom det òg er i naturleg evolusjon. Det er slike overraskingar som fører til utviklingssprang i naturleg evolusjon, og i design bryt dei opp etablert rutine (Dorst & Cross, 2001). Overrasking er mogleg fordi designaren kan sjå meir i skissa enn det som var tilsikta (Menezes & Lawson, 2006; Schön & Wiggins, 1992). Når me studerer ei skisse til skjermbilete og lever oss inn i korleis ho vil verta sedd av brukaren, kan me sjå knappar som fangar merksemda og minner oss om arbeidsoppgåver som er gløymt. Både positive og negative overraskingar inspirerer nye trekk (Schön, 1983).

Ein av grunnane til at desse overraskingane er so viktige er kompleksiteten i problemet. Når ein genererer trekk er det ikkje mogleg å ta omsyn til alle sider ved problemet eller forutsjå alle moglege konsekvensar. Det er det forsovidt heller ikkje når ein testar ut frå veldefinerte kriterium. I Schöns opne tilnærming skal designaren leva seg inn i ei tenkt verd og førestilla seg sjølv som brukar. I denne førestillinga står heile designet på prøve og målestokken er mennesket. Både nye mål og nye løysingar kan koma til syne[1].

4 Dei smidige metodane

Som annan design, handlar programutvikling ofte om uskikkeleg problem (Nerur & Balijepally, 2007). Tradisjonelle utviklingsmetodar (fossefallsmetodane) freista lenge å handtera aukande kompleksitet med meir komplekse prosessar, utan å lukkast. Smidige metodar er oppstått som ein motreaksjon.

[1] Dette har eg freista å utdjupa i ein annan artikkel (Schaathun, 2021).

Kontrasten mellom smidige metodar og fossefallstilnærminga er godt illustrert av Bedoll (2003), som skildrar to forsøk på å løysa det same problemet i flyindustrien. Kabelsystema ombord på fly omfattar titusenvis av kablar, og kvart einskild fly har typisk eit unikt kabelsystem. Boeing hadde på det tidspunktet om lag 1000 ingeniørar som utelukkande arbeidde med design av elektrisk utstyr og kabelsystem. Ynsket var eit datasystem som kunne hjelpa med å systematisera og modularisera kabelsystemet. Det fyrste prosjektet var eit stort fossefallsprosjekt, og Bedoll fortel korleis kravspesifikasjonen endra seg raskare enn dei kunne utvikla. Fyrst etter tre år og åtte *releases* fekk dei lov å testa systemet med brukarar. Konklusjonen var da at *korkje* systemet *heller* prosessen som systemet skulle støtta, var brukbar.

Då dei forsøkte igjen hadde dei berre ei lita prosjektgruppe, brukte enkle verkty (Visual Basic) og tok ei smidig tilnærming, i tett samarbeid med brukarane. Dei utvikla forretningsprosessen og datasystemet parallelt. Nye versjonar av programvaren vart sette i produksjon kvar veke.

Her kjenner me att fleire av fenomena som me har drøfta generelt. Programutviklarane står overfor eit uskikkeleg problem. So lenge forretningsprosessen er ukjend, so er det uråd å finna eit veldefinert problem for dei som skal utvikla datasystemet. Evolusjonsmetaforen er treffande. Det store spranget til rett løysing er umogleg, men det er mogleg å ta eitt lite steg åt gongen. Kvar produksjonsutgåve av programvaren dannar ein stabil form, som i dette tilfellet vert testa i faktisk produksjon.

Dette dømet er ekstremt, noko som Bedoll òg understreker. Behovet for datasystemet var prekært, og brukarane hadde difor sterke insentiv til å bidra aktivt. Sjølv enkle delløysingar hadde stor verdi og nye utgåver var heile tida velkomne. Mange prosjekt må klara seg me meir sporadisk brukarkontakt, og dei må då finna andre formar for trekkevaluering. Kva kan me då seia om styrkane ved smidige metodar generelt?

Det Smidige Manifestet[2] er det næraste me kjem ein generell definisjon av smidige metodar. Kjernen er fire verdiar:

1. *Personer og samspill fremfor prosesser og verktøy*
2. *Programvare som virker fremfor omfattende dokumentasjon*
3. *Samarbeid med kunden fremfor kontraktforhandlingar*
4. *Å reagere på endringer fremfor å følge en plan*

Me skal òg skjela til eit forslag til eit «hinsides smidig manifest», frå Kent Beck i 2011, vist i tabell 1. Han var òg ein av dei opphavlege forfattarane.

Dei to siste verdiane fylgjer direkte frå avgrensa fornuft. Planar, og i endå større grad kontraktar, vert fastlagde på starten av prosjektet med dei avgrensa ressursane som då er tilgjengelege. Når prosjektet stendig gjev ny innsikt som seier at opphavlege planar og kontraktar ikkje lenger er fornuftige eller relevante, har me ikkje anna val enn å reagera på endringar. Samarbeidet med kunden er naudsynt både for å dra nytte av innsikta som kunden har og for å einast om endringane og unngå avvikande forventingar og påfylgjande skuffelse.

[2] https://agilemanifesto.org/iso/no/manifesto.html

- Team vision and discipline over individuals and interactions (over processes and tools)
- Validated learning over working software (over comprehensive documentation)
- Customer discovery over customer collaboration (over contract negotiation)
- Initiating change over responding to change (over following a plan)

Tabell 1. Det hinsides smidige manifestet frå Kent Beck 2011, sitert frå Hohl mfl. (2018).

I det hinsides smidige manifestet er det ikkje nok å reagera på endringane. Me *veit* at opphavlege planar var grunna i sviktande kunnskapsgrunnlag, og det vil vera både uansvarleg og irrasjonelt ikkje proaktivt å søkja endring. Kunden er ikkje alltid medviten om denne situasjonen, og utviklaren, i kraft av kompetansen sin, har eit ansvar for å driva prosessen. Beck peiker òg på at ein ikkje alltid har ein kunde å samarbeida med, og ein må då tenkja på å finna ut kven kunden er.

Det andre punktet, om programvare som verkar, har ei forretningsmessig side i form av reell og umiddelbar verdi for kunden. Der er òg ei epistemologisk side. Programvaren dannar ein *stabil form* som opnar for eksperiment som kan gje ny innsikt, både om kvaliteten på valde løysingar og om rom for vidare utvikling. Slike eksperiment er mogleg både med og utan kunden. Beck presiserer dette ved å framheva stadfest læring. Med tanke på vidare utvikling er innsikta som me får av å køyra programvaren, viktigare enn sjølve programvaren. Me skal heller ikkje underkjenna andre formar for prototypar (*wireframes*, modellar, osv.) som kan danna grunnlag for eksperiment.

Det fyrste punktet, som prioriterer personer og samspel over prosessar og verkty, flytter fokus frå *korleis* til *kvifor*. Hensikta med prosessar og verkty er samspel, men der er ein risiko for å gløyma hensikta og fokusera på rituala. Dette gjeld særleg dersom ein freistar å løysa problem med meir prosess og fleire verkty. Avgrensa fornuft, og spesielt avgrensa arbeidsminne, gjer at der er ei grense for kor kompliserte prosessar me maktar å handtera. Ved å flytta fokus til personar og samspel får kvar einskild prosjektdeltakar ansvar for det som er vesentleg. Beck går eit hakk vidare. Det er ikkje samspel i seg sjølv som er målet, men felles mål og retning for gruppa. Samspelet er eit middel for å oppretthalda felles mål og målforståing.

For å venda tilbake til forskingsspørsmåla. Me har sett at avgrensa fornuft og uskikkeleg problem gjev eit teoretisk grunnlag for å seia at fossefallsmetodikken berre er relevant for relativt enkle problem. Metoden er ikkje i og for seg dårleg, men han føreset veldefinerte og stabile problem. Smidige metodar er smidige i den meininga at utviklaren stadig tilpassar seg skiftande mål- og problemforståing. Dette fritek oss derimot ikkje for kravet om objektive løysingar, so korleis kan me vera trygge på at smidige metodar gjev gode og riktige svar?

Ved at smidige metodar fokuserer på hyppige utgåver av køyrbar programvare, vert evolusjonsmetaforen frå Simon langt meir konkret enn for dei fleste andre formar for design. Dei stabile formane får oftare eit liv i produksjon og ikkje berre eit liv på teiknebrettet. Evaluering handlar ikkje om sluttprodukt,

men føregår for kvar utgåve og for kvar einaste *pull request*. Det er i denne evalueringsprosessen at me kan sikra oss objektivitet og sikker kunnskap. Trekkgenereringa kan gjerne vera intuitiv, men som Kahneman (2011, s. 49) seier, handlar rasjonell tenking om grundig og tolmodig vurdera dei idéane gjennom ein saktare tankeprosess. Som me lærer av Schön består evalueringa av ein serie med eksperiment der designaren aktivt søkjer å finna feil og lyte i si eiga løysing.

Den enklaste og mest openbare formen for eksperiment er brukartesting, og smidige metodar tek til orde for å gjera det ofte. Utviklaren gjer derimot designval oftare enn han kan møta brukaren, og ein er difor avhengig av fleire formar for testing. Mange smidige paradigme tek til orde for brukarhistorier, som eit format for brukarkrav. Brukarhistoriene dannar òg eit utgangspunkt for tankeeksperiment i rammeverket frå Schön. Ved å gjenfortelja brukarhistoria med utgangspunkt i eit konkret design eller ein prototype, kan utviklaren førestilla seg sjølv som brukar og sjå programvaren som om han var brukaren i historia. Førestillingsevna vår gjer det dermed mogleg å testa frå eit brukarperspektiv ogso mellom møta med brukaren. Kvar brukarhistorie kan altso brukast til å tilbakevisa ei løysing, og løysingane er objektive fordi dei kan tilbakevisast gjennom å fortelja brukarhistorier.

Fordi problema (og løysingane) er komplekse, kan endringar i programmet ha konsekvensar for andre brukarhistorier enn dei som ein meinte å løysa. Difor må ein òg i programutvikling tenkja som Schön, på heilskapen i systemet. Ein må fortelja gamle brukarhistorier på nytt for nye versjonar, og vera budd på uttilsikta effektar av seinare endringar.

Som me ser legg smidige metodar godt til rette for ei objektiv og epistemologisk tilnærming, men epistemologisk vissheit er ikkje noko ein får gratis. Objektivitet handlar om å *prøva* å finna lyte, heller enn å visa at programmet løyser eit problem. Utviklaren må aktivt designa effektive eksperiment, og initiera endringar gjennom aktivt søkja betre mål- og problemforståing. Samarbeidet med brukaren er uunnverleg, men ein skal presisera *samarbeid*. Heller ikkje brukaren har kompetanse til aleine å definera mål og problem.

5 Konsekvensar for undervisinga

Avslutningsvis skal me sjå på korleis me kan bruka denne innsikta i undervising og læring. Spørsmålet om korleis studentane lærer yrkespraksis skal me la liggja til ein annan artikkel. Hovudfokuset er korleis me objektivt kan vurdera om studentane sitt programmeringsarbeid er godt eller «riktig». Både sensor og student må kunne gjera denne vurderinga, og *objektivitet* inneber mellom anna at dei stort sett skal koma til same vurdering.

Det kan fyrst løna seg å spørja om smidige metodar er rett tilnærming for studenten. Kan henda skal me alltid be studenten fyrst stilla seg tre spørsmål:

1. Kva veit du om problemet?
2. Kva veit du for lite om?
3. Har du nok kunnskap til å definera eit velforma problem og leggja ein detaljert plan?

Teorien om uskikkelege problem fortel oss at det er heilt greitt om svaret på siste spørsmål er nei. I so fall må me velja ein smidig metode, men studenten skal òg vita at grunnen er manglande problemforståing. Då er det ikkje nok å løysa problemet. Ein må òg finna problemet. Dersom problemet er veldefinert frå starten, kan ein klara seg med ein formell sluttevaluering som viser at produktet løyser problemet. Denne situasjonen er studert før, og me skal ikkje gå nærare inn på han her.

Når utgangspunktet er eit uskikkeleg problem, fortel Simon oss at det er uråd å forstå systemet (løysinga) utan å vita ein del om historia. Dette tyder ikkje at me treng å høyra historia om prosjektarbeidet, om korleis studenten har strevd med verktysinnstallasjon, kontakt med oppdragsgjevar eller anna. Det er historiene til løysinga og til problemet me treng å høyra.

Historia består av ein serie med eksperiment og designval, der kvart val er bestemt av innsikta frå tidlegare steg. I prosjekt som fokuserer på brukarinteraksjon kan ein eksperimentera gjennom å fortelja brukarhistorier, der ein skildrar både positive og negative opplevingar i møtet med programvaren. Det er desse historiene som kan dokumentera at løysinga er fornuftig. (Prosjekt med hovudvekt på utrekningar kan krevja andre formar for eksperiment.)

Lat oss, som eit tankeeksperiment, tenkja oss at me krev at studentane skal dokumentera designval som er gjort undervegs, minst eitt per veke. Som ein del av designvala må dei dokumentera eksperiment og dei avvegingane som dei har gjort. Sensor får då informasjon om tre viktige ting. For det fyrste viser studenten korleis han har utført det *kognitive* handverket undervegs. Sensor kan då ta stilling til kor grundig og systematisk dette er gjort. Designval kan stadig vera subjektive, men sensor kan ta stilling til om dei vert gjort i tråd med god handverksmessig praksis. Vidare fortel desse eksperiment kvifor produktet er blitt som det er blitt. Somme designval kan ha vore fornuftige då dei vart tekne, utan at dei viser seg fornuftige i ei sluttevaluering. Til slutt vil desse eksperimenta, i alle fall til ein viss grad, dokumentera vanskegraden i prosjektet. Ein kan godt dokumentera eksperiment som har gjeve ny innsikt utan at dei har forbetra produktet. Når ein ser eksperimentet i samanheng med fortida på det tidspunktet det vart gjennomført, vil ein ofte sjå at det var riktig og naudsynt å gjennomføra det, sjølv om det kan ha vore eit blindspor. Slike eksperiment kan dokumentera at studenten meistrar faget like godt som faktiske køyrbare forbetringar.

Dette dokumentasjonskravet kan framstå som overdrive samanlikna med industripraksis. Den store skilnaden er likevel at dokumentasjonen vert lagde fram samla i ein sluttsensur, i staden for å verta presentert regelmessig i møte med brukarar og prosjekteigarar. Sluttsensuren er då ogso ein uunngåeleg skilnad mellom studentliv og yrkesliv.

Kritisk tenking vert ofte nemnd som sentralt i universitetsutdanning. Det inneber at ein er kritisk til eigne løysingar, og ikkje umiddelbart aksepterer eit innfall som ser ålreit ut. Denne kritiske tenkinga kan ikkje utsetjast til sluttevalueringa, fordi kritiske val vert tekne heile vegen gjennom prosessen. Eit av dei viktigaste verktya som me har for kritisk tenking er eksperiment som set løysinga

på prøve, og slike eksperiment må vera detaljerte nok til å gje presis informasjon. Brukarhistoriene er eit mogleg verkty til slike eksperiment, ogso når brukaren ikkje er tilgjengeleg.

Hyppige eksperiment er òg essensielt når me har mangefull innsikt i problemet. Boaler (2015) har peikt på at dei mest suksessrike menneska i verda er dei som gjer flest feil. Det er gjerne fordi den einaste måten å forstå problemet på, er å prøva ei løysinga og vera viljug til å ta feil.

6 Eksisterande litteratur

Der er skrive mykje om smidige metodar, sjølv om det kunnskapsteoretiske perspektivet er sjelden. Ein relativt ny litteraturstudie finst i hovudoppgåva åt Braams (2020). Nerur og Balijepally (2007) er ein relativt tidleg studie av det teoretiske grunnlaget for smidige metodar, og fylgjer parallellane til dei tidlegare utviklingane i design og arkitektur, og i strategisk leiing. Der finst og ein litteraturstudie frå Dingsøyr mfl. (2012).

Tang mfl. (2010) har studert dei kognitive prosessane ved programvaredesign ved å samanlikna observasjonar av to designlag. Dei observerer både løysingsdrivne og problemdrivne tilnærmingar, og dei stadfestar samevolusjon av problem og løysing. Browaeys og Fisser (2012) gjev ein epistemologisk studie av smidige metodar med utgangspunkt i kompleksitetstenkinga åt Morin.

Babb mfl. (2017) drøftar korleis smidige metodar har leidd til nye verkty og prosessar for hurtig integrasjon og levering av nye programversjonar. Dei ser ein mogleg ende på det smidige idealet som me kjenner, der hurtige leveransar vert viktigare enn problemforståing og læring.

Hohl mfl. (2018) tek eit tilbakeblikk på smidige metodar, i samarbeid med opphavsmennene bak manifestet. Han peiker m.a. på at manifestet i seg sjølv er ei løysing på eit problem til ei viss tid, og kan krevja vidare evolusjon. Der er òg ein skilnad mellom å fylgja smidige metodar og faktisk å *vera smidig* (≪doing agile≫ versus ≪being agile≫).

Design-perspektivet er godt kjend som *Design Science* innanfor *Information Systems*. Hevner mfl. (2004) vert gjerne sitert som milepålen i denne tradisjonen. Det er derimot verd å merkja seg at *Design Science* etter kvart har utvikla seg uavhengig av designtradisjonen.

7 Konklusjon

Det er vanskeleg å diskutera kva som er riktige løysingar i programutvikling og praktiske fag generelt, i alle fall når problema ikkje på førehand er presist og utvitydig formulerte. Dette er ekstra vanskeleg for dei som ikkje sjølv har vore med i prosjektet, slik som sensorar ved utdanningane. Det kan òg vera utfordrande for studentar som er i ferd med å læra faget. Det er heller ikkje berre i utdanningane at me legg vekt på epistemologi. Kulturen vår har stort sett, med relativt få unntak, lagt stor vekt på vitskapleg kunnskap, dvs. objektiv, reproduserbar og kontekstfri kunnskap.

Me hevdar her at det er mogleg objektivt å vurdera løysingar innanfor programutvikling, dersom me legg vekt på systematiske eksperiment, (t.d.) vha. brukarhistorier, for kvart einaste steg i utviklinga. Ei slik tilnærming byggjer på Schöns *reflection-in-action* og er kompatibel med smidige metodar, om enn kanskje ikkje ein sjølvsagt del av smidige metodar. Ein slik eksperimentell metode er likevel ikkje fyrst og fremst ein prosedyre å fylgja. Objektivitet krev, som me allereie veit frå Poppers vitskapsteori, at utviklaren aktivt søkjer å motbevisa eigne løysingar.

Referansar

Aristoteles. (1999). *Etikk* [Omsett av Anfinn Stigen]. Gyldendal Akademisk.

Babb, J. S., Nørbjerg, J., Yates, D. J. & Waguespack, L. J. (2017). The Empire Strikes Back: The end of Agile as we know it? *Communications of the Association for Information Systems. Selected Papers of the IRIS*, (8), 43–59.

Bedoll, R. (2003). A Tail of Two Projects: How 'Agile' Methods Succeeded After 'Traditional' Methods Had Failed in a Critical System-Development Project. I F. Maurer & D. Wells (Red.), *Extreme Programming and Agile Methods - XP/Agile Universe 2003* (s. 25–34). Springer Berlin Heidelberg.

Boaler, J. (2015). *Mathematical mindsets: unleashing students' potential through creative math, inspiring messages, and innovative teaching*. Jossey-Bass.

Browaeys, M.-J. & Fisser, S. (2012). Lean and agile: an epistemological reflection. *The Learning Organization*.

Braams, S. M. (2020). *The software development landscape: A rationalization of agile software development as a strategy in the face of organizational complexity* (Masteroppgåve). University of Twente.

Buchanan, R. (1992). Wicked problems in design thinking. *Design issues*, *8*(2), 5–21.

Carlson, S. (1978). The Sveriges Riksbank prize in economic sciences in memory of Alfred Nobel 1978 presentation speech. http://nobelprize.org/nobel_prizes/economics/laureates/1978/presentation-speech.html

Chua, J. S. M. (2009). Donald Schön, Herbert Simon and The Sciences of the Artificial. *Design Studies*, *30*(1), 60–68. https://doi.org/10.1016/j.destud.2008.09.001

Dingsøyr, T., Nerur, S., Balijepally, V. & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development [Special Issue: Agile Development]. *Journal of Systems and Software*, *85*(6), 1213–1221. https://doi.org/https://doi.org/10.1016/j.jss.2012.02.033

Dorst, K. (2019). Co-evolution and emergence in design. *Design Studies*, *65*, 60–77. https://doi.org/10.1016/j.destud.2019.10.005

Dorst, K. & Cross, N. (2001). Creativity in the design process: co-evolution of problem–solution. *Design Studies*, *22*(5), 425–437. https://doi.org/10.1016/S0142-694X(01)00009-6

Goldman, S. L. (2004). Why we need a philosophy of engineering: a work in progress. *Interdisciplinary Science Reviews*, *29*(2), 163–176. https://doi.org/10.1179/030801804225012572

Hevner, A. R., March, S. T., Park, J. & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 75–105.

Hohl, P., Klünder, J., van Bennekum, A., Lockard, R., Gifford, J., Münch, J., Stupperich, M. & Schneider, K. (2018). Back to the future: origins and directions of the "Agile Manifesto" – views of the originators. *Journal of Software Engineering Research and Development*, *6*(1), 15. https://doi.org/10.1186/s40411-018-0059-z

Kahneman, D. (2011). *Thinking, fast and slow*. Farrar, Straus; Giroux.

Maher, M. L., Poon, J. & Boulanger, S. (1996). Formalising design exploration as co-evolution. *Advances in formal design methods for CAD* (s. 3–30). Springer.

Menezes, A. & Lawson, B. (2006). How designers perceive sketches. *Design Studies*, *27*(5), 571–585. https://doi.org/10.1016/j.destud.2006.02.001

Nerur, S. & Balijepally, V. (2007). Theoretical Reflections on Agile Development Methodologies. *Commun. ACM*, *50*(3), 79–83. https://doi.org/10.1145/1226736.1226739

Nicol, D. J. & Macfarlane-Dick, D. (2006). Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in higher education*, *31*(2), 199–218.

Norman, D. (2013). *The design of everyday things: Revised and expanded edition*. Constellation.

Rittel, H. & Weber, M. (1973). Dilemmas in a general theory of planning. *Policy Sciences*, *4*, 155–169.

Schön, D. A. (1983). *The Reflective Practioner*. Ashgate Arena.

Schön, D. A. (1987). *Educating the reflective practitioner*. Jossey-Bass San Francisco.

Schön, D. A. & Wiggins, G. (1992). Kinds of seeing and their functions in designing. *Design Studies*, *13*(2), 135–156. https://doi.org/10.1016/0142-694X(92)90268-F

Schaathun, H. G. (2021). Where Schön and Simon agree: The Rationality of Design [Preprint submitted for peer review.].

Simon, H. A. (1969). *The sciences of the artificial* (1st). MIT press.

Simon, H. A. (1972). Theories of bounded rationality. *Decision and organization*, *1*(1), 161–176.

Simon, H. A. (1973). The structure of ill structured problems. *Artificial Intelligence*, *4*(3), 181–201. https://doi.org/10.1016/0004-3702(73)90011-8

Simon, H. A. (1996). *The sciences of the artificial* (3rd). MIT press.

Tang, A., Aleti, A., Burge, J. & van Vliet, H. (2010). What makes software design effective? [Special Issue Studying Professional Software Design]. *Design Studies*, *31*(6), 614–640. https://doi.org/10.1016/j.destud.2010.09.004