Kortfattet oppsummering av resultater fra prosjektet:

NYTTESTYRING I IT-PROSJEKTER

Prosjektleder: Magne Jørgensen
Simula Metropolitan

Nedenfor følger en kortfattet oppsummering av det som oppfattes som de viktigste resultatene fra prosjektet Nyttestyring i IT-prosjekter, støttet av midler fra Prosjekt Norge. Utfyllende dokumentasjon av resultater finnes i vedleggene.

Tema 1: Hva kjennetegner IT-prosjekter som leverer god nytte?

Prosjektet bygger videre på tidligere undersøkelser av hovedsakelig norske IT-prosjekter. Gjennomgående i alle undersøkelsene er følgende mønster (kjennetegn) for IT-prosjekter som leverer god nytte:

- Fleksible kontrakter (per time e.l. baserte avtaler, ikke fastpris)
- Rammeavtaler med flere leverandører, og prosjektet kjøres som «internprosjekt» og ikke som «leverandørprosjekt». God kunde-evne til å velge de mest kompetente utviklere og andre fra, helst flere, leverandører. (dårligste modell var valg av en stor leverandør som kjører prosjektet).
- God bruk av smidige metoder, med vekt på fleksibelt innhold og hyppige leveranser. I tillegg har vi funnet at:
    - Mens mer kravendringer *ikke* reduserer vellykketheten til smidige prosjekter (smidigs motto om «embrace change» støttes alså av empiri, så gjelder det motsatte for ikke-smidige prosjekter (som dermed har rett i at «scope creep» er uheldig – for dem).
    - Smidige metoder passer minst like godt for store som for små prosjekter. Dette har det vært en del diskusjon om, med mange som har ment at smidig kun passer for de mindre prosjektene. Vi finner at det er grunn til å tro at det er særlig for de store prosjektene at smidig utvikling er viktig for å kunne levere god nytte.
- Underveis nyttestyring og plan for hvordan nytte skal hentes ut. Et gjennomgående funn i undersøkelsene er at det som virkelig skiller de som lykkes har gode prosesser for underveis nyttestyring og planer for uthenting av nytte. Slike prosesser og planer ser ikke ut til å måtte være avanserte, men kan for eksempel godt bestå i at prosjektet har regelmessig gjennomgang av prioriteter av leveranser (både hva og når) der oppdaterte kost-nytte vurderinger brukes, samt planer som er klare på hvem som er ansvarlig, hvordan og når nytte skal realiseres.
- Bruk av autonome team. Her må bemerkes at det er store variasjoner og at vi ikke har solide resultater på hvor stor grad av autonomi som er bra og hvordan effekten avhenger av andre forhold. Forskning på dette pågår og resultater vil trolig foreligge i starten av 2019.

Tema 2: Hvordan gjennomføre en god analyse av nytte, inkludert usikkerhetsanalyse?

Felles enhet for nytte som muliggjør god nyttestyring

Mens kostnader stort sett har en felles enhet (kroner og øre) så er det ikke alltid like enkelt å sammenfatte total nytte. Typisk har IT-prosjekter en liste med nytte-effekter der noen er målt i kroner og øre, mens andre nytteeffekter er målt på annen måte, eller ikke målt i det hele tatt. Dette vil i mange tilfelle vanskeliggjøre en god underveis nyttestyring – og også en god sammenligning mellom alternativ.

Vi har foreslått en metode for kvantifisering av nytte, med felles enhet (nyttepoeng) til bruk for kvantitativ underveis nyttestyring. Metoden består noe forenklet i:

- Angivelse av liste av målsetninger for prosjektet (f eks M1, M2 og M3)
- Angivelse av viktighet for disse i prosent av total nytte levert (f eks M1 utgjør ca. 50% av nytten prosjektet skal levere og M2 og M3 25% hver.)
    - Dersom ett av målene er kvantifisert (f eks at M1 skal gi 20 mill. i innsparte kostnader) så kan de andre beregnes ut fra dette (f eks at siden M1 gir 20 mill innsparing, så er verdien av M2 og M3 5 mill hver).
    - Dersom ingen av målene er kvantifisert så brukes «nyttepoeng», som angir relativ nytte (f eks at M1 gis 20 nyttepoeng, og man derav avleder at M2 og M3 går 5 nyttepoeng hver).
- Liste av leveranser for prosjektet (f eks L1, L2 og L3)
- For hver av leveransene angis hvor mye disse bidrar til måloppnåelse (f eks hvor mye L1 bidrar til oppnåelse av nytten ved M1, M2 og M3).
- Bidraget fra hver leveranse til nytteverdi (måloppnåelse) brukes til vurdering av prioritet til leveranse og dermed til underveis nyttestyring.
- Kontinuerlig tilbakemelding fra underveis leveranser og hva disse har oppnådd av nytte, og om nødvendig justering av nytteestimater per leveranse og/eller mål.
- Evaluering av faktisk oppnådd nytte i etterkant.

Forskningsprosjektet har evaluert denne metoden for kvantifisering og styring av nytte, både hos SPK og Oslo kommune, og funnet at den lar seg gjennomføre. I hvilken grad den fører til bedre nyttestyring er fortsatt ikke avklart. En bok om metoden (på Springer-forlaget, vil være gratis nedlastbar) er planlagt for 2019.

Bedre metoder for usikkerhetsanalyse av nytte (og kostnader)

Vi har avdekket av estimering og usikkerhetsanalyse på nytte er et svært umodent område for de aller fleste norske IT-organisasjoner. Stort sett gjøres ikke usikkerhetsanalyser på nytte i det hele tatt, eller så gjøres det uten å kvantifisere usikkerheten. Dette finner vi at typisk gir en urettmessig fordel for de usikre alternativene (konseptene), som for eksempel nyutvikling, målt mot der mer forutsigbare alternativene, som videreutvikling av eksisterende løsninger. Manglende usikkerhetsanalyse av nytte gjør det også slik at faktisk ulønnsomme tiltak kan se ut til å være lønnsomme. Vi har foreslått – og delvis evaluert – en metode for usikkerhetsanalyse for nytte (og kostnader) som baserer seg på at prosjektet:

- Finner fram til (eller bruker ekspertvurderinger) fordeling for tidligere avvik mellom estimert nytte (og kostnad) for lignende prosjekter (f eks at 50% av tidligere prosjekter av lignende type har levert 80% eller mindre enn estimert)
- Bruker denne fordelingen til å estimere usikkerheten til nåværende estimat (f eks at dersom nytten er estimert til 20 mill, og 50% av tidligere estimater av nytte har levert 80% eller mindre nytte enn estimert, så er p50 (50% sikkert) for dette prosjektet at prosjektet leverer minst 16 mill. i nytte)

Ovennevnte metode har blitt evaluert for reelle prosjekter for evaluering av kostnader, og funnet å gi økt realisme. Forskningsprosjektets analyser tyder på at dette også vil være tilfelle for usikkerhetsanalyser av nytte.

Vedlegg 1: Vitenskapelige artikler med resultater fra prosjektet

- Jørgensen, M. (2018, May). Do Agile Methods Work for Large Software Projects? In International Conference on Agile Software Development (pp. 179-190). Springer, Cham.
- Jørgensen, M. (2018, December). Looking back on previous estimation error as a method to improve the uncertainty assessment of benefits and costs of software development projects. In 9th International Workshop on Empirical Software Engineering Practice (IWESEP 2019). (Innhold også presentert på Prosjekt Norges årlige konferanse i 2017)
- Jørgensen, M. (2018, December). Scope creep or embrace change? A survey of the connection between requirement changes, use of agile, and software project success. 12th International conference on Project Management (ProMAC) (pp. 673-681).
- Jørgensen, M. (2018). Relations between Project Size, Agile Practices and Successful Software Development (Akseptert for utgivelse i IEEE Software).

Vedlegg 2: Utvalg av industri- og forskningspresentasjoner der resultater fra prosjektet er presentert

- **Keynote XP-conference, 2018.** When is agile better? How the use of agile and autonomous teams affect success differently in different contexts (and other results)
- **Presentation PMI/Prosjekt Norge, 2018:** Agile software development and benefits management: A perfect match.
- **Keynote ICSSE, 2018**. What makes software projects successful?
- **Invited talk, Delft symposium on data analytics, 2018.** The world is skewed. Ignorance, use, misuse, misunderstandings, and how to improve cost and benefits uncertainty analyses in software development projects.
- **Keynote Software (DnD), 2018**. Milliardinvesteringer i digitalisering. Hva gir det oss?
- **Presentasjoner på HIT-nettverkets seminarer**:
    - Oktober 2018: Storskala smidig IT-utvikling: Erfaringer med Spotify-modellen, SAFe og LeSS, Casper Lassenius
    - Mars 2018: Usikkerhetsvurderinger for nyttepoeng og kostpoeng, Jo Hannay

VEDLEGG 1

Scope Creep or Embrace Change? A Survey of the Connections Between Requirement Changes, Use of Agile, and Software Project Success

Abstract

Traditionally, a high degree of requirement change has been considered harmful for the success of software projects. Software professionals who use agile software development methods tend to view this topic differently. They tend to view requirement changes more as opportunities, which should be welcomed. Possibly, both views are correct but valid in different software development contexts. This paper aims at increasing the understanding of the connections between the degree of requirement change, choice of development method, and project success. Seventy software professionals were asked to provide information about their last software project. A higher degree of requirement changes, here defined as more than 30% of the requirements added, deleted, or changed during the project's execution, was connected with a higher proportion of successful projects in an agile development context, but only when this included frequent deliveries to production. Our results consequently support that the agile claim of "embrace change" has merit, but only in agile contexts.

1. Introduction

When software professionals are asked what they consider the main risk factors of software projects, they tend to include factors related to the requirement specifications. The survey reported in [1] is a good illustration. In that survey, the respondents ranked "misunderstanding the requirements" the second most important risk factor, "lack of frozen requirements" the sixth most important risk factor, and "changing scope/objectives" the seventh most important risk factor. Ranking incomplete and changing requirement specifications as important risk factors is in accordance with the traditional view of software development and requirement engineering. This view typically considers a requirement specification as consisting of "a set of system requirements which, as far as possible, is complete, consistent, relevant and reflects what the customer actually wants" [2].

Some software professionals seem to have different views on changed requirements. Those who use agile development methods recommend, among others, valuing "responding to change over following a plan,"[1] and to promote the principle of "welcome changing requirements, even late in development."[2] They also seem to think of requirement changes during the project's execution as opportunities to increase client values rather than as threats to the success of the project [3]. This corresponds with the observation that agile methods to some extent are designed for flexibility in scope and frequent requirement changes, e.g., as implemented in the common agile practice of flexible scope and frequent deliveries to client with opportunities for feedback and learning during the project execution.

The study reported in this paper tries to shed some light on the connection between requirement changes, development methods, and project outcomes. This include the goal of examining whether both viewpoints could be right, that is, that many requirement changes are connected with better outcomes for agile software projects, but worse outcomes for non-agile software projects. The main research questions are:

RQ1: How is the connection between amount of requirement changes and project outcome dependent on the development method?

RQ2: Among agile software projects, is there a difference in the connection between amount of requirement change and project outcome for project with and without frequent delivery to clients?

The second research question is motivated by our previous research, see [4], where frequent delivery to client were found to be one of the practices with strongest connection to project success.

The remainder of this paper is organized as follows: Section 2 describes selected related work on the effect of requirement changes, Section 3 describes the design and the results of the survey, Section 4 discusses the results and concludes.

2. Related work

A study by Serrador and Pinto [5], which examined 1002 software projects, suggests that the

[1] www.agilealliance.org/agile101/the-agile-manifesto/. Retrieved May 22, 2018.

[2] www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/. Retrieved May 22, 2018.

most successful projects were those with most effort spent on specifying the requirements before the projects were initiated. The survey, and review, paper [6] reports "functional, performance, and reliability requirements and scope are not documented" as the second most important software project risk factor. A survey of software managers reports that they considered requirement volatility among the top software failure risk factors [7].

The project survey reported in [8] finds a negative correlation between requirement changes and cost control. Similarly, the study in [9] reports a negative effect of requirement changes on product performance, measured as system reliability, ease of use, ability to meet users' requirements, and user satisfaction. The same study also reports a negative effect of requirement changes on project performance measured as budget and schedule control.

The survey of software projects reported in [10], which examined the connection between increases in the requirement scope and the degree of client satisfaction with the project, found that a large requirement increase was connected with more project failures for traditional projects but not for agile projects. Although this finding is highly relevant for the study in this paper, and indicates that the choice of development method matters for the effect of requirement changes on project outcomes, the study had limitations. The traditional projects, on average, were much larger (and were likely to be more complex) and had a higher number of requirement changes than the agile projects. The difference in how requirement changes and client satisfaction were connected, therefore, could be a result of factors other than the choice of development method.

A survey of 399 agile software projects [11] reports that agile teams' ability to respond to requirement changes, measured as the proportion of change requests implemented (response extensiveness) and the speed (response efficiency), was positively connected to the ability of the software functionality delivered to meet the requirements, achieve goals, and satisfy users. A high response extensiveness had no large effect on the other project success dimensions, suggesting that responding to additional requirement changes was connected to better client satisfaction and benefits, without harming the other project success measures.

The survey reported in [4] found that agile projects with a flexible scope had almost twice as high a success rate as agile projects without a flexible scope. This result may be interpreted as supporting the benefit of adopting the agile principle of welcoming change.

An inherent problem in studying requirement specifications and requirement volatility is that we do not have commonly accepted and easy-to-implement measures of the size and complexity of a requirement change, the types of requirement changes, or the degree of change of a requirement specification [12]. The negative, or positive, consequences of a requirement change may depend, for example, on whether the change is only minor or leads to a large amount of rework, whether due to improved insight into client needs or external changes, and whether the change appears early or late in the project.

The great majority of previous research results, as far as we can see, suggest that more requirement changes are connected with more problematic and less successful software development. However, most of the research was conducted in a non-agile software development context. Therefore, whether the "embrace change" claim made by agile software professionals has some merit remains unproved. This is in particular the case, taking into account that more recent studies [4, 10, 11] give some hope for positive effects of requirement changes in the context of agile projects.

3. The survey

3.1. Design

The survey requested the participants, who were project managers and software developers from different organizations participating in a seminar on software cost estimation, to provide information about their last completed software projects with budgets of more than €100,000. Seventy-five responses were received. Five of the responses were incomplete, i.e., included "don't know" responses, and therefore removed, leaving 70 complete responses. Each response included information about the following:

- The respondent's role (free text) and length of experience (years).
- The budget category of the project: €100,000–1 million[3], €1–10 million, >€10 million.
- The type of development method used: Agile, Waterfall/Traditional, Mixed/other.
- Frequency of completed software functionality delivered to production or to user evaluation with feedback (this variable was included based on the results in [4], where delivery frequency was an essential variable for success with agile projects): None, 1–4 per year, More than 4 per year.
- Percentage of requirements added, removed, and/or updated: 0–10%, 10–30%, More than 30%.
- Reasons why the requirements were added, removed, and/or updated:[4] Learned about client

[3] The original questions were in Norwegian and used Norwegian currency. The budget values are approximate monetary values assuming that EUR 1 = NOK 10.

needs or gained insight during the project execution, External changes, Insufficient requirement analysis before the project started, Other reasons.

- Perceived project outcome (for each of the project success dimensions below, the respondent was requested, based on his/her evaluation, to choose one of the outcome categories: Very successful – Successful – Acceptable – Problematic – Very problematic): Client benefits, Technical quality of software, Cost control, Time control, Work efficiency.

We categorized the total performance (outcome) of a project as follows:

- Successful: The project was evaluated as very successful or successful on all five success dimensions (client benefits, technical quality of software, cost control, time control, and work efficiency)
- Acceptable: The project was not successful but was evaluated as at least acceptable on all five success dimensions.
- Problematic: The project was evaluated as problematic or very problematic on at least one of the success dimensions.

In the analysis section, we mainly present analyses based on the proportion of successful and problematic projects. The proportion of acceptable projects can be derived from the proportion of successful and problematic projects.

Due to few responses for some of the categories, we decided to join the categories "Mixed/other" and "Waterfall/traditional", creating the category "Non-agile". This gives very rough development method categories, but enables a comparison of what was considered agile by the respondents with the other projects. Similarly, the few responses with less than 10% requirement changes led us to join this category with the 10-30% category. The choice of 30% as our boundary value is to some extent arbitrary, but hopefully useful to gain some insight into difference of project with much and with less requirement changes.

3.2. Limitations

When interpreting the survey results, the following limitations should be kept in mind:

- The sample of respondents and their projects is not necessarily representative of other contexts. While this may strongly affect the characteristics of the data set, it may have less impact on the connections we focus on in this study. There is clearly a need for more studies to assess the generality and context dependencies of the results identified.
- The survey asked for the perceived (subjective) performance related to the success dimensions and did not use more objective measures of the project outcome, what they meant by use of agile or non-agile development methods, and a requirement change. Although this makes the evaluations highly subjective, and there will be differences in use of terms among the respondents, it may also have advantages. It may be, for example, that delivering the software one month late is acceptable in one project context but leads to large problems in another context. Mechanical evaluations of measured time overrun may not enable such meaningful distinctions.
- The respondents (34% were project managers or team leaders and 66% were software developers) were all from the provider side of the projects. This may have affected the assessment of the projects' success. The results of a similar survey (see [4]) found, however, that providers and clients tend to give similar evaluations of software projects, even when evaluating the client benefits. In addition, a role bias is mainly a problem for the main (interaction) analyses in this paper if the role bias is different for different development methods, which we believe is not the case.
- The number of responses is low for the interaction analyses of this paper, especially for non-agile projects. This limits the robustness of the results, excludes the use of tests for statistical significance, and points to the need for follow-up studies to validate the findings.

There is no guarantee that the respondents had the required information about the project, even though they chose to respond and had the option of leaving questions unanswered or using the don't know category. The respondents' experience, which, on average, was 14 years (only 6 respondents had less than 4 years of experience), gives some confidence that they were sufficiently competent to possess the required information.

3.3. Results

As can be seen in Table 1, 43% of the projects had more than 30% requirement changes (inserted, removed or updated requirements) during project execution. On average, the projects with more than 30% requirement changes were somewhat more successful (27% of them were successful) than those with less than 30% requirement changes (18% of them were successful). The projects with more than 30% requirement changes were also, on average, slightly less problematic (33% of them were problematic) than those with less than 30% changes

[4] It was possible to give more than one reason for the requirement changes. Twenty-five percent of the respondents did this.

(37% of them were problematic). This not substantial difference in project outcomes related to requirement changes hides, however, a large difference when the development method is included as an interacting variable (see Fig. 1 and Fig. 2).

Table 1. Project characteristics

Variable	Characteristics
Project size	58% less than €1 million 33% more than €1 million 9% larger than €10 million
Development method	74% agile 26% non-agile
Delivery frequency	36% 4 or fewer per year 64% more than 4 per year
Requirement changes	57% less than 30% 43% more than 30%
Reason for change (more than one reason possible)	78% learning/insight 16% external change 27% insufficient up-front analysis
Project outcome	25% successful 39% acceptable 36% problematic

Fig. 1 shows that the proportion of successful projects increased (from 15% to 31%) with more requirement changes for agile projects but decreased (from 25% to 0%) for non-agile projects. Notice that the proportion of successful non-agile projects is higher than that of the agile projects when there are fewer than 30% requirement changes but substantially lower when there are more requirement changes. There were only four non-agile projects with more than 30% requirement changes, which means that we should interpret the decrease in the success rate for the non-agile projects with great care. Previous research (see Section 2), however, supports a decrease in the success rate for non-agile software projects with many requirement changes. The results for non-agile projects with many requirement changes, although based on very few observations, therefore, are in accordance with some previous results.

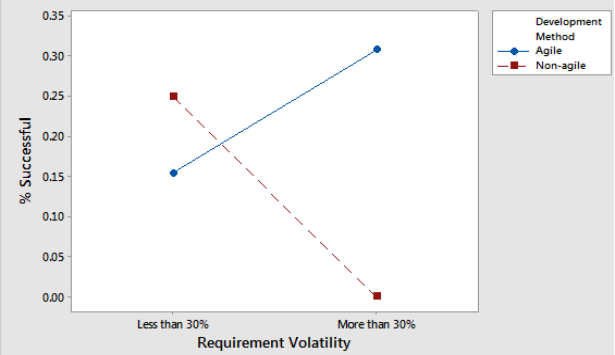


Figure 1. Development methods, requirement change, and proportion of successful projects.

Fig. 2 shows a weak decrease (from 31% to 27%) in the proportion of problematic projects with more requirement changes for agile projects. The corresponding observation for non-agile projects is an increase (from 50% to 75%) in the proportion of problematic projects. As before, the number of non-agile projects with more than 30% requirement changes are few, and the results for non-agile projects with many requirement changes, consequently, are not very robust.

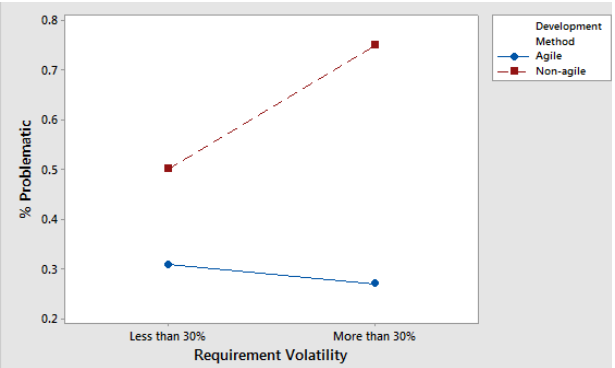


Figure 2. Development methods, requirement change, and proportion of problematic projects.

Table 2 shows the proportion of projects evaluated as "very successful" or "successful" for each of the success dimensions, development methods, and requirement change categories. The data suggest that the use of agile development methods is connected with an increase in the proportion of successes from less than 30% to more than 30% requirement changes for all success dimensions, but especially for the success dimensions technical quality (72% - 48% = 24% point increase) and cost control (50% - 38% = 12% point increase).

Table 2. Proportion projects evaluated to be "successful" or "very successful" for each success dimension, development method, and requirement change category

Requirement change		Less than 30% change		More than 30% change	
Development method		Agile	Non-agile	Agile	Non-agile
Success dim.	Client benefits	77%	54%	85%	25%
	Technical quality	48%	42%	72%	0%
	Cost control	38%	50%	50%	0%
	Time control	46%	50%	54%	0%
	Work efficiency	62%	42%	67%	50%

In total, answering our RQ1, the results displayed in Fig. 1, Fig. 2 and Table 2 suggest that there is an interaction effect from development method on the connection between requirement change and project outcome. The agile software projects performed better in contexts with more requirement changes, while the opposite was the case for the non-agile projects.

Motivated by the results in [4], and answering RQ2, we expected to see a difference in the success rate between agile projects with many (more than four per year) and with fewer (four or fewer per year) deliveries of completed software functionality to production or to user evaluation. This is what we see in Fig. 3 and Fig. 4.

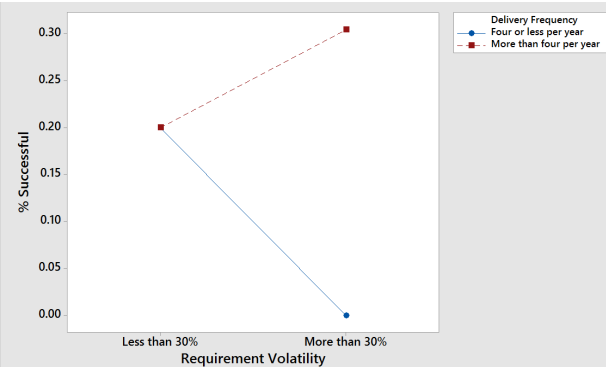


Figure 3. Delivery frequency, requirement change, and success for agile projects.

As can be seen in Fig. 3 and Fig. 4, the agile projects were more successful and less problematic in contexts with many requirement changes when the projects had frequent deliveries (more than 4 per year) to production or proper user evaluation of completed functionality. Frequent delivery did, however, not make any difference in the project's success rate and gave only a slightly lower rate of problematic projects when there were fewer requirement changes. To what extent frequent deliveries to production, with feedback, causes more requirement changes, leads to project success in situations with more requirement changes, or indicates a development context with success inducing elements, such as more involved clients, is hard to see from the data. This is another topic for future examination.

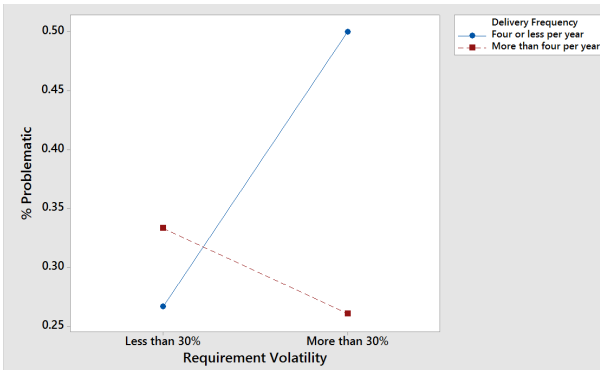


Figure 4. Delivery frequency, requirement change, and problems for agile projects.

Requirement changes may differ considerably in complexity, implications for rework, and how much the changes disrupt the project execution. As an initial step in understanding the influence of the type of requirement change on the project performance, we examined the effect of many requirement changes on project performance for the three reasons (learning or better insight, external changes, and insufficient requirement analysis) individually. The results are displayed in Table 3. We include only the results for the agile projects, because there were too few observations to give similar, meaningful results for non-agile projects.

Table 3. Success and failure rate, per reason (agile projects only)

Req. change	Less than 30% changes			More than 30% changes		
Reason	Le	Ex	In	Le	Ex	In
Success	20%	0%	0%	30%	17%	20%
Accept.	55%	33%	50%	33%	50%	20%
Problem.	25%	67%	50%	22%	50%	60%

[i] Le = Learning/insight, Ex = External, In = Insufficient analysis

The data in Table 3 do not reveal a clear pattern connecting the reasons for and the degree of requirement changes. The proportion of successful projects increased and the proportion of problematic projects decreased with more requirement changes for all requirement change reasons. Notice, however,

the higher problem rates for agile projects where the requirement changes were categorized as externally induced or caused by insufficient analysis compared to when the requirement changes were categorized as caused by learning or better insight.

Contextual differences may explain the differences in how the requirement changes, development method, and project performance are connected. Many important contextual variables were not collected, such as how late the requirement change occurred and the skill of the development team. It might nevertheless be interesting to examine if there are essential differences between agile and non-agile projects based on the data we collected; see Table 4. The values related to "Reasons for changes" are the proportion of projects where the reason was believed by the respondent to have caused all or part of the requirement changes, if any, in the project. None, one, or more reasons could be provided for the same project.

Table 4. Context differences between agile and non-agile

Characteristic	Measure or category	Development method	
		Agile	Non-agile
Respondent's experience	Mean length of experience (years)	13	15
Budget size	Proportion costing less than €1 million	62%	44%
	Proportion costing more than €1 million	38%	56%
Requirement change	Proportion with less than 30% change	50%	76%
	Proportion with more than 30% change	50%	24%
Reason for changes	Proportion due to learning/insight during project execution	82%	67%
	Proportion due to external changes	25%	11%
	Proportion due to insufficient requirement analysis	22%	44%

As can be seen, there were fewer, but not substantially fewer, agile projects (38% vs. 56%) in the category of projects with a budget of more than €1 million, more agile projects (50% vs. 24%) in the category of projects with more than 30% requirement changes, and for agile projects, respondents were more likely to provide the requirement reasons "learning/insight" (82% vs. 67%) and "external changes" (25% vs. 11%) and less likely to give the reason "insufficient requirement analysis" (22% vs. 44%). There were no large differences in the average length of respondents' experience for agile and non-agile software projects (13 vs. 15 years). The directions of the contextual differences shown in Table 4 are not surprising. Agile development methods are more commonly used for smaller projects, agile projects receive more requirement changes, and agile software professionals are less likely to think about requirement changes caused by insufficient requirement analysis, given less emphasis on producing up front complete and detailed requirement specifications. The higher degree of externally induced requirement changes may indicate that agile methods were more frequently used in contexts with higher environmental (external factors-based) uncertainty. All these differences point at possible differences in development complexity, for example, slightly larger projects for non-agile and perhaps more requirement uncertainty for agile projects, which, in turn, may explain some of the observed differences in the project outcomes for agile and non-agile projects. There is, however, little that suggest that the identified differences in contexts, which are not very large, explain the reported differences in how well agile and non-agile software projects succeed in situations with much requirement changes.

4. Discussion and conclusion

Most software projects experience that requirements are added, removed, or changed during the project execution. In as much as 50% of the agile and 24% of the non-agile projects included in our survey, more than 30% of the requirements were added, removed, or updated during the project execution. Requirement changes may be viewed as a threat or as an opportunity. Traditionally, requirement changes have been viewed as a risk factor, that is, a threat to the success of a software project. Agile software developers, however, tend to view requirement changes differently. They tend to view changes as creating opportunities to deliver more client benefits, and view them as something that should be welcomed in software projects.

The present results provide support for both views. When agile methods were used, but only when used with frequent deliveries of completed functionality to productions or user evaluation, many requirement changes were connected to higher proportions of successful projects and lower proportions of problematic projects. For non-agile projects and agile projects without frequent deliveries

to production, the outcome was the opposite. Many requirement changes for such projects were connected to less successful and more problematic projects.

The connection examined in this paper, that is how the development method influences the connection between requirement changes and software project success, has not been much investigated empirically. The only previous study we were able to identify is the one reported in [10]. As reported in Section 2, that study found a positive connection between a large increase in requirements and more satisfied clients for agile but not for non-agile software projects. Although limited to added requirements, i.e., not including changed requirements, and using client satisfaction as the only success measure, this result is consistent with what we found.

In the present study, non-agile projects (see Table 4) were larger than agile projects but not by much, and we believe the difference is not large enough to explain the differences in project outcomes. Indeed, we found larger projects to be somewhat more successful and less problematic (33% successful and 30% problematic projects) than smaller projects (21% successful and 39% problematic projects).

The limited number of variables and observations in this study means that we were unable to gain much insight into the underlying mechanisms that created the difference in project performance for different levels of requirement changes and different development methods. We cannot, as discussed in Section 3, be sure that the observed differences between successful and problematic projects were caused by, as opposed to just correlated with, differences in development method.

It is perhaps not surprising that a development method designed for flexibility in scope and frequent requirement changes, that is, the agile software development method, leads to better project outcomes than traditional, non-agile, methods when there are many requirement changes. What is perhaps more surprising is that projects following the agile method, when including the agile practice of frequent delivery to client, did *better* when there were *more* rather than fewer requirement changes. Currently, we find it hard to suggest mechanisms that should make it easier to succeed with more rather than fewer requirement changes. We suspect that the use of agile development methods, but mainly when implementing a practice with frequent deliveries to production, combined with many requirement changes correlates with the presence of other, essential success factors. This may include success factors related to more competent and involved clients, better and more frequent feedback and learning during project execution, better benefits management processes, more skilled developer teams, and better software testing facilities [13].

These interpretation challenges, together with the study limitations discussed earlier, mean that there is a need for more, carefully designed studies that not only try to replicate our results and examine the connections, but also try to better understand the context, patterns, and mechanisms that lead to the differences. This may be important in an evidence-based attempt to improve requirement management practices and project outcomes.

Changes in requirements are here to stay, and our ability to manage them is essential for success in software development. The present results provide some evidence in support of that agile development methods, when implementing frequent deliveries to production or to user evaluation with feedback, are a good choice when expecting many requirement changes.

5. References

[1] Schmidt, R., K. Lyytinen, and P.C. Mark Keil, *Identifying software project risks: An international Delphi study*. Journal of management information systems, 2001. **17**(4): p. 5-36.

[2] Sommerville, I. and P. Sawyer, *Requirements engineering: a good practice guide*. 1997: John Wiley & Sons, Inc.

[3] Erickson, J., K. Lyytinen, and K. Siau, *Agile modeling, agile software development, and extreme programming: the state of research.* Journal of database Management, 2005. **16**(4): p. 88.

[4] Jørgensen, M., *A survey on the characteristics of projects with success in delivering client benefits.* Information and Software Technology, 2016. **78**: p. 83-94.

[5] Serrador, P. and J.K. Pinto, *Does Agile work?—A quantitative analysis of agile project success.* International Journal of Project Management, 2015. **33**(5): p. 1040-1051.

[6] Kappelman, L.A., R. McKeeman, and L. Zhang, *Early warning signs of IT project failure: The dominant dozen.* Information systems management, 2006. **23**(4): p. 31-36.

[7] Tiwana, A. and M. Keil, *The one-minute risk assessment tool.* Communications of the ACM, 2004. **47**(11): p. 73-77.

[8] Zowghi, D. and N. Nurmuliani. *A study of the impact of requirements volatility on software project performance*. in *Software Engineering Conference, 2002. Ninth Asia-Pacific*. 2002. IEEE.

[9] Govindaraju, R., et al., *Requirement volatility, standardization and knowledge integration in software projects: an empirical analysis on outsourced IS development projects.* Journal of ICT Research and Applications, 2015. **9**(1): p. 68-87.

[10] Suma, V. and K. LakshmiMadhuri. *Influence of Scope Creep on Project Success: AComparative Study between*

Conventional ApproachVerses Agile Approach. in *IEEE International Conference on Advanced research in Engineering and Technology (ICARET)*. 2013.

[11] Lee, G. and W. Xia, *Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility.* Mis Quarterly, 2010. **34**(1): p. 87-114.

[12] McGee, S. and D. Greer, *Towards an understanding of the causes and effects of software requirements change: two case studies.* Requirements Engineering, 2012. **17**(2): p. 133-155.

[13] Jørgensen, M., P. Mohagheghi, and S. Grimstad, *Direct and indirect connections between type of contract and software project outcome.* International Journal of Project Management, 2017. **35**(8): p. 1573-1586.

Do Agile Methods Work for Large Software Projects?

Magne Jørgensen(✉)

Simula Research Laboratory, 1364 Fornebu, Norway
magnej@simula.no

Abstract. Is it true that agile methods do not scale well and are mainly useful for smaller software projects? Or is it rather the case that it is particularly in the context of larger, typically more complex software projects that the use of agile methods is likely to make the difference between success and failure? To find out more about this, we conducted a questionnaire-based survey analyzing information about 101 Norwegian software projects. Project success was measured as the combined performance of the project regarding delivered client benefits, cost control, and time control. We found that that projects using agile methods performed on average much better than those using non-agile methods for medium and large software projects, but not so much for smaller projects. This result gives support for the claim that agile methods are more rather than less successful compared to traditional methods when project size increases. There may consequently be more reasons to be concerned about how non-agile, rather than how agile methods, scale.

Keywords: Agile development methods · Project size · Project success

1 Introduction

Much has been written about the extent to which agile methods are suitable for large software projects. An early attempt to summarize what we know about agile methods and their success when used in large software projects, authored by Dybå and Dingsøyr [1], concludes: "The evidence […] suggests that agile methods not necessarily are the best choice for large projects." Similarly, the review published by Jalali and Wohlin [2] finds: "[…] there is not sufficient evidence to conclude that Agile is efficiently applicable in large distributed projects." More recent reviews, see for example [3, 4], emphasize challenges related to the use of agile methods for large software projects and, similarly to the previous reviews, report little or no evidence to support the use of agile methods for large software projects. Not only is much of the research literature sceptical about the use of agile methods for large software projects, but several software professionals also seem to think that agile methods are mainly for smaller software projects.[1] It is, in addition, not difficult to find examples of failed, large-scale agile

[1] For an example of an opinion-based argumentation of why agile is not useful for large projects, see blog.inf.ed.ac.uk/sapm/2014/02/14/agile-methodologies-in-large-scale-projects-a-recipe-for-disaster/. This blog post concludes that "*Large-scale development projects are serious business: agile development has no place here.*"

J. Garbajosa et al. (Eds.): XP 2018, LNBIP 314, pp. 179–190, 2018.
https://doi.org/10.1007/978-3-319-91602-6_12

software projects.[2] A comprehensive review of experience reports and case studies on the challenges and success factors regarding the introduction of agile in large-scale software development can be found in [5].

There are also reported cases where agile methods have been successfully used for large software projects, see for example [6], and reports where agile methods are claimed to have had a positive impact on the outcome of large software projects, see for example [7, 8]. Finally, there are guidelines on how to succeed with large-scale agile projects, such as [9], which claim to be based on the successful completion of large software projects using agile methods.

These diverging results and opinions on the use of agile on large software project may appear to be confusing. There are, however, several reasons why we should not expect consistent results and opinions about the effect of using agile methods on larger software projects:

- *We do not have a clear, commonly agreed upon understanding of what it means to work agile*. Agile is not a well-defined method, but rather a set of values, principles, and practices. There are consequently many good and bad ways of implementing and using agile methods. There may, in addition, be external factors that complicate the use of good agile, such as the use of fixed price contracts or insufficient involvement by the client [10]. The same problems are present for non-agile methods, which may include an even larger variety of practices. There are good and bad ways of using most software development methods and it is frequently not clear when it is the inexperience and lack of skill in using a method and when it is inherent flaws in a method that contribute to software project failures.
- *The development method is only one of many factors affecting the success of a software project.* Other factors, especially the level of provider and client competence, may be even more important to explain the outcome of large software projects.
- *We do not agree on what a large software project is.* A large software project may be defined relatively to those that an organization is used to completing or with absolute measures such as budget size, number of developers, complexity, or number of development teams [11]. In addition, the difference between a large project (e.g., a project consisting of two teams and costing 10 million Euros) and a mega-large project (e.g., a project consisting of ten teams and costing 100 million Euros) may be substantial.
- *We see it when we believe it (confirmation bias).* People are good at summarizing experience in a way that defends their beliefs. As documented in [12], those who believe in the benefits of agile will tend to find evidence supporting the use of agile even in random project data without any true patterns connecting development method and project success. One example of how to confirm a strong belief in agile

[2] See, for example, the UK National Audit Office report: www.nao.org.uk/wp-content/uploads/2013/09/10132-001-Universal-credit.pdf. It is from the report not clear to what extent they think that it was agile development itself, the actual implementation and use of agile or the project's lack of experience with the use of agile that contributed to the failure of the project.

(or other) development methods is to categorize a software project as non-agile, or at least not using agile methods properly, if it fails, i.e., if it works it is agile, if it fails it is not true agile.

Despite the above methodological problems we may be able to find out more about the scalability of agile methods by systematically collecting empirical evidence. If large software projects using agile methods typically perform better than projects using other methods, then this supports the claim that agile methods do scale to larger projects. It may give this information even if we do not know exactly how agile was implemented and used by the projects, are unable to use a commonly accepted and good definition of what a large project is, and there are other factors that also matter for success. Many companies may have adopted agile methods just recently, which means that if we find that agile software projects perform worse, but perhaps not much worse, than non-agile as the project size increases, we may not be able to conclude that agile methods will not work on larger software projects. It may then improve as their competence in using the methods improves.

In this paper we empirically compare agile and non-agile software development projects by surveying a set of projects, collecting information about their size (as measured by their budget), their use of development methods, and their degree of success. The research question of our study is:

How is the relationship between project size, as measured by its budget, and success affected by the development method?

As indicated earlier in this section, there are many studies on the use of agile methods on large-scale software projects, and there are many strong opinions about which method is the better to use on large projects. In spite of this, we have been unable to find peer-reviewed research articles empirically analysing size-dependent differences in success of projects using agile and non-agile development methods. A non-peer reviewed study by the Standish Group from 2016[3] reports that projects using agile development methods performed better than those using waterfall-based methods for small, medium, and large project sizes, and particularly the largest projects. For the largest projects, the failure rate was 42% for waterfall projects and 23% for agile projects. For the smallest project, the difference is smaller, with an 11% failure rate for waterfall and a 4% failure rate for agile projects. This study indicates that agile methods is not only well suited for large projects, but also increasingly more suited as the project size increases. This is, to our knowledge, the only related work we can compare our results with.

[3] There are reasons to be sceptical about the results published by the Standish Group; see our comments on their survey methods on a previous survey in [13]. In its 2016 report the Standish Group (www.standishgroup.com), improved the definition of success to include not only being on time, on cost, and with the specified functionality, but also that the project delivers satisfactory results (blog.standishgroup.com/post/23). Satisfactory results include, they claim, client value. This improvement, given that it is properly integrated in their survey and that they have improved their sampling of projects, may make their recent results more valid and useful.

The remaining article is organized as follows. Section 2 describes the survey design, limitations, and results. Section 3 briefly discusses the results and concludes.

2 The Survey

2.1 Survey Design

The respondents of the survey were participants at a seminar on management of software development projects in Oslo, Norway, March 2015.[4] All participants were asked to provide information about their last project, including:

- The respondent's role in the project.
- The project's outcome in terms of client benefits, cost control, and time control.
- The project's budget.
- The project's use of agile practices, and the respondent's assessment of how agile the project had been.

We received information about 108 projects. An examination of the responses showed that seven of them did not include the required information regarding one or more of the variables used in our analysis. Removing these left 101 valid responses in the data set.

Characteristics of the respondents and their projects include:

- *Role*: 56% of the respondents were from the client side and 44% from the provider side.
- *Client benefits*: 35% were categorized as "successful," 55% as "acceptable," and 10% as "unsuccessful" or "failed."
- *Cost control*: 30% were categorized as "successful," 32% as "acceptable," and 38% as "unsuccessful" or "failed."
- *Time control*: 37% were categorized as "successful," 32% as "acceptable," and 31% as "unsuccessful" or "failed."
- *Budget*: 48% of the projects had a budget less than 1 million Euros, 25% between 1 and 10 million Euros, and 27% more than 10 million Euros.[5]
- *Agile practices*: When asked to rank their project with respect to how agile it was from 1 (very agile) to 5 (not agile at all), 17% responded with 1, 25% with 2, 40% with 3, 14% with 4, and 4% with 5.

The participants were asked to name the agile practices they had used in their last project. Comparing those descriptions, emphasizing the use of product backlogs, frequent/continuous delivery to client, the use of scrum or similar management processes, autonomous teams, and the use of velocity to track progress, with responses regarding the degree of agility of the project using the scale from 1 to 5, we found it

[4] Results from this survey have not been published earlier, but the design and project performance measures are similar to those in the survey published in [14].

[5] The original survey was in Norwegian and used Norwegian Kroner (NOK) as currency. The Euro-values are the approximate values corresponding to the NOK-values.

reasonable to cluster the projects as "agile" if the response was 1 or 2, "partly agile" if the response was 3, and "not agile" if the response was 4 or 5. There were, however, no simple connection between the self-assessed degree of agility (using the scale from 1 to 5) and the implemented agile practices. This makes the development category boundaries, especially the boundary between agile and partly agile, to some extent fuzzy and subjective. While this may limit the strength of the analysis, it is clear from the analysis that those categorized as agile on average have more agile practices than those categorized as partly agile. While we believe that this is sufficient for meaningful analyses, it is important to be aware of that degree of agility in our study is based on the respondents subjective assessment.[6]

Our measure of a project's level of success used a combination of three success dimensions: client benefits, cost control, and time control. To be categorized as "acceptable", we require a score of at least "acceptable" on all three dimensions. Fifty-four percent of the projects were categorized as acceptable using this definition. Notice that the inverse of "acceptable" (46% = 100% − 54%) is the set of projects assessed to have a non-acceptable outcome on at least one of the success dimensions, i.e., the set of "problematic" projects. To be categorized as "successful," we require that all three dimensions should be assessed as "successful." Only 12% of the projects belonged to that category.

2.2 Limitations

The survey has a number of limitations that it is important to be aware of when interpreting the results, including:

- *Representativeness*. Our sample consists only of Norwegian software projects and is a convenience sample based on input from people visiting a seminar on software project management. The common use of agile methods in our data set suggests that many of the companies represented by the participants had (possibly much) experience in the use of agile methods. From more in-depth studies of software projects in similar contexts, see [10], and common sense we know that companies tend to have more problems in the initial phase when they introduce agile methods compared to subsequent projects. The level of agile maturity and other largely unknown sample characteristics, may affect how valid it is to extrapolate our results to other context.
- *Perception, not measurement*: Several of the survey questions, particularly those related to project outcome, are based on the respondents' perceptions, not measured data. This has some drawbacks, for example, different people may have different viewpoints regarding the same project. It may also have some advantages. The degree of success in time control, for example, may be more meaningfully assessed subjectively. In one context, a 10% time overrun may point to a time control failure, while in another context, the same overrun may be acceptable.

[6] The set of agile practises, combined with the project's own assessment of degree of agility, of a project and other project data used in the analyses will be sent to interested readers upon request to the author.

- *Role bias*. We decided to join the responses of those on the client and the provider side, even though there may have been systematic differences in their responses. For example, those in the client role seem to have been less critical than those in the provider role when assessing the outcome of the projects. Using our measure of acceptable outcomes, those on the client side found 66% of the projects to be acceptable, while the figure was 46% when assessed by those on the provider side. Those on the client and the provider side gave however approximately the same average score regarding client benefits, i.e., 37% of the projects assessed by the clients were successful regarding client benefits, while the figure was 32% when assessed by the providers. If the role bias is not dependent on the degree of use of agile methods, which we believe is the case, joining the responses of the two roles will not affect the direction of the interaction effect reported later in this paper.
- *Correlation vs. causation*. There may be systematic differences in the non-measured characteristics of the agile and the non-agile software projects. In particular, it may be that the client and/or provider competence was higher for those using one type of development method, e.g., providers and clients using agile methods may have been more competent than those using non-agile methods. This will exaggerate the effect of a development method if the most competent clients and providers are more likely to choose the better development method. As with role bias, the direction of the interaction effects from project size is less likely to be affected by such differences.
- *Few observations*. There are few projects for several combinations of development method and project size category, in particular for the non-agile projects. The low statistical power means that tests of the statistical significance of the interaction effect on the development method are not feasible. It also implies that there are limitations regarding the robustness of our results and that small to medium large differences in success rates are caused by random variance in outcomes. Our results should consequently be understood as initial, exploratory results to be followed up with more empirical research.
- *Size vs. complexity.* We categorize project size based on the project's budget. While the budget is likely to reflect the amount of effort spent, it does not necessarily reflect the complexity of the project. There may consequently be relevant differences between large and simple, and large and complex software projects that our analysis is unable to identify.

2.3 Results

The results section emphasizes key takeaways from our study, especially those related to the connection between project size, development method and project outcome.

Table 1 gives the proportion of observations per budget and development method category. It shows that agile and partly agile methods are frequently used even for the largest projects. They are used in 33% and 56% of the largest projects, respectively. While this does not say anything about the usefulness or harm of using agile methods as project size increases, it documents that many of the software professionals involved considered agile and partly agile development methods to be useful for larger projects. Notice the increase in use of partly agile as the project size increases from medium

to large. This may suggest that some software professionals believe less in working fully agile when projects get large.

Table 1. Proportion use of development method per budget size category

Budget size	Agile	Partly agile	Not agile	# projects
Small	37% (18)	42% (20)	21% (10)	48
Medium	58% (15)	19% (5)	23% (6)	26
Large	33% (9)	56% (15)	11% (3)	27
# projects	42	40	19	101

Table 2 and Figs. 1, 2, 3, 4 and 5 show the interacting effect of development methods on the connection between project size and:

(i) Proportion of acceptable projects (Fig. 1)
(ii) Proportion of successful projects (Fig. 2)
(iii) Mean score for client benefits (Fig. 3)
(iv) Mean score for cost control (Fig. 4)
(v) Mean score for time control (Fig. 5)

The scores of the success dimensions are coded with 4 for successful, 3 for acceptable, 2 for unsuccessful, and 1 for failed projects. This scale is, according to measurement theory, an ordinal scale. We believe, nevertheless, that the mean scores (which strictly speaking require at least an interval scale) give a good indication of the typical outcome regarding client benefits, cost control, and time control.

Our results do not support the claim that projects using agile or partly agile methods do worse than non-agile methods on larger projects. Quite the opposite, the data indicates that large projects using agile or partly agile methods were more likely to be assessed as acceptable than medium large projects using these methods. The non-agile projects performed reasonably well for the smallest projects, just a little worse than the agile and partly agile projects, but very badly on the medium and large software projects. In fact, among the non-agile projects of medium and large size, there were no projects in our data set that met the criterion of being perceived acceptable or better on all success criteria. Although consisting of a small sample, only nine projects used non-agile methods for medium and large projects; this weakly indicates that it is non-agile rather than agile methods that have most problems with larger software projects. This result—i.e., that non-agile methods score relatively poorly compared to agile projects and that the performance difference increases as the project size increases—is similar to that reported in the Standish Group's Chaos Report for 2016.

For most of the measures, there were not much difference in the assessed outcome for projects using agile and only partly agile. The most notable exceptions were projects assessed to be successful in all three dimensions (Fig. 2), wherein agile performed better than partly agile for large, but worse for medium large projects.

Table 2. Success with use of development method per budget size category

Budget size	Agile	Partly agile	Not agile
Total success (% acceptable)			
Small	72%	60%	60%
Medium	46%	40%	0%
Large	67%	60%	0%
Total success (% successful)			
Small	28%	10%	10%
Medium	7%	20%	0%
Large	11%	7%	0%
Client benefits (mean score)			
Small	3.5	3.1	3.1
Medium	3.3	3.4	3.0
Large	3.4	2.8	2.3
Cost control (mean score)			
Small	3.2	2.9	2.9
Medium	3.5	2.8	1.8
Large	3.4	2.9	1.0
Time control (mean score)			
Small	3.3	3.3	2.8
Medium	2.9	2.6	1.7
Large	2.8	2.9	2.5

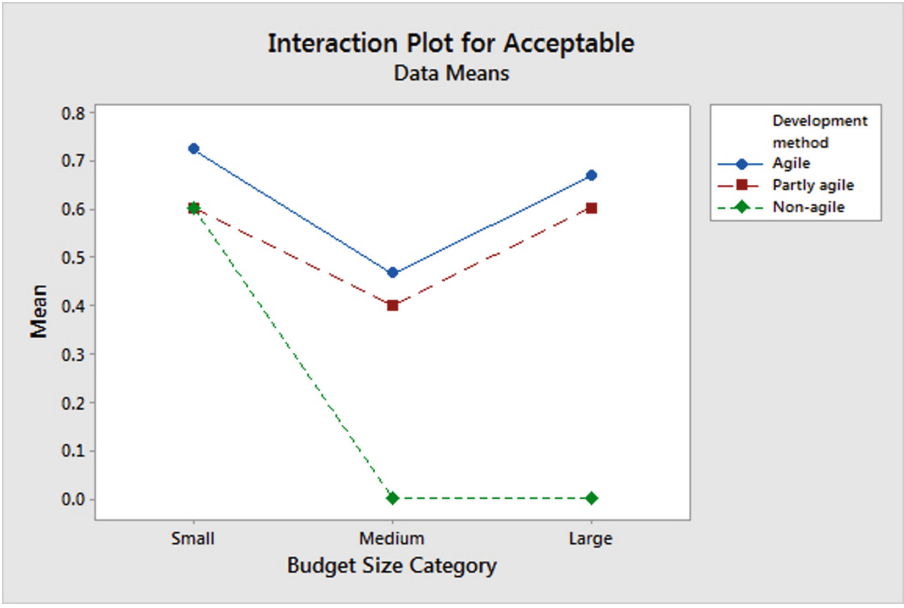


Fig. 1. Proportion of acceptable projects

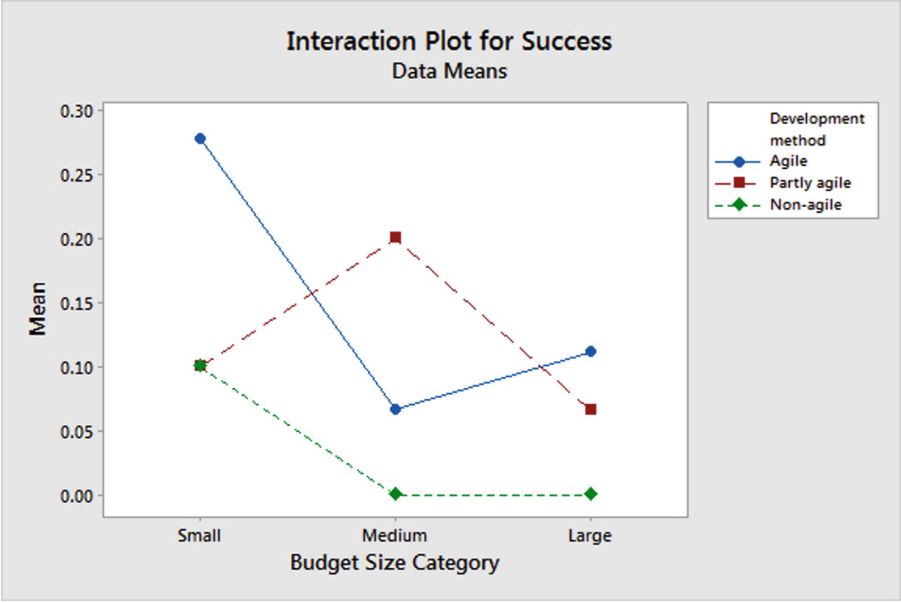


Fig. 2. Proportion of successful projects

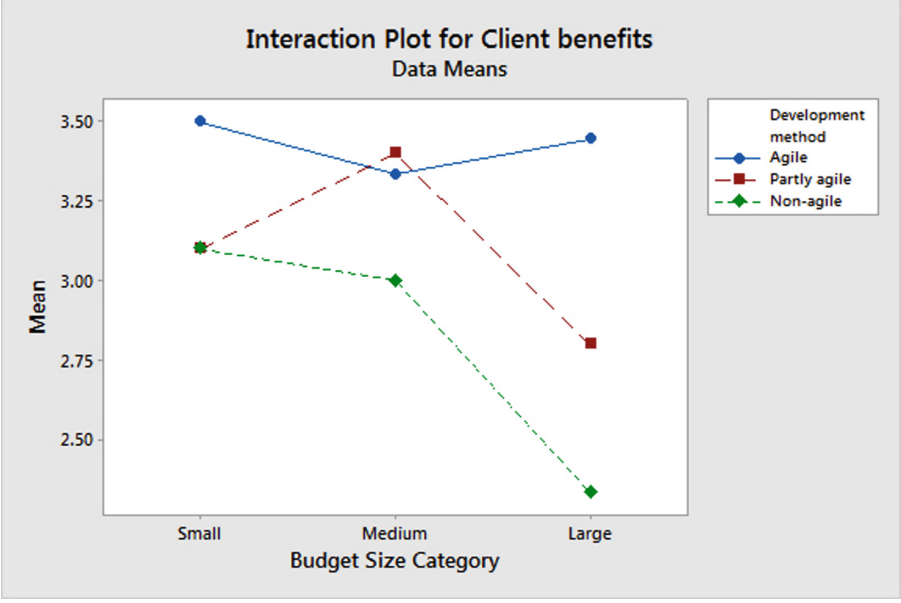


Fig. 3. Client benefits

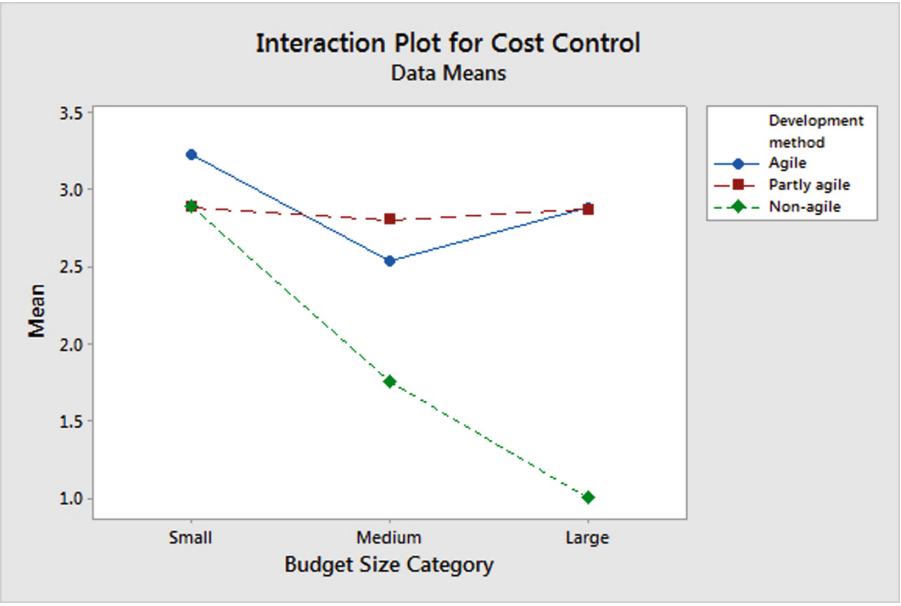


Fig. 4. Cost control

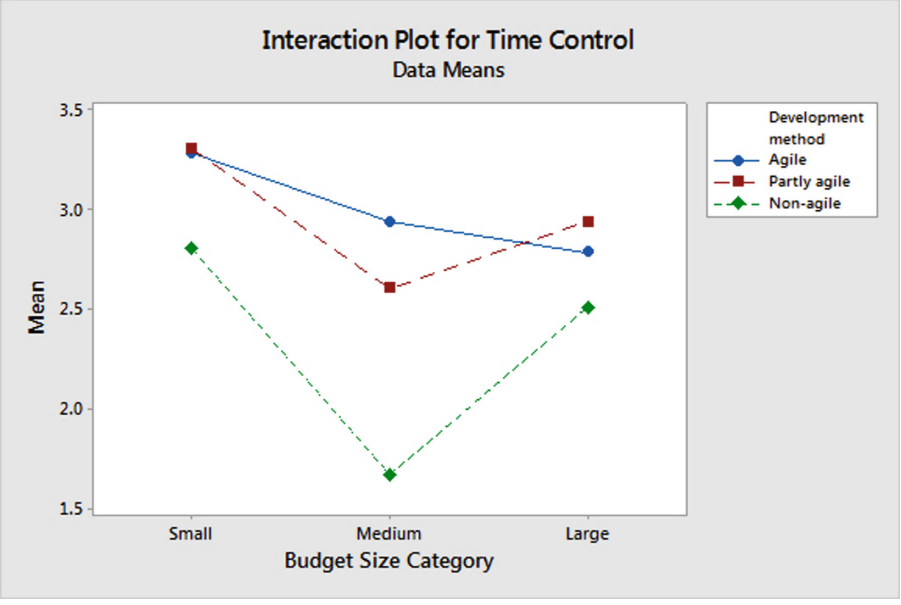


Fig. 5. Time control

3 Discussion and Conclusion

There are reasonable arguments both in favour and against good performance of agile methods on large projects. An example of an argument in favour of their use is that it is increasingly more unlikely that requirements will remain stable as the size of the software project increases. The understanding of needs is likely to change during the course of the project, and there will most likely be external changes leading to requirement changes. Agile development methods, implementing a process where change is a more integrated part, may consequently be better able to deal with the high requirement volatility of many large projects [10, 14]. An example of an argument sometimes used against the use of agile methods on large software projects is that the lack of upfront planning and architectural thinking, make projects more risky with increasing size.[7] Consequently, it is possible to analytically argue in favour of both agile and more plan-driven, non-agile software development methods. To find out which argumentation in practice is the stronger, and whether agile methods typically are good for large projects, requires empirical evidence.

The results from this study do this and provide evidence about how projects with agile practices perform on important success criteria. As pointed out in Sect. 2.2 there are several threats to the validity of our results, but the results do give some evidence in support of that the typical medium and large software projects using agile practices perform acceptably on essential success criteria. This was not the case for typical software projects using non-agile methods in our data set. Consequently, our data suggests that the question is not so much whether agile methods work well for large software projects, but rather how well non-agile software development methods work for such projects. Large projects are inherently risky, and our data suggests that the failure risk is reduced rather than increased with the use of agile methods instead of non-agile methods.

References

1. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: a systematic review. Inf. Softw. Technol. **50**(9), 833–859 (2008)
2. Jalali, S., Wohlin, C.: Global software engineering and agile practices: a systematic review. J. Softw. Evol. Process **24**(6), 643–659 (2012)
3. Khalid, H., et al.: Systematic literature review of agile scalability for large scale projects. Int. J. Adv. Comput. Sci. Appl. (IJACSA) **6**(9), 63–75 (2015)
4. Turk, D., France, R., Rumpe, B.: Limitations of agile software processes. arXiv preprint arXiv:1409.6600 (2014)
5. Dikert, K., Paasivaara, M., Lassenius, C.: Challenges and success factors for large-scale agile transformations: a systematic literature review. J. Syst. Softw. **119**, 87–108 (2016)

[7] See for example: www.6point6.co.uk/an-agile-agenda, which predicts that UK is wasting 37 billion GBP annually on failed agile projects. This number is based on a survey of CIOs, suggesting a 12% complete failure rate of agile projects. They did not calculate the waste on failed non-agile projects.

6. Dingsøyr, T., et al.: Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. Empir. Softw. Eng. **23**, 490–520 (2016)
7. Lagerberg, L., et al.: The impact of agile principles and practices on large-scale software development projects: a multiple-case study of two projects at ericsson. In: ESEM 2013. IEEE, Baltimore (2013)
8. Ebert, C., Paasivaara, M.: Scaling agile. IEEE Softw. **34**(6), 98–103 (2017)
9. Elshamy, A., Elssamadisy, A.: Applying agile to large projects: new agile software development practices for large projects. In: Concas, G., Damiani, E., Scotto, M., Succi, G. (eds.) XP 2007. LNCS, vol. 4536, pp. 46–53. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73101-6_7
10. Jørgensen, M., Mohagheghi, P., Grimstad, S.: Direct and indirect connections between type of contract and software project outcome. Int. J. Proj. Manag. **35**(8), 1573–1586 (2017)
11. Dingsøyr, T., Fægri, T.E., Itkonen, J.: What Is large in large-scale? *A taxonomy of scale for agile software development*. In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., Raatikainen, M. (eds.) PROFES 2014. LNCS, vol. 8892, pp. 273–276. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13835-0_20
12. Jørgensen, M.: Myths and over-simplifications in software engineering. Lect. Notes Softw. Eng. **1**(1), 7–11 (2013)
13. Jørgensen, M., Moløkken-Østvold, K.: How large are software cost overruns? A review of the 1994 CHAOS report. Inf. Softw. Technol. **48**(4), 297–301 (2006)
14. Jørgensen, M.: A survey on the characteristics of projects with success in delivering client benefits. Inf. Softw. Technol. **78**, 83–94 (2016)

Looking back on previous estimation error as a method to improve the uncertainty assessment of benefits and costs of software development projects

Magne Jørgensen
Simula Metropolitan,
Center for Digital Engineering
Oslo, Norway
magnej@simula.no

***Abstract*— Knowing the uncertainty of estimates of benefits and costs is useful when planning, budgeting and pricing projects. The traditional method for assessing such uncertainty is based on prediction intervals, e.g., asking for minimum and maximum values believed to be 90% likely to include the actual outcome. Studies report that the traditional method typically results in too narrow intervals and intervals that are too symmetric around the estimated most likely outcome when compared with the actual uncertainty of outcomes. We examine whether an uncertainty assessment method based on looking back on the previous estimation error of similar projects leads to wider and less symmetric prediction intervals. Sixty software professionals, with experience from estimating software project costs and benefits, were randomly divided into a group with a traditional or a group with a looking back-based uncertainty assessment method. We found that those using the looking back-based method had much wider prediction intervals for both costs and benefits. The software professionals of both groups provided uncertainty assessment values suggesting a left-skewed distribution for benefits and a right-skewed distribution for cost, but with much more skew among those using the looking back-based method. We argue that a looking back-based method is promising for improved realism in uncertainty assessment of benefits and costs of software development projects.**

Keywords—uncertainty assessment, software benefits and cost, controlled experiment

I. INTRODUCTION

There is no shortage of studies of human judgement documenting that people will give prediction intervals that are too narrow and too symmetric when asked to use the traditional uncertainty assessment method, i.e., the method based on giving minimum and maximum values with, typically, 90% confidence in including the actual value [2, 14, 18]. A typical result demonstrating the lack of correspondence between the confidence level and rate of including the actual effort (the hit rate) in the prediction interval is the one in [11], where software professionals giving traditional minimum–maximum intervals included the actual effort only 20–40% of the time, in spite of being instructed to be, typically, 90% sure to include the actual effort. Even after extensive feedback and training, the confidence level is typically much higher than the hit rate [5]. While knowledge of the problems with the traditional method for eliciting effort uncertainty intervals is not new, in particular when used in situations with high uncertainty, it is in common use and promoted, e.g., in the context of the PERT (Program Evaluation and Review Technique) tool and as part of the PMBOK (Project Management Book of Knowledge) [12, 16]. A possible reason for the promotion and widespread use of this method, frequently known as three-point estimates, is that the statistical theory behind it is sound, there are no clear alternatives, and there is typically little on-the-job feedback to show that the judgment-based input to the method frequently is strongly biased [11]. Unfortunate consequences of too narrow and symmetric uncertainty intervals are, amongst others, unrealistic cost-benefits analyses and too low budgets.

In previous papers we suggested and empirically evaluated an alternative method, based on the assumption that the distribution of the estimation accuracy of earlier, similar, software projects can be used to predict the uncertainty of new projects [6, 8-10]. Assume, for example, that a software professional wants to assess the uncertainty of a project that has been estimated to cost around 1 million USD. He or she looks back on estimation error experience from similar projects (memory-based or based on actual estimation error measurement) and reports that only 20% of them cost less than the estimated cost and around 10% cost more than twice the estimated cost. The software professional may be asked to add more empirical error data to provide the full empirical error distribution, but even these two data points (p20 = 1 mill. USD and p90 = 2 mill. USD) are sufficient to establish an uncertainty distribution, given the selection of a proper non-symmetric distribution belonging to the location-scale family [3, 13], e.g., a log-normal or gamma distribution. Figure 1 displays the log-normal distribution based on the above two estimation error data points. The cumulative error distribution is displayed in Figure 2. Figure 2 shows, for example, that the p50 (the value it is 50% likely to overrun) is around 1.3 mill. USD.

The results when using this alternative uncertainty assessment method have been good, suggesting increased realism compared to the traditional method [6, 17].

This paper extends our previous evaluations of the outlined alternative method by adding an analysis of the uncertainty assessment of benefits and of the benefits to costs ratio (return on investment) of software projects. In addition, the paper, compared to the previous papers, has a more explicit focus on differences in the skewness of the distributions provided by the traditional and the alternative uncertainty assessment method.

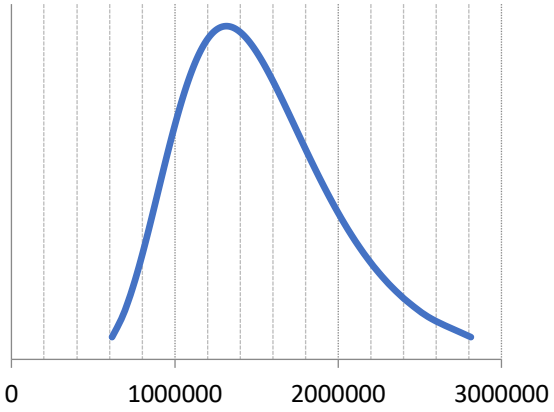


Fig. 1. Probability density cost distribution (log-normal)

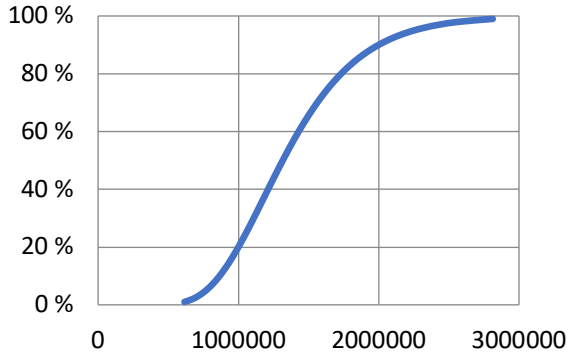


Fig. 2. Cumulative cost distribution (log-normal)

The hypotheses to be tested are the following:

H1: Those asked to look back (using the alternative method) will give wider and more *left-skewed* uncertainty intervals for software development benefits.

H2: Those asked to look back (using the alternative method) will give wider and more *right-skewed* uncertainty intervals for software development costs.

H3: The uncertainty assessment of those asked to look back (using the alternative method) will result in a lower benefits-to-costs ratio (return on investment) for software development costs.

The motivation for H1 and H2 is that we expect that software professionals instructed to look back will be reminded that much higher benefits and much lower costs than, respectively, the estimated benefits and costs are rare. Much lower benefits and much lower costs than estimated are, on the other hand, not uncommon. H3 will be true if H1 and H2 are true, but it is nevertheless interesting to examine on its own. Our expectation is that the difference in the expected benefits-to-costs ratio between those using the traditional and the alternative uncertainty assessment method is substantial.

II. Method

The participants were software professionals, mainly managers and projects leaders, attending a seminar on benefits management. The software professionals were first instructed to indicate their level of experience in benefits and costs estimation on a scale from 0 (none) to 5 (very high). Of the total of 65 responses received, from around 100 seminar participants, five had no experience with either costs or benefits management. These were removed from the analysis, leaving 60 responses for our analysis. The median experience level of the remaining participants was 3 (medium high) for benefits and 3 for costs estimation.

Following their responses on their experience level, the participants read the following description of a scenario based on a real-life software project (translated from Norwegian):

Digital solution for planning applications for construction work – uncertainty of costs and benefits

The municipal council of Oslo has decided to develop an IT system that will ease the planning applications of the non-professional and professional actors of the municipality desiring to build houses and other constructions.

The planning applications should, after secure login and selection of the relevant application type, be pre-registered with all the information the municipality already has about the person, company or property the application concerns. The field of the application where information about the construction work is to be provided should include links to the relevant regulations.

The user's application input will be checked automatically (where possible) with respect to the regulation adherence and correctness of the application type. Warnings should be provided if any non-adherences are found. The users should have the opportunity to download relevant maps, and have the functionality to put their own constructions on that map. All application information, e.g., feedback and decisions from the municipality, should be done through the software system. Status updates should in addition be given using secure email.

The estimate of the expected ***benefits*** *– given by experienced people at the municipality – is that the municipality will save 4 man-years per year when the system has been implemented. This will mainly be a consequence of the higher quality of the planning applications received and the need for fewer iterations before an application can be approved. In addition, it is expected that the users will save about 10 man-years each year through a simplified application process. Together with other quantitative benefits (summed over 10 years) the expected total benefit is expected to be around 120 million Norwegian Kroner. Non-quantitative benefits such as happier users and less illegal construction work are not included in the calculations.*

The estimate of the expected ***costs*** *– given by an external provider with relevant experience, is estimated to be around 65 million Norwegian Kroner.*

The above information is clearly not sufficient to say much about the realism of the benefits and costs estimates. Try, nevertheless, based on your own experience and other relevant knowledge with similar projects, to assess the uncertainty of the estimates of the benefits and of the costs.

The participants were randomly divided into two groups: traditional and alternative uncertainty assessment. These two groups had different instructions about the format of their uncertainty assessments.

Traditional Group (minimum–maximum values)

Uncertainty of the benefits estimate

Based on my experience with similar projects, I believe that the actual benefits (with 90% certainty) will be in the interval: _____ (minimum) and _____ (maximum) Norwegian Kroner.

Uncertainty of the costs estimate

Based on my experience with similar projects, I believe that the actual costs (with 90% certainty) will be in the interval: _____ (minimum) and _____ (maximum) Norwegian Kroner.

Alternative Group (looking back on previous benefits and costs estimation error)

Uncertainty of the benefits estimate

Input the proportion of similar (they do not have to be very similar), already completed, software projects for which you believe the benefits achieved were:

Less than half of the estimated benefits: _____% (Proportion of projects: 0%=none ... 100%=all)

Less than the estimated benefits: _____% (Proportion of projects: 0%=none ... 100%=all)

More than twice the estimated benefits: _____% (Proportion of projects: 0%=none ... 100%=all)

Uncertainty of the costs estimate

Input the proportion of similar (they do not have to be very similar), already completed, software projects for which you believe the actual costs were:

More than twice the estimated costs: _____% (Proportion of projects: 0%=none ... 100%=all)

More than the estimated costs: _____% (Proportion of projects: 0%=none ... 100%=all)

Less than half the estimated costs: _____% (Proportion of projects: 0%=none ... 100%=all)

The key difference between the two groups was that the first group used the traditional minimum–maximum uncertainty assessment method, with a given confidence level (here 90%), while the second were asked to assess the actual uncertainty (as indicated by their estimation error) of previous software projects. We suspected, as described in Section 1, that the traditional method would lead to the assessment of substantially less and more symmetric uncertainty than the uncertainty assessment method based on those looking back on the error, and implicitly the uncertainty, of similar, previously completed, software projects.

III. RESULTS

The assessments of the two groups differed greatly, both in terms of degree of uncertainty and in the amount of right- and left-skewedness of the implied uncertainty distributions. Tables I and , together with Figures 3–6 display key characteristics of the uncertainty assessments. For readability purposes, we translated the original uncertainty values into percentages of the estimates, i.e., in percent of the benefits estimate of 120 mill. Norwegian Kroner and of the costs estimate of 65 mill. Norwegian Kroner.

We have, for simplicity, assumed that a 90% confidence effort interval implies that the minimum is interpreted as the 5% level (p5), where it is only 5% likely that the actual value will be equal or less, and that the maximum is interpreted as the 95% level (p95), where it is 95% likely that the actual value will be equal or less. This is a common, although not necessary, interpretation of a 90% confidence effort prediction interval.

The *interval width* is measured as: *(p95-p5)/estimate*, where p5 and p95 are the values directly provided by the software professionals in the traditional method group, and the fitted values, assuming a PERT-distribution (using the tool @risk), for those in the alternative method group. The choice of a PERT-distribution to calculate the p5 and p95 for those in the alternative method group is based on the fact that this is a method frequently used in effort uncertainty assessment situations and that it enables us to compare the same pX-values for the two uncertainty assessment methods. The estimates are the same for the two groups, i.e., 120 mill. Norwegian Kroner for the benefits and 65 mill. Norwegian Kroner for the costs.

The *interval skew* is measured as: *distribution mean/estimate*, where the distribution mean is calculated using the PERT-formula: (Minimum + 4 x Estimate + Maximum). The minimum and maximum values are, as before, those provided by the software professionals in the traditional group and the fitted ones in the alternative uncertainty assessment group. The PERT-formula assumes that the Estimate is the mode (the most likely value). While this was not clear from the scenario description (Section 2), the intended interpretation of the benefits and costs estimate was not described, and it makes no large difference for the comparison of the two approaches. A skew-value larger than one, i.e., when the mean (expected value) of the distribution is higher than the estimate, indicates a right-skewed distribution, while a skew-value less than one indicates a left-skewed distribution. Notice that our measure of interval skew deviates from the traditional measure of distribution skew based on the difference between mode and mean.

TABLE I. TRADITIONAL UNCERTAINTY ASSESSMENT (MEDIAN VALUES)

Uncertainty assessment	***Benefits***	***Cost***
Minimum (p5)	67% of estimate	81% of estimate
Maximum (p95)	125% of estimate	154% of estimate
Interval width	0.54	0.69
Right-/left-skew	0.95 (weak left-skew)	1.08 (weak right-skew)

TABLE II. ALTERNATIVE UNCERTAINTY ASSESSMENT (MEDIAN VALUES)

Uncertainty assessment	*Benefits*	*Cost*
Probability of actual value less than half of estimate	30%	1%
Probability of actual value less than estimate	65%	25% (=100% - 75%)
Probability of actual value more than estimate	35% (=100% - 65%)	75%
Probability of actual value more than twice the estimate	5%	30%
Fitted minimum (p5)	22%	61%
Fitted maximum (p95)	200%	325%
Interval width	1.78	2.64
Right-/left-skew	0.89 (weak left-skew)	1.66 (strong right-skew)

Figures 3–6 display the benefits and costs uncertainty distributions of the groups. The uncertainty distributions are based on fitting the distribution to the three values p5, estimate (interpreted as the mode) and p95. The values are transformed so that: i) The value 1.0 is the estimated benefits in Figures 1 and 3, and the estimated costs in Figures 2 and 4, and ii) The values are in percentage of the estimate, e.g., the value 1.4 denotes a value 140% of the estimate. For each graph, the p5 (minimum) and p95 (maximum) values are indicated.

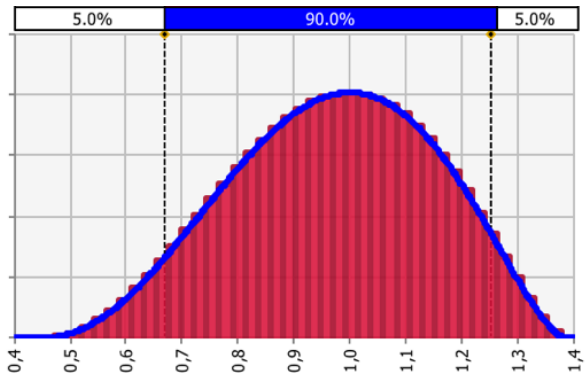


Fig. 3. Benefits distribution for traditional uncertainty assessment

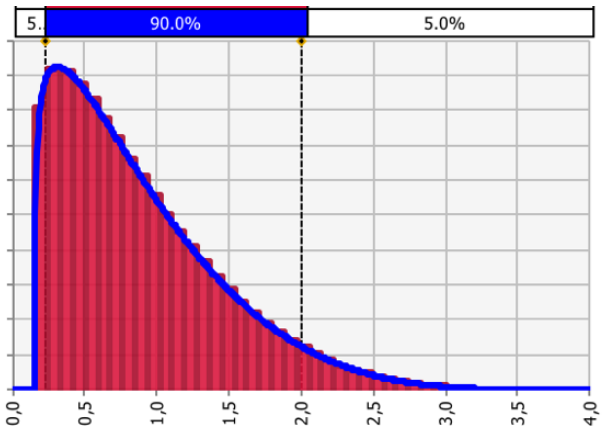


Fig. 4. Costs distribution for traditional uncertainty assessment

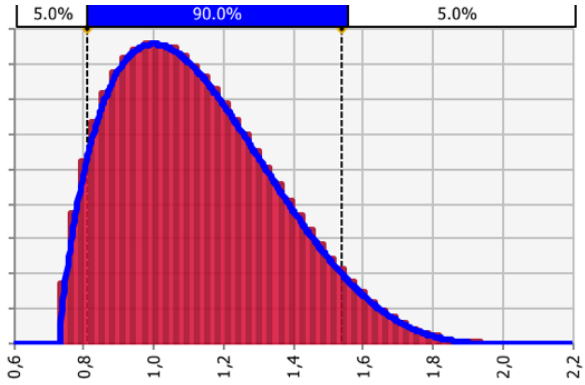


Fig. 5. Benefits distribution for alternative uncertainty assessment

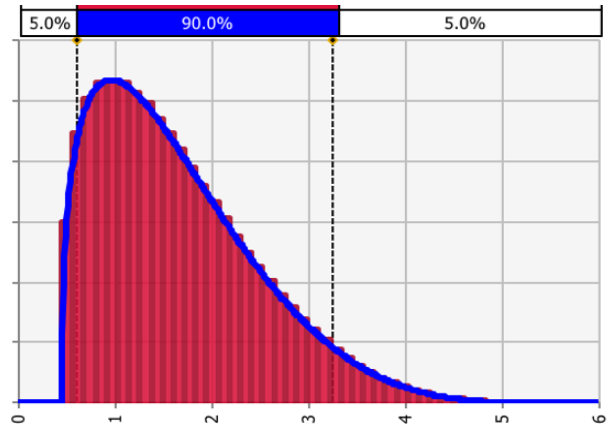


Fig. 6. Costs distribution for alternative uncertainty assessment

As can be seen from the values at the x-axes of the graphs and in the tables, there is a substantial difference in the assessed uncertainty between the traditional and the alternative method. Both H1, wider distributions, and H2, more left-skewed benefits distribution and, more right-skewed cost-distribution, when using the alternative uncertainty assessment method are consequently supported.

IV. DISCUSSION

A. *Which method led to the most realistic assessments?*

A key question is which of the uncertainty approaches led to the most realistic assessment. While, in this case, the answer to this would require that we knew the outcome of the (still on-going) project[1], we argue that there are at least two reasons to believe that the alternative approach gave the most realistic assessments:

- Looking back on previous experiences, sometimes called the use of "reference class" estimation or "analogy"-based estimation, when estimating software costs and benefits is documented to give more realism [4, 7]. While this has mainly been documented for cost estimates, we find it reasonable to assume that a

[1] In fact, we would not know the realism of the uncertainty assessments even if we knew the outcome of the project. In order to know the realism of the uncertainty assessments we would need many uncertainty assessments and actual outcomes, and to compare the confidence level or probability with the actual hit rate. After all, being 90% confident means that one will be wrong in 10% of the cases.

similar realism improvement will be present in uncertainty assessment contexts.

- Empirical data suggest that the uncertainty of a project of the type used as the case in this study is high. Software development projects in Norway were, for example, found to have an average costs overrun of 67% for projects with a public client [15]. Other surveys, for example [1], find that costs overrun distributions are strongly right-skewed, with 41% of data management projects having a costs overrun of more than 25%, and many of them 2-3 times more.

A further argument in favour of the alternative uncertainty assessment method is that the assessments were based on the respondents' actual experiences about typical estimation error and bias, e.g. how typical over-estimating the benefits and under-estimating the costs of similar projects were. The respondents were randomly divided into groups, which implies that the group using the traditional method probably is likely to have had, as a group, about the same experience regarding costs and benefit estimation error. In other words, those using the traditional method assessed the uncertainty to be much lower than what they had probably experienced in similar projects prior to this one. As far as we can judge, there was nothing in the project description that indicates a substantially lower complexity or risk of this project compared to other governmental projects of similar size and type.

B. Implications for benefits-to-costs ratio (test of H3)

An interesting implication of our results is that the benefits-to-costs ratio (return on investment) analysis including uncertainty would give very different values for the two approaches.

As a benchmark value, we start with the *non-stochastic (statistically naïve) benefits-to-cost analysis*, i.e., without taking uncertainty into consideration. Here we use the estimated benefits and costs and get a return on investment of 120 mill. Norwegian Kroner / 65 mill. Norwegian Kroner = 1.85, i.e., the expected benefits-to-costs ratio is strongly positive. Very often, as far as we have experienced, this non-stochastic value is the one used when making decisions about whether to start a software project or not.

Then we use at the uncertainty assessment of those in the traditional group, using the benefit and costs distributions based on the median assessments of p5 and p95, the PERT-distribution and a Monte Carlo simulation to simulate the ratio of benefits to costs (10,000 simulations). We then get an expected return of investment of 1.6 (see Figure 7), i.e., the expected benefits-to-costs ratio is still strongly positive, although slightly less than with the non-stochastic calculations.

Finally, we use the uncertainty assessment of those in the alternative group, using the median probability assessments (PERT-fitted p5 and p95) and simulate the benefits-to-costs ratio using Monte Carlo simulation (10,000 simulations). Now the expected benefits-to-costs ratio is as low as 1.2, see Fig8. In this case, the probability of making no profit at all is as high as 40%. In other words, using the alternative, arguably more realistic, uncertainty analysis makes it much less obvious that the project is worth starting.

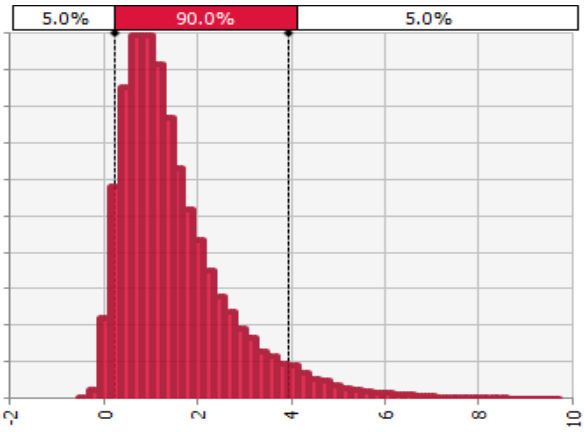


Fig. 7. Benefit / Costs – traditional uncertainty assessment

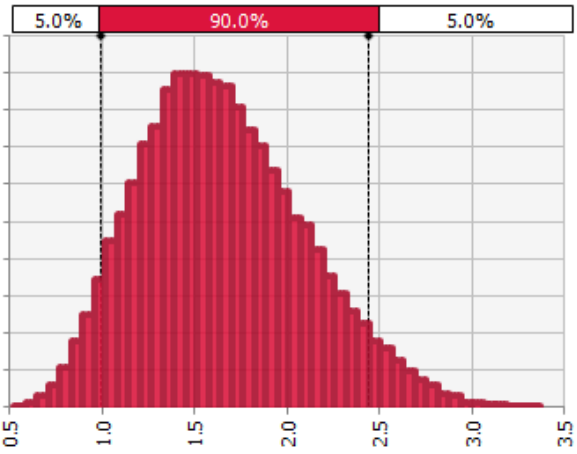


Fig. 8. Benefits / Costs – alternative uncertainty assessment

C. Limitations

There are several limitations to take into consideration when interpreting and using the results reported in this experiment. While the results are consistent with previous results on the traditional uncertainty assessment approach, i.e., that it leads to too narrow and symmetric intervals, we cannot exclude that those using the traditional intervals had the most accurate assessment of the underlying uncertainty. This can only be assessed when aggregating assessments and outcomes over many projects. What we can be confident about, however, is that those using the traditional uncertainty assessment were much more optimistic about the uncertainty than would be warranted by similar projects. We interpret this as a high likelihood of those in the traditional group being over-optimistic about the uncertainty.

The generalizability of the results to other project contexts and other software professionals is to a large extent unknown, as neither the project nor the participants were selected to represent a particular population. When looking at the roles, experience level and organizations of the participants (using the list of participants of the seminar), however, we see that they represent relevant roles. They were, with a few exceptions, software managers on the client side or project managers on the

provider side. The fact that they spent time visiting a seminar on benefits management, and had previous experience in estimating benefits and costs, indicates that they may have been more than averagely interested and, perhaps, more than average skilled in this topic.

We have assumed an underlying PERT-distribution for our analyses. The uncertainty values and results are affected by this choice. We evaluated the use of log-normal and gamma distribution, which gave similar results, i.e., there is little reason to believe that the choice of underlying uncertainty distributions had a large impact on the result.

V. Conclusions

Software professionals asked to give benefits and costs uncertainty assessments based on the estimation error they recalled having experienced on similar software projects (termed the alternative method) gave wider uncertainty intervals than those using the traditional minimum–maximum 90% confidence intervals. It also led to more left-skewed benefits distributions and more right-skewed costs distributions. The difference in assessment of benefits and costs uncertainty led to a substantial difference in the assessment of the profitability of the project, i.e., the benefits-to-costs ratios were highly favorable using the traditional method while much less so for the alternative method.

Assuming that the recalled projects were similar in terms of uncertainty to the one to be assessed, the alternative method is, we argue, likely to have led to more realistic uncertainty assessment. Previous empirical results on the use of reference-class and analogy-based, i.e., looking-back based, estimation models for software development effort, support the suggestion that looking back on previous experience-based methods leads to more realistic judgments.

We plan to conduct more studies comparing the traditional and different variants of the alternative uncertainty assessment method, where we will vary the elicitation format and, preferably, compare with the actual benefits achieved and costs spent.

References

[1] Budzier, A. and B. Flyvbjerg, *Overspend? Late? Failure? What the data say about IT project risk in the public sector.* Commonwealth Governance Handbook, 2012. **13**: p. 145-157.

[2] Connolly, T. and D. Dean, *Decomposed versus holistic estimates of effort required for software writing tasks.* Management Science, 1997. **43**(7): p. 1029-1045.

[3] Cook, J.D., *Determining distribution parameters from quantiles*. 2010, UT MD Anderson Cancer Center Department of Biostatistics Working Paper Series.

[4] Flyvbjerg, B., *Curbing optimism bias and strategic misrepresentation in planning: Reference class forecasting in practice.* European Planning Studies, 2008. **16**(1): p. 3-21.

[5] Gruschke, T.M. and M. Jorgensen, *The role of outcome feedback in improving the uncertainty assessment of software development effort estimates.* Acm Transactions on Software Engineering and Methodology, 2008. **17**(4).

[6] Jørgensen, M., *Realism in assessment of effort estimation uncertainty: It matters how you ask.* IEEE Transactions on Software Engineering, 2004. **30**(4): p. 209-217.

[7] Jørgensen, M., *Top-down and bottom-up expert estimation of software development effort.* Information and Software Technology, 2004. **46**(1): p. 3-16.

[8] Jørgensen, M., *The Ignorance of Confidence Levels in Minimum-Maximum Software Development Effort Intervals.* Lecture Notes on Software Engineering, 2014. **2**(4).

[9] Jørgensen, M. and D.I.K. Sjøberg, *An effort prediction interval approach based on the empirical distribution of previous estimation accuracy.* Information and Software Technology, 2003. **45**(3): p. 123-136.

[10] Jørgensen, M. and K.H. Teigen. *Uncertainty Intervals versus Interval Uncertainty: An Alternative Method for Eliciting Effort Prediction Intervals in Software Development Projects*. in *International Conference on Project Management (ProMAC)*. 2002. Singapore.

[11] Jørgensen, M., K.H. Teigen, and K. Moløkken, *Better sure than safe? Over-confidence in judgement based software development effort prediction intervals.* Journal of Systems and Software, 2004. **70**(1-2): p. 79-93.

[12] Kerzner, H., *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. 2003: John Wiley & Sons.

[13] Little, T., *Schedule estimation and uncertainty surrounding the cone of uncertainty.* Software, IEEE, 2006. **23**(3): p. 48-54.

[14] McKenzie, C.R.M., M. Liersch, and I. Yaniv, *Overconfidence in interval estimates: What does expertise buy you?* Organizational Behavior and Human Decision Processes, 2008. **107**: p. 179-191.

[15] Moløkken, K., M. Jørgensen, S.S. Tanilkan, H. Gallis, A.C. Lien, and S.E. Hove, *Project Estimation in the Norwegian Software Industry-A Summary*. . 2004: Simula Research Laboratory, 3.

[16] PMI, *Guide to the Project Management Body of Knowledge (PMBOK(r) Guide)-Sixth Edition*. 2017.

[17] Winman, A., P. Hanson, and P. Jusling, *Subjective probability intervals: how to reduce overconfidence by interval evaluation.* Journal of experimental psychology: learning, memory and cognition, 2004. **30**(6): p. 1167-1175.

[18] Yaniv, I. and D.P. Foster, *Precision and accuracy of judgmental estimation.* Journal of Behavioral Decision Making, 1997. **10**(1): p. 21-32.

Relations between Project Size, Agile Practices and Successful Software Development

Magne Jørgensen

Simula Metropolitan, Oslo, Norway

ABSTRACT. The use of agile methods in the execution of large-scale software development is increasing. To find out more about the effect of this on project performance, information was collected about 196 Norwegian IT-projects. Increased project size was associated with decreased project performance for both agile and non-agile projects, but the projects using agile methods had better performance than the non-agile projects for all examined project size categories. Flexible scope, frequent deliveries to production, a high degree of requirement changes and more competent providers are candidates to explain the better performance of agile projects.

Keywords: agile methods, project performance, project characteristics

A traditional response to increased size and complexity of work is to implement more planning and management formalism [1]. Agile software development methods, on the other hand, try to remove or reduce much of the traditional project management formalism. Does this mean that agile, as indicated in [3] mainly work for smaller projects? Or do agile methods work well for larger projects as suggested in [2, 5]. The available empirical evidence is mixed and does not allow strong claims. In addition, the evidence does not give much insight into when, if at all, agile methods tend to work well for large projects. This shortage of empirical evidence motivated the survey reported in this paper, aiming at answering the following two questions:

1) How well do larger agile software projects perform compared to smaller projects and non-agile projects?

2) Which agile practices and characteristics are connected with better performance?

The Survey

Respondents and data collection

The survey participants were Norwegian software professionals visiting three different seminars on project management in 2016 and 2017. The software professionals provided information about their *last* completed projects. 216 responses were received. After removing responses without the minimum information needed for the analysis, i.e. the budget size category, the development method and the perceived performance of the project, there were 196 unique responses remaining. The project information was given anonymously, in Norwegian, using the survey tool Qualtrics. There was a "don't know" option for all project information items to ensure that the respondents only answered when they felt they had sufficient knowledge.

The software professionals had on average 13 years of experience, with 70% having 8 or more years. 69% of the respondents were from the provider side and 31% were from the client side. 71% had technical roles in the reported project, e.g. architects or developers, and 29% had managerial roles, e.g. product owners, team leaders and project managers.

Project characteristics

The project characteristics requested from the participants is described in Table 1. The included variables are those that were found to distinguish between successful and failed software projects in an earlier survey [4]. To avoid too few observations for some categories the analyzed category "high" ("low") includes both "very high" ("very low") and "high" ("low") responses.

Table 1. Project characteristics*

Characteristic	Categories
Budget size (used as measure of project size)[1]	Small (<1 mill Euro) Medium (1-10 mill Euro) Large (>10 mill Euro)
Development method[2]	Agile Non-agile
Requirement volatility[3]	High (>30% changes) Low (<=30% changes)
Perceived flexibility of scope	High Low
Perceived detail of upfront project plan	High Low
Perceived detail of upfront requirement specification	High Low
Frequency of deliveries to production[4]	>4 per year <= 4 per year

Contract type	Time & materials Fixed price
Perceived provider competence	High Low
Perceived client competence	High Low

*: The full questionnaire is available to interested readers upon request.
1: The budget size categories small, medium and large are the same as those found to separate the effect of agile practices in [5].
2: There is no commonly accepted definition of what it means to work agile. I used the respondents' own perception of whether they worked agile or not in the first analysis and added analyses of the effect of different agile practices and characteristics in the second analysis.
3: The threshold of 30% is based on what was closest to the median level of perceived amount of requirement change of the projects.
4: The original categories were "none", "1-4" and "more than 4", where the two first were joined. Notice that even non-agile projects, e.g., incremental or timeboxing-based projects, may have deliveries to production during the project execution.

Project performance

After describing characteristics of the project, each participant assessed the performance of their last completed project, as he/she perceived it, using the scale: *very successful – successful – acceptable – problematic – very problematic* for each of the success dimensions: client benefits (value), cost control, time control, productivity and technical quality.

To define the project's overall performance, we used the following categorization:

Successful:	Successful or better on all five success dimensions,
Acceptable:	Acceptable or better on all five success dimensions
Failed:	Very problematic on at least one success dimension.

Data collection challenges

Different participants may be involved in the same projects, leading to the possibility of duplicate projects in our data set. The variance in organizations of the participants, as analyzed from the list of seminar participants and the typically large size of their organizations, indicates that the number of duplicates, if any, is very low.

Participants from the client and the provider side, as well as participants in different roles, may have different knowledge and perceptions of a project's performance. While this subjectivity may affect the accuracy of the reported success and failure rates, it is less likely to change the direction of the connection between development methods, project size and project performance.

An examination of the list of participants shows that the majority of them belong to or worked for large organizations with mainly administrative software applications. Consequently, the results may mainly be valid within this context.

Results

In total, 16% of the software projects were categorized as successful, 52% as acceptable and 7% as failed. The small and medium sized projects had the best performance with 15% and 22% categorized as successful, 55% and 50% as acceptable, and 7% and 4% as failed, respectively. The larger projects had 5% categorized as successful, 41% as acceptable and 14% as failed. The decrease in project performance with increased project size corresponds to findings in other studies, e.g., [6].

Seventy-four percent of the projects were categorized as agile. These projects, see Table 2, had better average success rate than the non-agile projects for all three size categories. Figure 1 displays this interaction effect for projects with acceptable project performance. An analysis using a general linear model (GLM) with the variable development method (agile and non-agile) nested into the variable budget size (small, medium and large) gives a that the difference in proportion acceptable projects is statistically significant, one-sided tests, for small ($p<0.01$) and medium ($p=0.03$) sized projects, but not, due to the lower number of observations, for large sized projects ($p=0.12$).

Table 2. Relationship between budget size category, development method and project performance*

Project performance	Development method	Small (n=120)	Medium (n=54)	Large (n=22)
Successful (n=31)	Agile	19%	24%	7%
	Non-agile	0%	19%	0%
Acceptable (n=102)	Agile	65%	58%	50%
	Non-agile	19%	31%	25%
Failed (n=13)	Agile	2%	3%	14%
	Non-agile	23%	6%	13%

* The percentages are the proportion of successful, acceptable and failed projects for projects same budget size category and same development method.

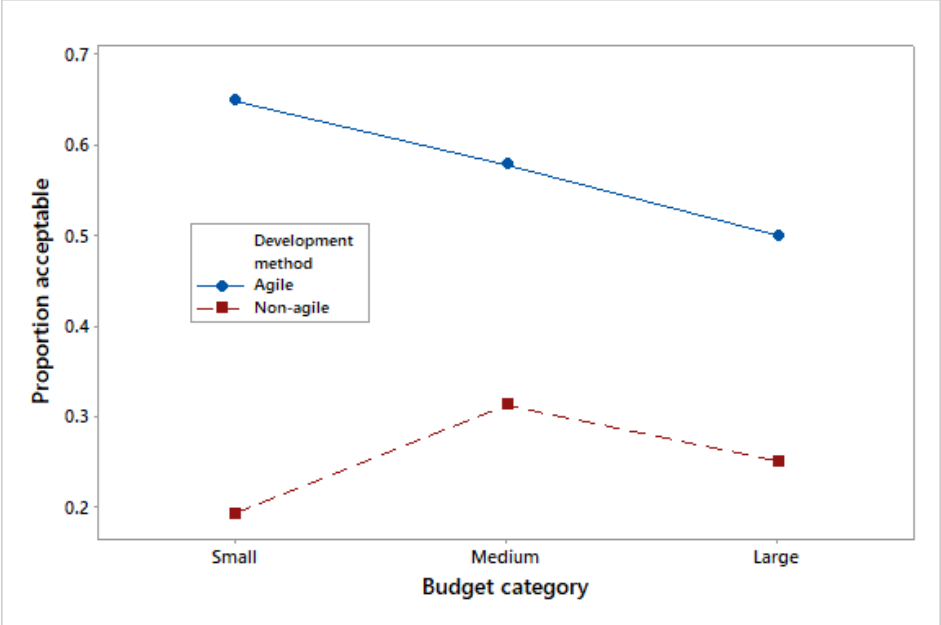


Figure 1. Interaction plot of projects with acceptable performance

The analysis of practices and context characteristics (factors) potentially connected with better performance of agile projects was completed as follows. First, the factors more frequently observed in agile than in non-agile projects were identified through a chi-square analysis. Due to more frequent use these factors may explain the better performance of agile projects even if they have a similar, positive effect on non-agile projects. Second, the connection between all factors and acceptable project performance (the largest performance category) was analyzed.

The factors associated with a statistically significant (here set as $p<0.05$) higher proportion of agile projects were *high requirement volatility* (50% of agile projects and 33% of non-agile projects had more than 30% requirement changes, $p=0.04$), *frequent deliveries to production* (68% of agile projects and 32% of non-agile projects had more than four deliveries to production per year, $p<0.01$) and *flexible scope* (79% of agile projects and 47% of non-agile projects had a perceived high degree of scope flexibility). There were no statistically significant differences in proportion of projects with *detail of project plan* (60% of agile and 53% of non-agile projects were perceived to have little detail in project plans, $p=0.75$), *detail of requirement specification* (55% of agile and 54% of non-agile projects were perceived to have little detail in requirement specification, $p=0.96$), and *contract type* (51% of agile and 58% of non-agile used fixed price contracts, $p=0.53$).

Table 3 displays the results for the proportion of projects with acceptable performance for the analyzed factors. Notice that the sum of observations is lower than the full dataset of 196 projects due to “don’t know” answers.

Table 3. Proportion projects with acceptable performance*

Factor	Category	Agile (n=146)	Non-agile (n=50)
Requirement volatility	High (n=80)	58%	13%
	Low (n=97)	61%	29%
Delivery frequency	>4 per year (n=99)	70%	25%
	<=4 per year (n=60)	49%	21%
Scope flexibility	High (n=71)	85%	33%
	Low (n=26)	50%	40%
Detail of project plan	High (n=59)	67%	18%
	Low (n=81)	53%	21%
Detail of req. spec.	High (n=63)	55%	13%
	Low (n=76)	61%	26%
Contract type	Fixed price (n=66)	60%	17%
	Time & materials (n=60)	60%	23%

* The percentages are the proportion of acceptable projects for projects with same factor category and same development method. There are too few observations (low statistical power) for some of the combinations of categories to conduct meaningful tests of statistical significance for the interactions in Table 3. The differences should consequently be interpreted as indications of relationships, not as strong evidence.

The results in Table 3 suggest that experiencing high requirement volatility did not greatly affect the proportion of acceptable agile projects, while the proportion of acceptable non-agile projects decreased from 29% to 13%. Frequent delivery to production seems to have had a much stronger positive connection with better performance for agile than for non-agile projects. This practice was also more common among agile projects and may therefore contribute to a better performance of agile projects both by being more frequently used and by having a stronger positive connection. Higher scope flexibility was connected with much higher proportion of acceptable performance for agile projects, and a lower proportion for non-agile projects. The factors including detail of project plan, detail of requirement specification and contract type did not contribute much to explaining an improved performance of agile projects.

Table 4 suggests that as the project size increased from small to medium/large a high degree of requirement changes further increased the superior performance of the agile projects. A higher delivery frequency was associated with larger increase in acceptable agile than in acceptable non-agile projects. Similarly, higher flexibility of scope was associated with increased performance of small agile and decreased performance of small non-agile projects.

Table 4. Proportion projects with acceptable performance, per size category*

Factor	Category	Agile		Non-agile	
		Small (n=94)	Medium/large (n=52)	Small (n=26)	Medium/large (n=24)
Requirement volatility	High (n=80)	62%	54%	13%	14%
	Low (n=97)	65%	47%	20%	38%
Delivery frequency	High (n=99)	73%	65%	-	38%
	Low (n=60)	54%	41%	13%	27%
Scope flexibility	High (n=71)	86%	84%	14%	-
	Low (n=26)	55%	40%	57%	-

* The percentages are the proportion of acceptable projects for projects with same factor category, budget size category and development method. There are too few observations (low statistical power) in some of the categories to conduct meaningful tests of statistical significance for the interactions in Table 4. The differences should consequently be interpreted as indications of relationships, not as strong evidence. The fields with "-" have fewer than five observations, due to missing data about a project or few occurrences, and the proportions were not calculated.

If agile projects attract more competent providers or clients, this may contribute to the difference between agile and non-agile projects. An analysis of the project data demonstrated that that the agile software projects were indeed perceived to have more competent clients and providers (Chi-square test of independence gives p=0.02 and p=0.01, respectively). A binary logistic regression model with the elements client competence (high vs. low), provider competence (high vs low), development method (agile vs non-agile), requirement volatility (high vs. low), delivery frequency (high vs. low) and scope flexibility (high vs low), using the performance measure acceptable (1=acceptable, 0=not acceptable) as the dependent variable gives a 5.7 and 2.4 times higher likelihood of observing an acceptable project when having a high compared to a low or medium competent provider (p=0.046) and client (p=0.27, not statistically significant), respectively. More studies are needed to analyse how client and provider competence interact with agile practices and contexts to explain differences in project performance.

Conclusions

The survey of 196 Norwegian software projects provides empirical support for the use of agile methods on larger as well as smaller software projects, especially when including flexible scope and frequent delivery to production, and in contexts with high requirement changes. A contributing factor seems to be that agile projects tend to have more competent providers.

References

[1] Child, J., *Predicting and understanding organization structure.* Administrative Science Quarterly, 1973. **18**(2): p. 168-185.

[2] Dingsøyr, T., N.B. Moe, T.E. Fægri, and E.A. Seim, *Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation.* Empirical Software Engineering, 2018. **23**(1): p. 1-31.

[3] Dybå, T. and T. Dingsøyr, *Empirical studies of agile software development: A systematic review.* Information and software technology, 2008. **50**(9): p. 833-859.

[4] Jørgensen, M., *A survey on the characteristics of projects with success in delivering client benefits.* Information and Software Technology, 2016. **78**: p. 83-94.

[5] Jørgensen, M. *Do Agile Methods Work for Large Software Projects?* in *International Conference on Agile Software Development*. 2018. Porto, Portugal: Springer.

[6] Sauer, C., A. Gemino, and B.H. Reich, *The impact of size and volatility on IT project performance.* Communications of the ACM, 2007. **50**(11): p. 79-84.

Biography: Magne Jørgensen is professor in software engineering at the University of Oslo and chief research scientist at Simula Metropolitan. He has extensive industry experience as consultant and manager and currently serves at the Norwegian digitalisation board where he advises governmental software projects. His research interests include project management, evidence-based software engineering and expert judgment. His recent book on effort estimation can be downloaded for free from: tinyurl.com/timepredictions.

VEDLEGG 2

When is agile better?

How the use of agile and autonomous teams affect success differently in different contexts (and other results)

We analysed the connections between software project outcome and the following factors:

- Development method
- Contract type
- Sourcing strategy
- Requirement volatility
- Project size
- Benefits management
- And a little bit about the use of autonomous teams

Philosophy: Success and failure patterns, not factors

Studies

- Four surveys with participants on IT management seminars
  - Asked to give information about their last, completed (or cancelled) project
  - 60-150 participants in each
  - From both client and provider side and many roles
- An interview-based study of 32 governmental software development projects
- Project data from an offshoring marketplace
  - More than 400.000 projects/tasks
  - Most of them very small

In spite of all the challenges, our empirical results may have value.

Weak evidence, as long as it is not misleading, is often better than no evidence.

SINCE THIS WORKSHOP IS ABOUT AUTONOMOUS TEAMS, LETS START WITH THAT …

”THE TEAM HAS SUBSTANTIAL FREEDOM IN SELECTING, SCHEDULING, PROCESSING AND/OR COMPLETING TASKS”

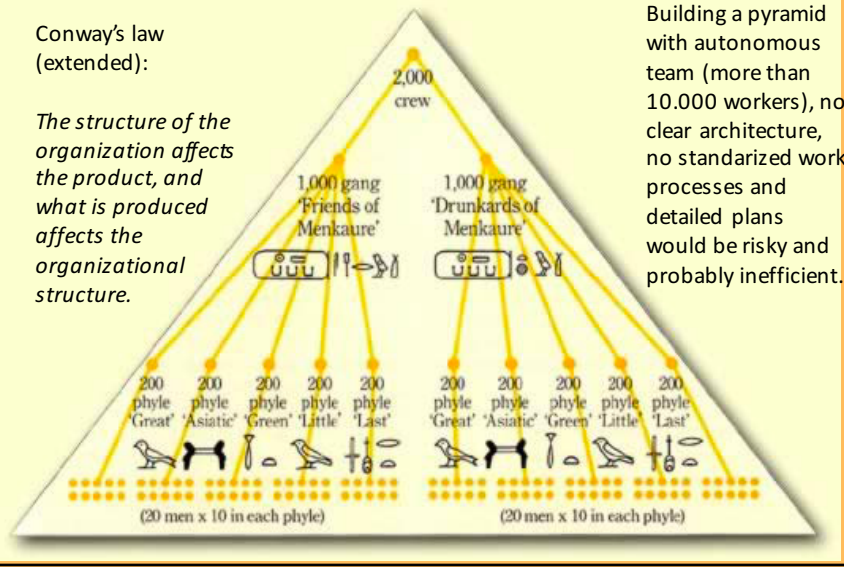
Autonomous teams are useful for many types of tasks, and is not a new way of collaborative effort

It may not fit all types of tasks.

Here: The organization of pyramid construction (Giza)

Conway's law (extended):

The structure of the organization affects the product, and what is produced affects the organizational structure.



Building a pyramid with autonomous team (more than 10.000 workers), no clear architecture, no standarized work processes and detailed plans would be risky and probably inefficient.

IS SOFTWARE DEVELOPMENT MORE LIKE GAME HUNTING IN TEAMS OR PYRAMID CONSTRUCTION?

(DOES SOFTWARE DEVELOPMENT USUALLY BENEFIT FROM THE USE OF AUTONOMOUS TEAMS?)

I GUESS YOU ALREADY THINK YOU KNOW THE ANSWER ON THIS, BUT LET'S GET EMPIRICAL. NEVER TRUST CLAIMS WITHOUT EMPIRICAL DATA.

Does it for example end up with (autonomous) teams fighting each others (as in a rugby scrum)

Survey design … (unpublished)

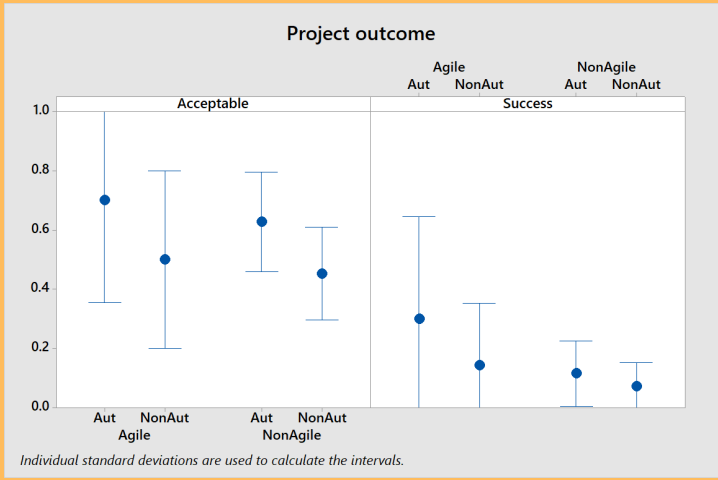
- Survey of 101 software projects (their last project, both provider and client respondents)
- ”Do you consider the development team(s) of the project to have been ”self-organized”?
  - Yes, no, don’t know (don’t know answers removed from analysis)
- 45% reported that the team(s) were self-organized
  - The question forces a dichotomy and is a subjective assessment.
  - Assumes that ”self-organized” is close to what people will think of as autonomous.
- The providers reported much higher proportion of self-organized teams than the clients (73 vs 23%).
  - Indicates a differences in use of terminology, lack of knowledge or something else …

Here is what we found ...

- Self-organized teams (average values)
  - Were more frequently used for **smaller projects** (2.3 vs. 3.0, using a scale from 1 to 4, where 2 = Small (0.1-1 mill Euro) and 3 = medium (1-10 mill Euro)
  - Were assessed to be slightly **more agile** (2.5 vs 2.8, using a scale from 1=very agile to 5=not at all agile) and used more agile practises (3 vs. 1)
    - More use of product backlog (71 vs 61%), velocity (40 vs 11%), stand up meetings (69 vs 29%), but same degree of frequent deliveries (2.2 vs 2.2, on a scale from 1=frequent deliveries to client and 4=only end-deliveries)
  - Had a slightly **less involved client** (1.9 vs 1.8, using a scale from 1="very involved" to 4 ("not much involved").
  - Were **less** likely to have a **detailed, upfront project plan** (40% vs 60%).
  - Had about the **same requirement volatility** (1.9 vs 2.0, where 1=very much and 4=very few/none) and **similar use of contracts** (only slightly less use of fixed price contracts).

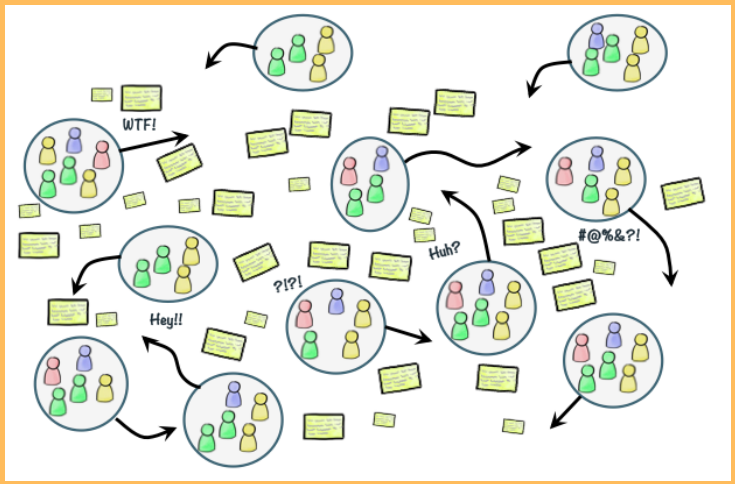
More importantly, did self-organized (autonomous) teams deliver better results?

Yes! Especially when working agile with frequent deliveries to client

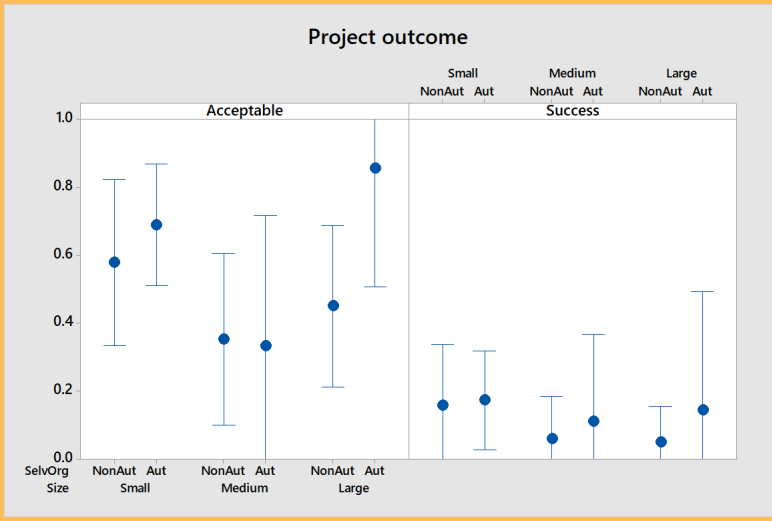


Agile = Perceived as "very agile"/"agile" and with freq. deliveries during the project execution.
Acceptable = Perceived as acceptable or better wrt client benefits, time control and cost control
Successful = Perceived as successful wrt client benefits, time control and cost control

What about scaling? Does autonomous teams on large projects lead to chaos?



It seems to scale well ...



Small =< 1 mill Euro, Medium = 1-10 mill Euro, Large = > 10 mill Euro

Selected results (related to agile) from our surveys

A survey of 63 Norwegian software development projects

ELSEVIER

Information and Software Technology

Volume 78, October 2016, Pages 83-94

A survey on the characteristics of projects with success in delivering client benefits

Magne Jørgensen

Show more

https://doi.org/10.1016/j.infsof.2016.05.008

Get rights and content

When looking at agile projects we found that "agile is not agile"

The numbers show the increase (in percent points) in proportion of successful projects

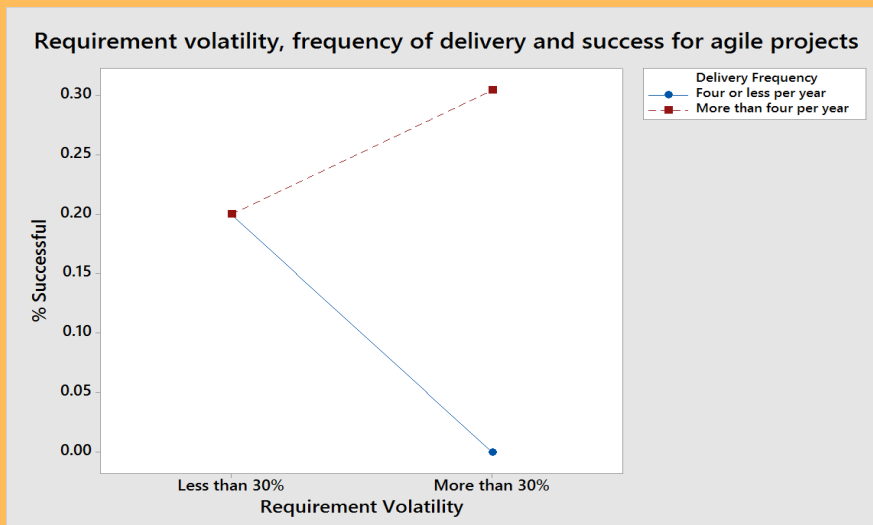
	Agile	Frequent delivery to production	Flexible scope
Client benefits	16%	22%	29%
Technical quality	21%	6%	32%
Budget control	2%	22%	29%
Time control	8%	11%	24%
Efficiency	11%	5%	24%

Agile was only connected with more client benefits when including frequent delivery to production and flexible scope.

Agile projects not including these practices were LESS successful than non-agile projects!

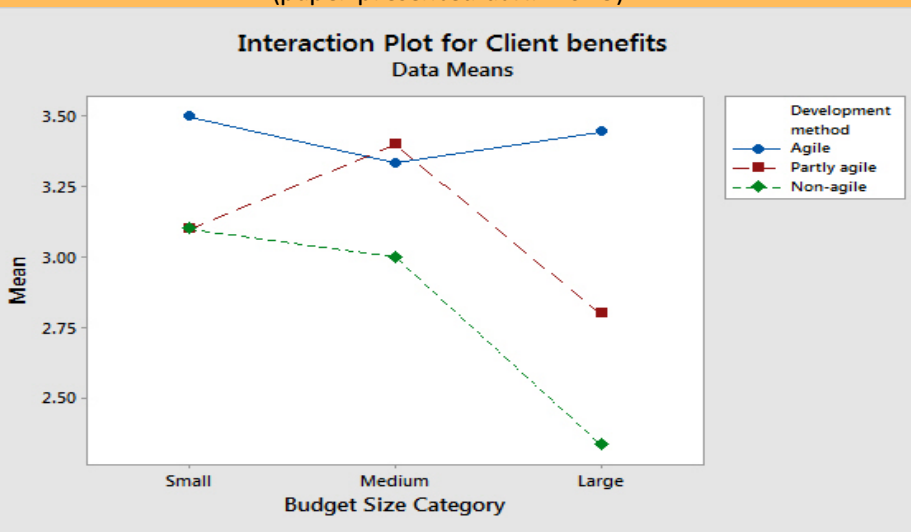
Similar results in our later follow-up surveys and studies

Similarly, in a more recent study (unpublished), we found that the presence of frequent deliveries in agile projects was mainly important when connected with high requirement volatility



Agile software projects seem to be less affected by large project size

(paper presented at XP 2018)



Analysis of data about more than 400.000 small projects (offshoring marketplace) and an in-depth survey of 35 large governmental projects

ELSEVIER

International Journal of Project Management

Volume 35, Issue 8, November 2017, Pages 1573-1586

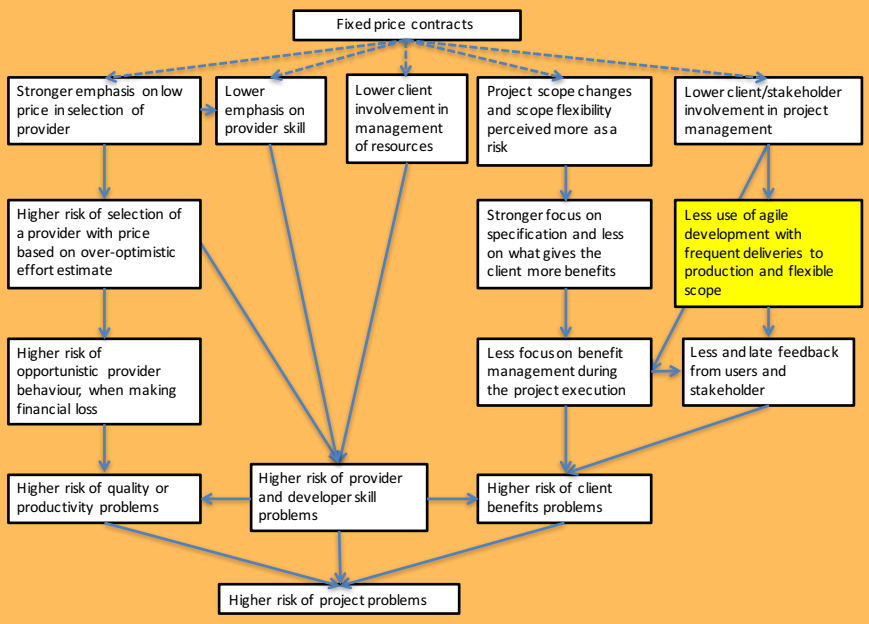
Direct and indirect connections between type of contract and software project outcome

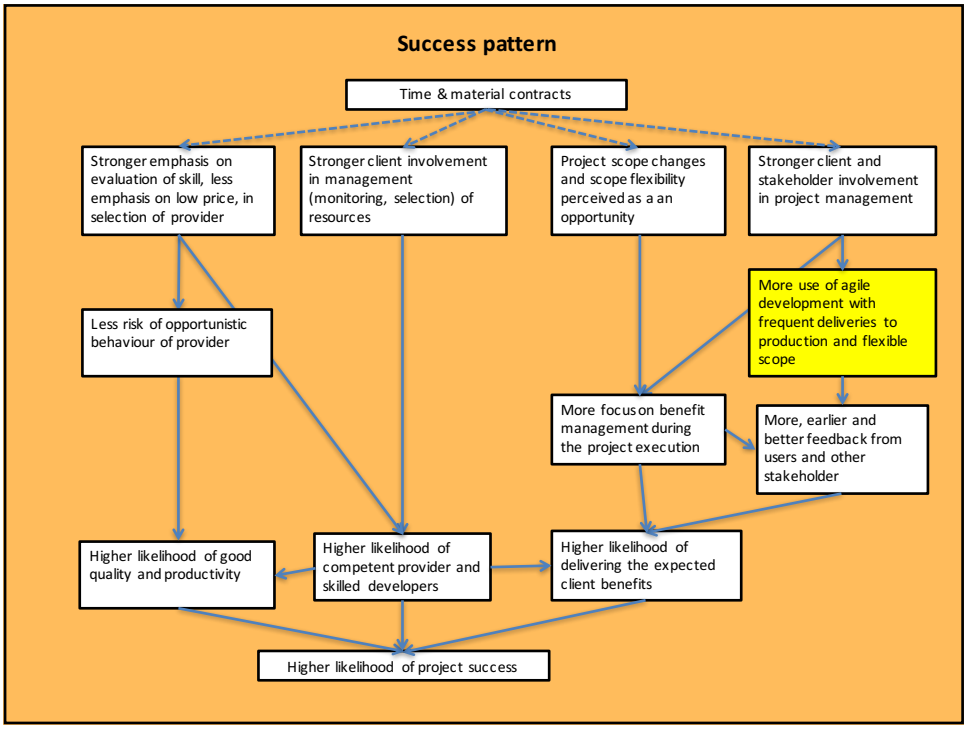
Magne Jørgensen, Parastoo Mohagheghi, Stein Grimstad

Show more

https://doi.org/10.1016/j.ijproman.2017.09.003

Failure pattern starting with the choice of contract



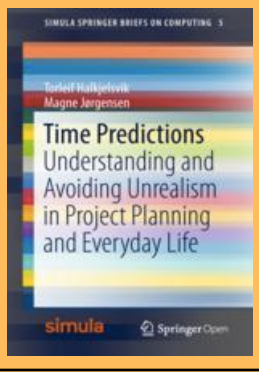


What I wanted to say ...

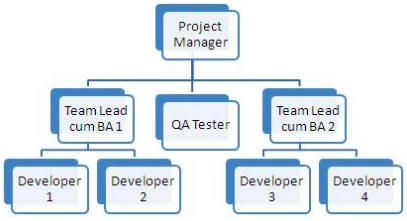
- The evidence (although not very strong) suggests that autonomous teams are more successful.
  - The causal connections may be complex
- Agile is not agile, and especially «frequent deliveries to production» (enabling feedback) and »flexible scope» is connected with more success.
  - This is especially the case when there is a high requirement volatility (which to some extent is caused by the feedback) and when projects get larger.
- It is when we analyse success and failure patterns, not factors, that we get the most useful results and the best insight.

QUESTIONS?

Download my newest book for free: tinyurl.com/timepredictions



Pyramid construction organizational thinking common in many workplaces



Analysis challenges:

- Poorly defined concepts, e.g., what is agile and what is an autonomous team?
- Forcing dichotomies on continuous scales
- Cause-effect vs correlation
- Subjectivity in measurement
- Little control of sample representativeness (convenience samples, mainly from Norway)
- Missing context information

Based on:

- Jørgensen, M. (2016). A survey on the characteristics of projects with success in delivering client benefits. *Information and Software Technology*, *78*, 83-94.
- Jørgensen, M., Mohagheghi, P., & Grimstad, S. (2017). Direct and indirect connections between type of contract and software project outcome. *International Journal of Project Management*, *35*(8), 1573-1586.
- Jørgensen, M. (2017, May). Software development contracts: the impact of the provider's risk of financial loss on project success. In *Proceedings of the 10th International Workshop on Cooperative and Human Aspects of Software Engineering* (pp. 30-35). IEEE Press.
- Do Agile Methods Work for Large Software Projects? (2018, April) To be presented at *XP 2018*, Porto, Portugal.
- Huge investments in digitalization. What does it give us in return? Keynote Software 2018 (DnD's annual conference, Oslo, Norway).

What are key characteristics of software projects (digitalization projects)?

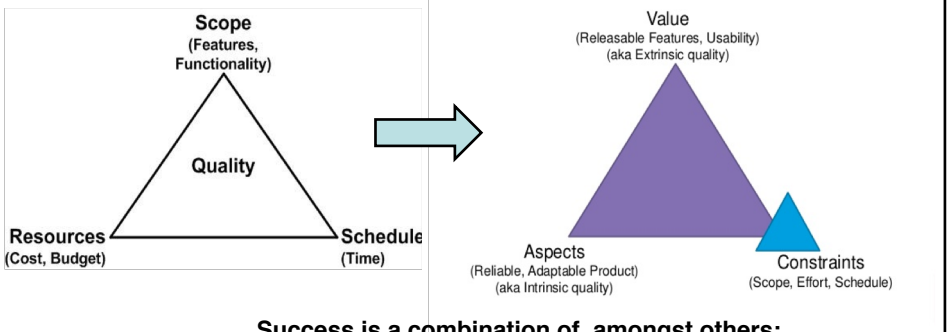
- Exposed to a **fast-changing world** (technology, needs, opportunities)
- Producing **innovations** (never constructing the same twice)
- **Transformation** projects (change of work processes)
- Enables **agility** (such as scope flexibility, less upfront planning and specification work, frequent deliveries, late changes)
- **Continuous development** (the organization of software development work as a project is by more and more software professionals believed to problematic.)

What is benefits management in software development?

- A set of processes, optimally including:
  - Identify and estimate benefits (and costs)
  - Develop a plan for when and how to realize benefits
  - Allocate responsible for the realization of the benefits
  - Continuous delivery, prioritization and management of benefits during the project execution
  - Evaluation of realized benefits
- Large variation in how (and if) these steps are implemented

What does it mean to succeed and to fail with software development?

Software project success



Success is a combination of, amongst others:

- Client benefits delivered
- Cost control
- Time control
- Development efficiency
- Software properties (technical quality)
- Learning

Our studies on benefits management:

- Nine surveys, with 50-200 participants each, representing around 1000 Norwegian software projects in the public and the private sector.
- In-depth, interview-based examination (case studies) of 35 software projects in the public sector of Norway
- Ongoing studies in two large organization on benefits management in large scale agile

Success and failure rates found in our studies

All studies give similar results:

- Around 50-60% successful projects
- Around 30-40% problematic (but not failed) projects
- Around 10% failed projects

How is agile and benefits management connected?

It helps to work agile, but …

	Agile	Frequent delivery to production	Flexible scope
Client benefits	16%	22%	29%
Technical quality	21%	6%	32%
Budget control	2%	22%	29%
Time control	8%	11%	24%
Efficiency	11%	5%	24%

… only when including frequent delivery to production and flexible scope.
Agile projects not including these two practices were LESS successful than non-agile projects! These two practices are strongly connected to benefits management.

Similar results in our follow-up surveys and studies

Benefits management helps, especially during the project execution …

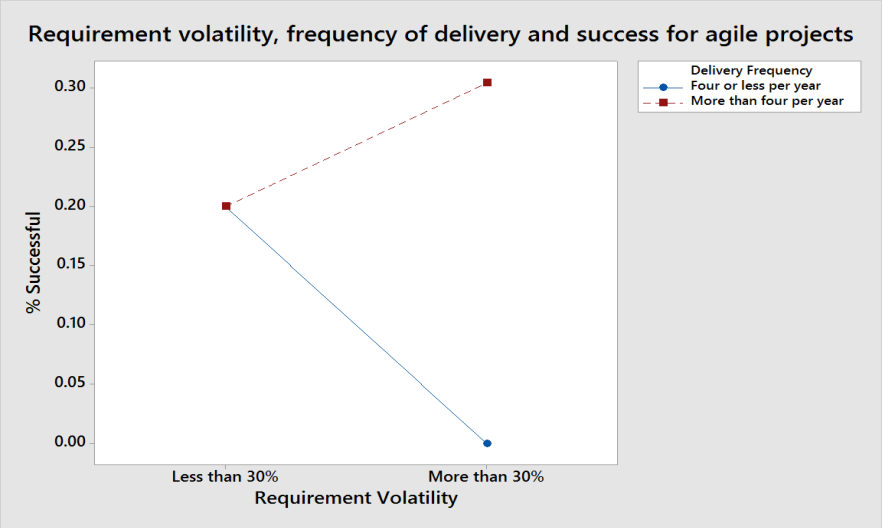
Survey 1:

Benefit management practices	Proportion	Increase in success rate (wrt benefits)
Cost-benefit analysis (up front)	47%	6%
Benefit responsible appointed	57%	22%
Plan for benefit management	33%	31%
Benefit management during proj. execution	53%	34%
Evaluation of benefit during/after proj. exec.	31%	19%

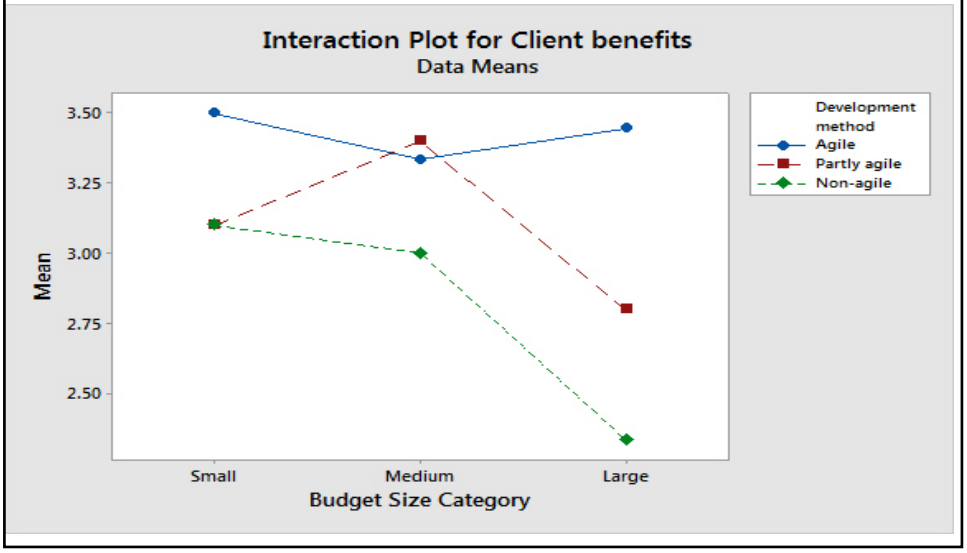
Survey 2 (in-depth study):

Benefit management practices	Present	Not present/don't know
Cost-benefit analysis (up front)	31% with problems	22% with problems
Benefit responsible appointed	28% with problemer	29% with problems
Plan for benefit management	29% with problems	28% with problems
Benefit management during proj. execution	20% with problems	35% with problems

Successful benefits management in a changing world requires agile development



Agile software projects with benefits management practices during project execution seem to be less affected by large project size

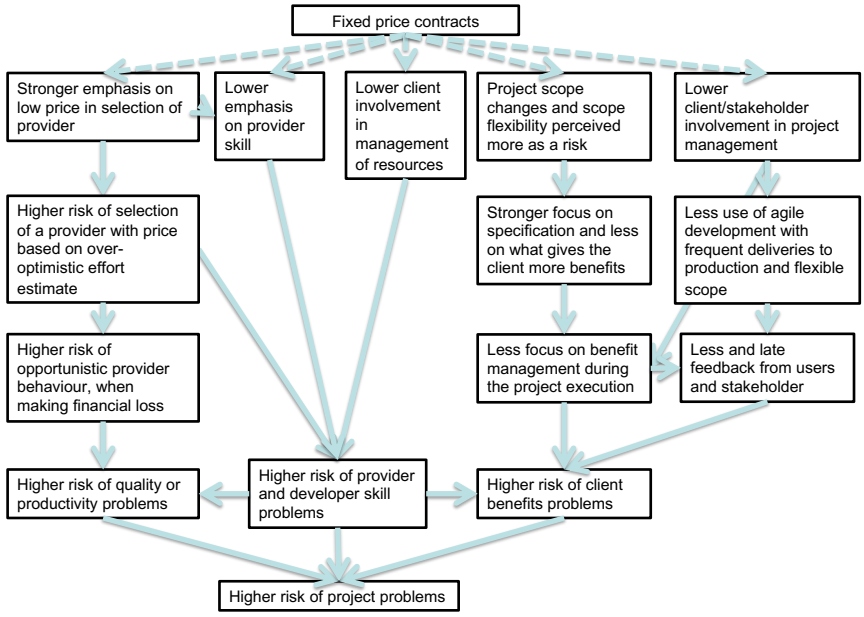


Time & material type of contracts much better for both the client and the provider. Why is that?

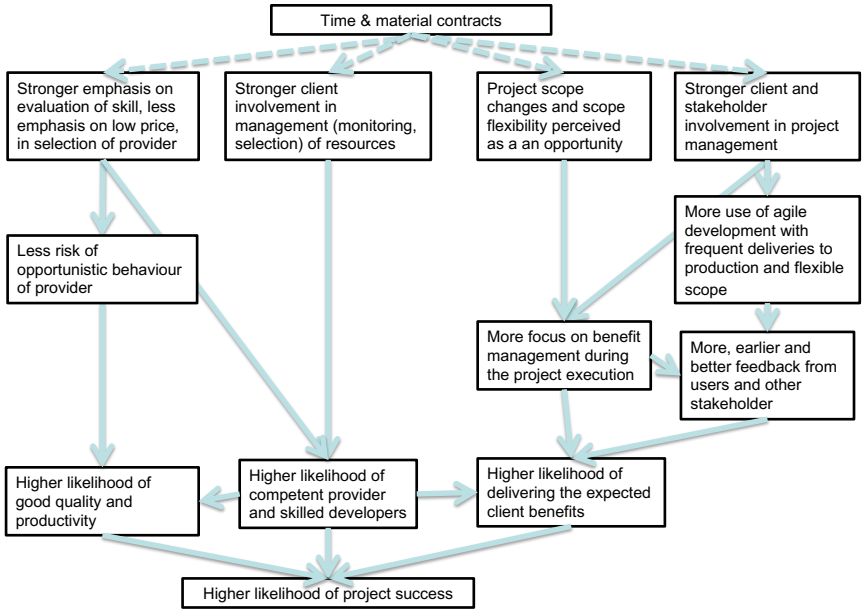
First study: Extremely negative results for Fixed price contracts.

	Fixed price	**Time & Material**
Client benefits	**0% (success rate)**	**59%**
Technical quality	22%	24%
Budget control	33%	31%
Time control	11%	29%
Efficiency	0%	19%

Failure pattern: Interaction between contract, agile and benefits management



Success pattern: Interaction between contract, agile and benefits management …



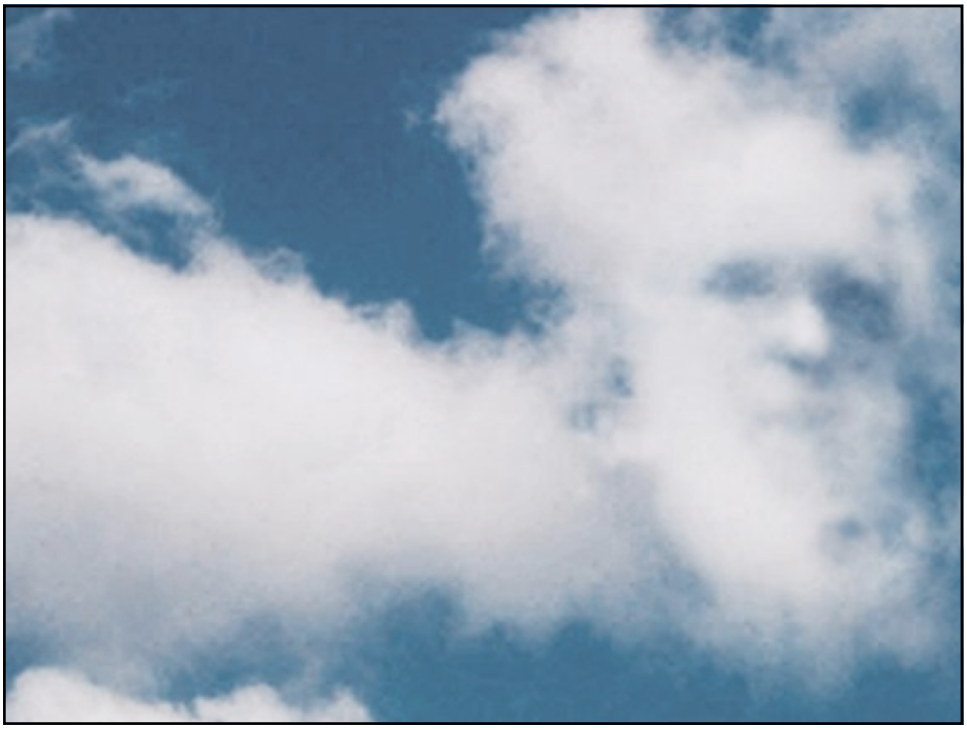
Conclusions

- There are success and failure patterns, not isolated success and failure factors
- Agile development, with its frequent deliveries and flexibility in scope, enables good benefits management during project execution
- Other factors, especially choice of contract, supports or limits the ability to implement good benefits management practices in agile development.
- It is essential that the client is strongly involved in the planning and execution of benefits management

What makes software projects successful?

ICSSE keynote 21 April 2018

Magne Jørgensen
University of Oslo & Simula Metropolitan, CDE



Based on:

- Jørgensen, M. (2016). A survey on the characteristics of projects with success in delivering client benefits. *Information and Software Technology*, *78*, 83-94.
- Jørgensen, M., Mohagheghi, P., & Grimstad, S. (2017). Direct and indirect connections between type of contract and software project outcome. *International Journal of Project Management*, *35*(8), 1573-1586.
- Jørgensen, M. (2017, May). Software development contracts: the impact of the provider's risk of financial loss on project success. In *Proceedings of the 10th International Workshop on Cooperative and Human Aspects of Software Engineering* (pp. 30-35). IEEE Press.
- Do Agile Methods Work for Large Software Projects? (2018, April) To be presented at *XP 2018*, Porto, Portugal.
- Huge investements in digitalization. What does it give us in return? Keynote Software 2018 (DnD's annual conference, Oslo, Norway).

How successful are IT investments?

> *You can see the computer age everywhere but in the productivity statistics.*
> Robert Solow (1987)

This ”productivity paradox” is claimed even today. Is it true?

“… 10% increase in ICT investment leads to a 0.6% increase in growth” (i.e., around half of the current (very low) increase in productivity is due to ICT-investments!)

”… the growth impact of ICT has grown over time.”

A high number of studies (and researcher bias ...)

Information Economics and Policy
Volume 38, March 2017, Pages 38-54

The productivity paradox: A meta-analysis

Petr Polák

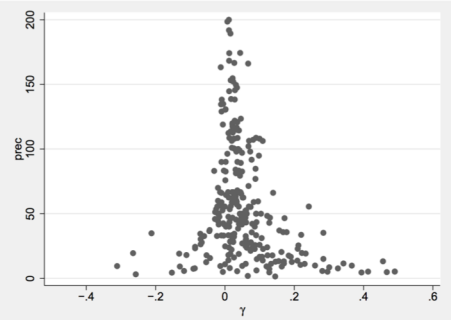


Figure 5: Funnel graph – before 2002

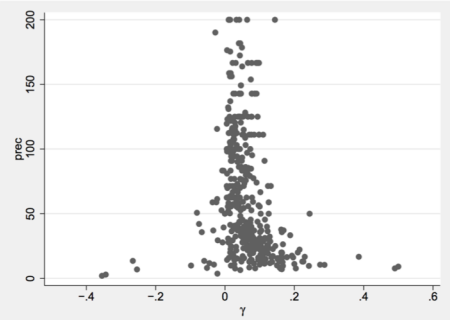


Figure 6: Funnel graph – after 2002

BUT …

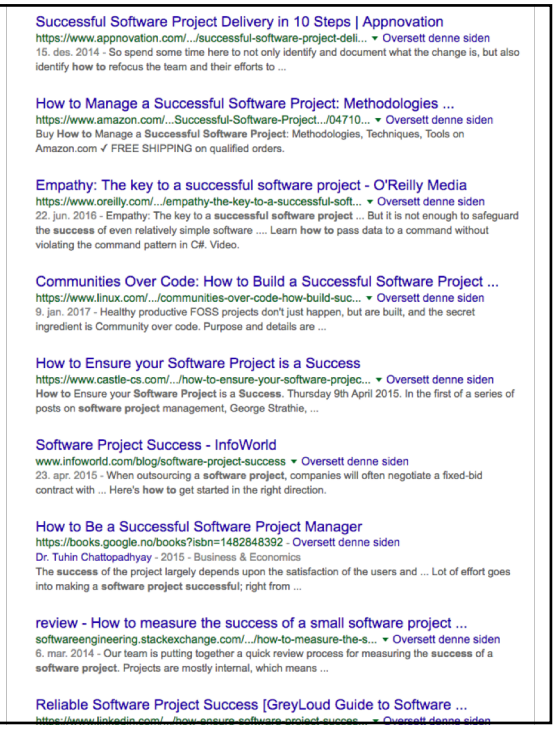
There are much wasted and failed ICT investments …

Around 10% of all IT projects are cancelled or completed with little or no client benefits.

About 50% get into substantial problems with either client benefits, technical quality, cost control, time control or development productivity.

Why don’t we know how to avoid failures and be successful with software development?

There are thousands of reports, research papers and presentations on how to succeed with software development projects



Successful Software Project Delivery in 10 Steps | Appnovation
https://www.appnovation.com/.../successful-software-project-deli... ▾ Oversett denne siden
15. des. 2014 - So spend some time here to not only identify and document what the change is, but also identify **how to** refocus the team and their efforts to ...

How to Manage a Successful Software Project: Methodologies ...
https://www.amazon.com/...Successful-Software-Project.../04710... ▾ Oversett denne siden
Buy **How to** Manage a **Successful Software Project**: Methodologies, Techniques, Tools on Amazon.com ✓ FREE SHIPPING on qualified orders.

Empathy: The key to a successful software project - O'Reilly Media
https://www.oreilly.com/.../empathy-the-key-to-a-successful-soft... ▾ Oversett denne siden
22. jun. 2016 - Empathy: The key to a **successful software project** ... But it is not enough to safeguard the **success** of even relatively simple software .... Learn **how to** pass data to a command without violating the command pattern in C#. Video.

Communities Over Code: How to Build a Successful Software Project ...
https://www.linux.com/.../communities-over-code-how-build-suc... ▾ Oversett denne siden
9. jan. 2017 - Healthy productive FOSS projects don't just happen, but are built, and the secret ingredient is Community over code. Purpose and details are ...

How to Ensure your Software Project is a Success
https://www.castle-cs.com/.../how-to-ensure-your-software-projec... ▾ Oversett denne siden
How to Ensure your **Software Project** is a **Success**. Thursday 9th April 2015. In the first of a series of posts on **software project** management, George Strathie, ...

Software Project Success - InfoWorld
www.infoworld.com/blog/software-project-success ▾ Oversett denne siden
23. apr. 2015 - When outsourcing a **software project**, companies will often negotiate a fixed-bid contract with ... Here's **how to** get started in the right direction.

How to Be a Successful Software Project Manager
https://books.google.no/books?isbn=1482848392 - Oversett denne siden
Dr. Tuhin Chattopadhyay - 2015 - Business & Economics
The **success** of the project largely depends upon the satisfaction of the users and ... Lot of effort goes into making a **software project successful**; right from ...

review - How to measure the success of a small software project ...
softwareengineering.stackexchange.com/.../how-to-measure-the-s... ▾ Oversett denne siden
6. mar. 2014 - Our team is putting together a quick review process for measuring the **success** of a **software project**. Projects are mostly internal, which means ...

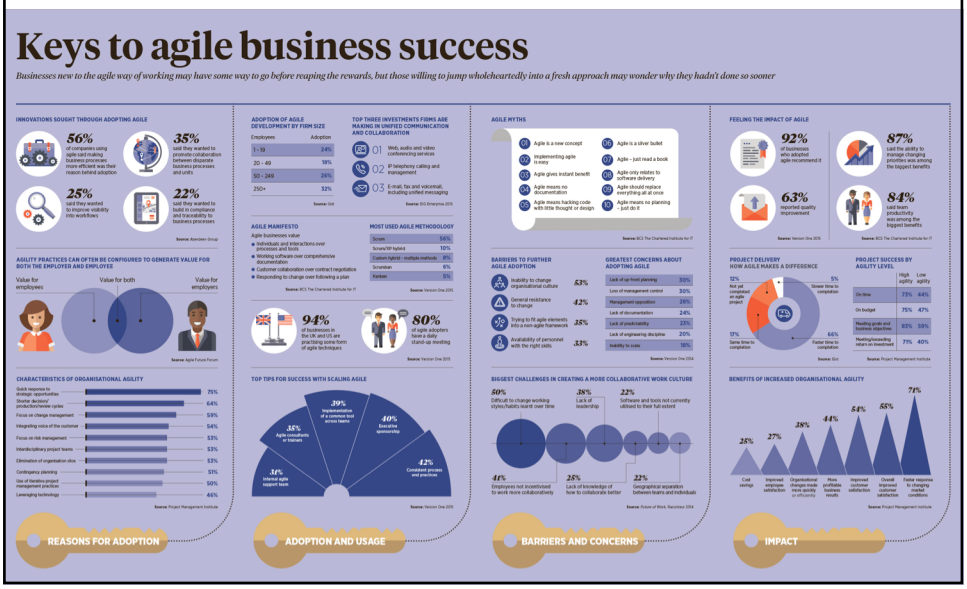
Reliable Software Project Success [GreyLoud Guide to Software ...
https://www.linkedin.com/.../how-ensure-software-project-succes... ▾ Oversett denne siden

Example list of success factors

SUCCESS CRITERIA	IMPORTANCE (POINTS)
1. User Involvement	19
2. Executive Management Support	16
3. Clear Requirements	15
4. Proper Planning	11
5. Realistic Expectations	10
6. Short Project Milestones	9
7. Competent Staff	8
8. Ownership	6
9. Clear Vision & Objectives	3
10. Hard-Working, Focused Staff	3
TOTAL	100

The list of success factors has not changed much since the 1960s!
More or less the same list is for example presented in:
Gotterer, M.H. *Management of computer programmers*. Proceedings of the spring joint computer conference. 1969. ACM

Many methods claiming success



Cobb's paradox?

We know why projects fail, we know how to prevent their failure – so why do they still fail?

What is a proper response to Cobb's paradox? Do software professionals ignore the knowledge?

Cobb's paradox is no paradox. We don't know much about why something fails and how to succeed.

Even worse. The truth is that ...

- The high complexity and innovativeness of product, process and people organization means that we can hardly expect to succeed all the time
- Much of what happens is outside of the control of the project
- Connections are context dependent and hard to identify and understand
- There is a network of connections and we're inherently poor at identifying and understanding indirect relationships
- The relationships are probabilistic and we're inherently poor at understand non-deterministic relationships

... we'll probably never understand fully what it takes to succeed

A diversion into probabilistic relationships

Just to illustrate how poor we are at identifying them

Representativeness bias

(seeing patterns that are not there)

Question: Assume five throws with a fair coin. Which of the following sequences is more likely to occur?
Alt. 1: Head-Head-Head-Head
Alt. 2: Head-Tail-Head-Tail

Answer: Same probability

Relevance: We tend to use to the representative heuristic (Alt 2. is more "representative" of sequence of coin flipping) and think that non-representative sequence (such as Alt. 1) are surprising patterns.

Failure of seeing true patterns

Question: Assume a sequence of throws with a fair coin. Which of the following two sequences is more likely to occur **FIRST**?
Alt. 1: Head-Head
Alt. 2: Tail-Head
Example: Head-Tail-Tail-Head-Head….
→ Tail-Head occurs before Head-Head

Answer: It is 75% likely to first observe Tail-Head and only 25% likely to first observe Head-Head

Relevance: Some probabilistic connections are connected, hidden and non-intuitive. Difficult to see them …

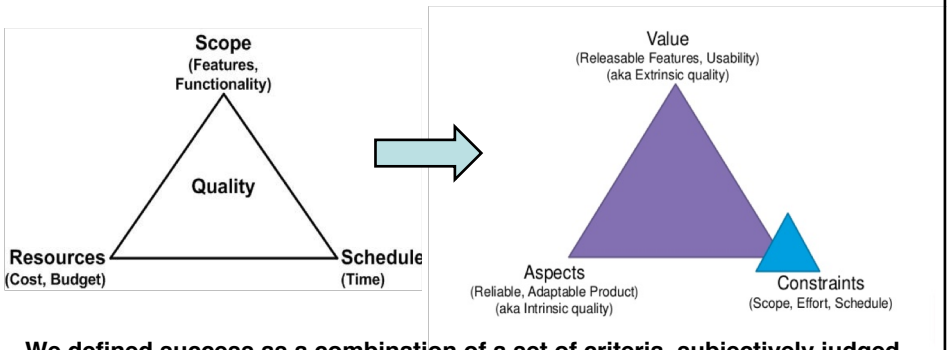
One more... (mainly for fun, but also to show how poor our probabilistic intuition is)

- A country has regulated that no family is allowed to have more than one son, but as many daughters as they want.
- This means that allowed sequences of child-births are:
  - Boy (stop, not allowed to have more children)
  - Girl-Boy (stop)
  - Girl-Girl-Boy (stop)
  - etc.
- **Question**: How does this law affect the proportion of men and women in the country?
- **Answer**: Not effect at all. There will still be about 50-50 men and women

Back to software projects ...

What does it mean to succeed and to fail with software development?

Software project success



We defined success as a combination of a set of criteria, subjectively judged by the client and project manager:

- Client benefits delivered
- Cost control
- Time control
- Development efficiency
- Software properties (technical quality)

Our definition of success and failure of software project

Scale (values for each success criterion):

- Very successful
- Successful
- Acceptable
- Problematic
- Very problematic

Project outcome (for the project as whole):

- **Successful**: Acceptable or better on all five criteria (benefits, cost, time, quality, efficiency)
- **Problematic**: Problematic or worse on at least one criteria
- **Failure**: Cancelled, or delivering no or very little client benefits

Our studies (2015-2017):

- Nine surveys, with 50-200 participants each, representing around 1000 Norwegian software projects in the public and the private sector.
- In-depth, interview-based examination (case studies) of 35 software projects in the public sector of Norway
- Analysis of a data set consisting of more than 400.000 small, international IT-projects/tasks

Success and failure rates found in our studies

All studies gave similar results:

- Around 50-60% successful projects
- Around 30-40% problematic (but not failed) projects
- Around 10% failed projects

Like other studies, we have insufficient control of the representativeness of the samples and with definitions and measures of success. Other contexts, measures and data collection methods, may give other success and failure rates.

More interesting
(and more robust results):

How are things connected?

Question 1: Does the software development method matter?
(Does it help to work agile?)

Common belief (amongst agile people): Yes

Our studies: Yes, agile helps, but …

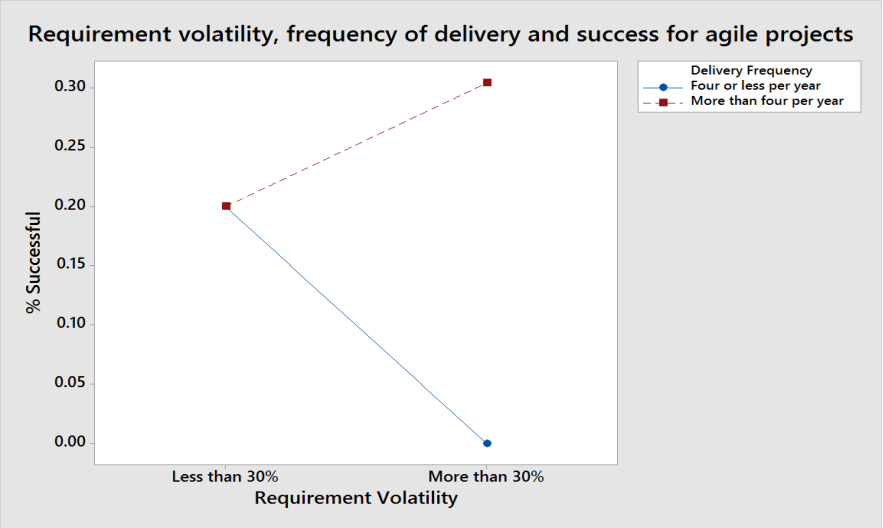
The numbers show the increase (in percent points) in proportion of successful projects

	Agile	Frequent delivery to production	Flexible scope
Client benefits	16%	22%	29%
Technical quality	21%	6%	32%
Budget control	2%	22%	29%
Time control	8%	11%	24%
Efficiency	11%	5%	24%

… only when including frequent delivery to production and flexible scope. Agile projects not including these practices were LESS successful than non-agile projects!

Similar results in our follow-up surveys and studies

Agile is not agile (requirement change and type of agile development)



Question 2: Are larger (and presumably more complex) projects less successful?

Common belief: Yes

Our (initial) result: No

Large projects not less successful than smaller ones (similar finding in all studies)

Criterion	< 1 mill Euro	1-10 mill Euro	> 10 mill Euro
Client benefits	31%	47%	35%
Tech. quality	24%	28%	25%
Budget control	24%	47%	47%
Time control	29%	35%	35%
Efficiency	24%	12%	24%

The numbers (percentages) represent the proportion of projects assessed to be successful or very successful with respect to a success criterion.

But, the first results hid that we only had studied completed projects

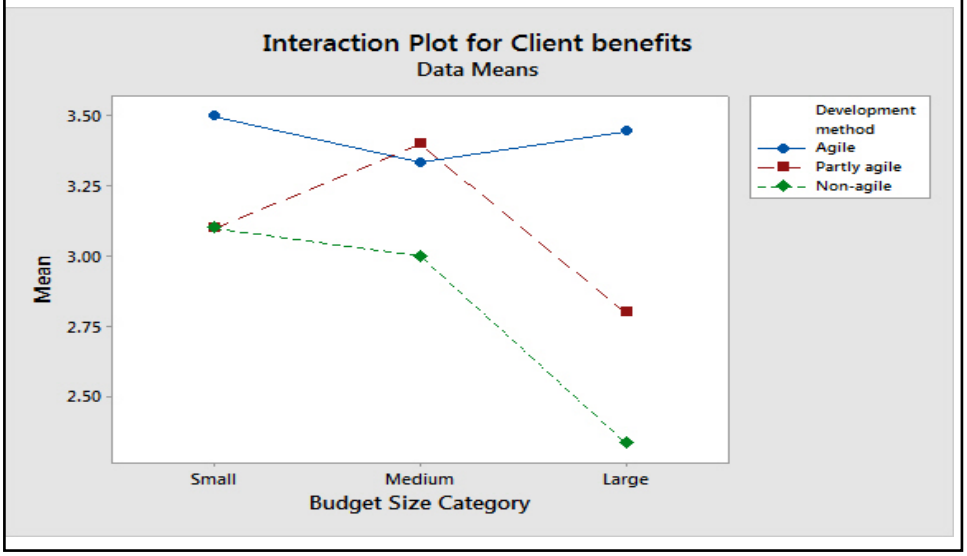
Adding non-completed projects in follow-up studies gave that the largest projects (> 10 mill Euro) were strongly over-represented in the group of failed projects (2-3 times more frequent).

A rule of thumb (based on offshoring projects) is that ten times larger project size leads to twice the risk of failure.

Also of interest:

- Different reasons for problems for small and large projects.
- Higher risk of failure with larger projects should not be used to divide ”logical connected deliveries” into separate projects.

Agile software projects seem to be less affected by large project size



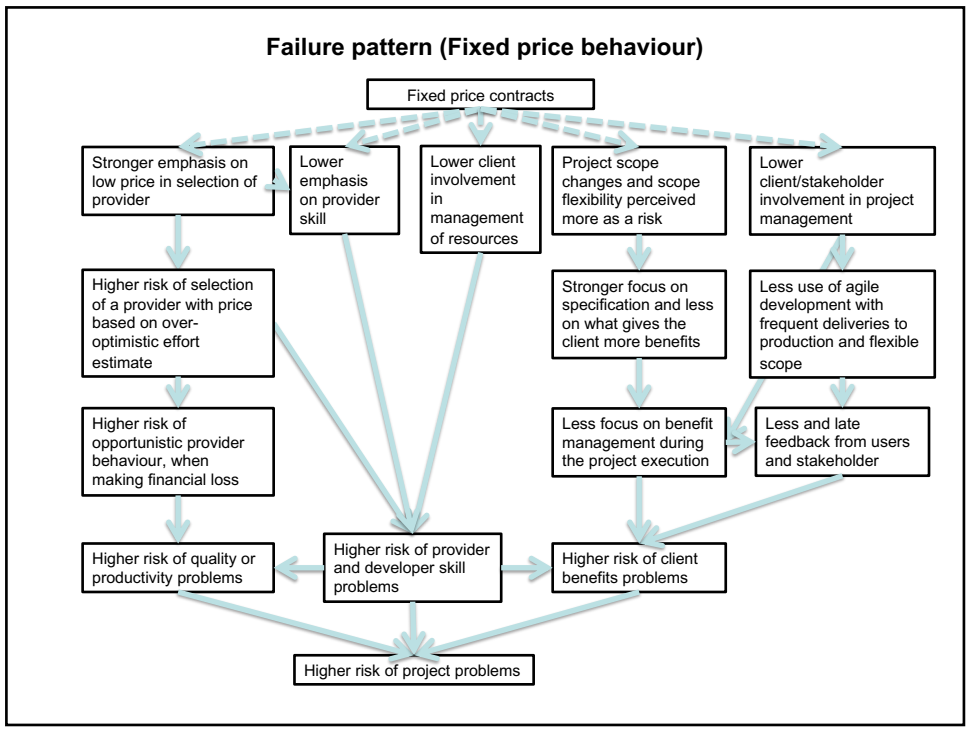
Question 3: Does contract type matter?

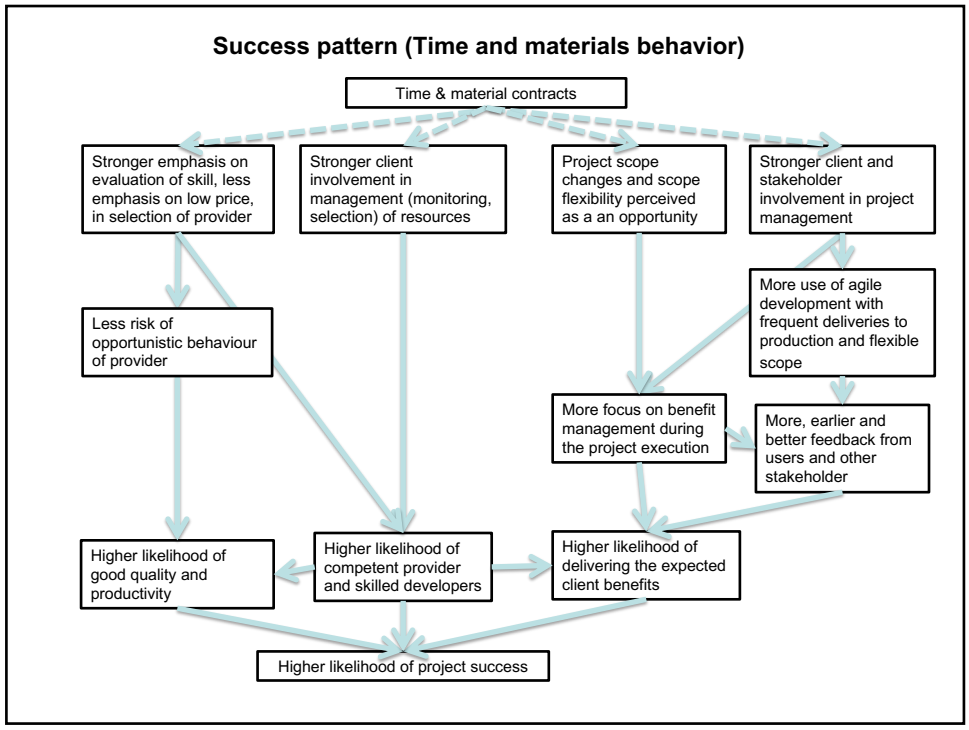
Common belief (amongst clients)**:**
Fixed price contracts is the better (for us)

Our finding: Time & material type of contracts much better for both the client and the provider

First study: Extremely negative results for Fixed price contracts.

	Fixed price	**Time & Material**
Client benefits	**0% (success rate)**	**59%**
Technical quality	22%	24%
Budget control	33%	31%
Time control	11%	29%
Efficiency	0%	19%


Failure pattern (Fixed price behaviour)
Fixed price contracts
Stronger emphasis on low price in selection of provider
Lower emphasis on provider skill
Lower client involvement in management of resources
Project scope changes and scope flexibility perceived more as a risk
Lower client/stakeholder involvement in project management
Higher risk of selection of a provider with price based on over-optimistic effort estimate
Stronger focus on specification and less on what gives the client more benefits
Less use of agile development with frequent deliveries to production and flexible scope
Higher risk of opportunistic provider behaviour, when making financial loss
Less focus on benefit management during the project execution
Less and late feedback from users and stakeholder
Higher risk of quality or productivity problems
Higher risk of provider and developer skill problems
Higher risk of client benefits problems
Higher risk of project problems


Success pattern (Time and materials behavior)
Time & material contracts
Stronger emphasis on evaluation of skill, less emphasis on low price, in selection of provider
Stronger client involvement in management (monitoring, selection) of resources
Project scope changes and scope flexibility perceived as a an opportunity
Stronger client and stakeholder involvement in project management
More use of agile development with frequent deliveries to production and flexible scope
Less risk of opportunistic behaviour of provider
More focus on benefit management during the project execution
More, earlier and better feedback from users and other stakeholder
Higher likelihood of good quality and productivity
Higher likelihood of competent provider and skilled developers
Higher likelihood of delivering the expected client benefits
Higher likelihood of project success

Question 4: Does it help with "benefits management"?

Common belief: Yes

Our finding: Not all benefit management practices led to much improvements

Survey 1:

Benefit management practices	Proportion	Increase in success rate (wrt benefits)
Cost-benefit analysis (up front)	47%	6%
Benefit responsible appointed	57%	22%
Plan for benefit management	33%	31%
Benefit management during proj. execution	53%	34%
Evaluation of benefit during/after proj. exec.	31%	19%

Survey 2 (in-depth study):

Benefit management practices	Present	Not present/don't know
Cost-benefit analysis (up front)	31% with problems	22% with problems
Benefit responsible appointed	28% with problemer	29% with problems
Plan for benefit management	29% with problems	28% with problems
Benefit management during proj. execution	20% with problems	35% with problems

Characteristics of the successful project

Success pattern

- Good control of ambition level. Avoiding ”too much” at the same time and good at saying ”no” to adding complexity.
- Use of contracts that avoid ”fixed price”-behavior.
- Client with competence to select and manage competent providers and individual resources (not so much focus on low price)
  - Selection of resources from more than one provider
- Flexibility in scope (not only ”must have”-functionality)
- Client is (as a minimum) strongly involved in the planning and execution of benefits management.
- Use of agile development with frequent deliveries to production (or at least with proper testing/feedback from real users)
- Early start of involvement of stakeholders (especially the users) and planning and preparing for deployment.

Extra material

Regional differences in failure rate

Table: Client = columns, Provider = rows

Client / Provider	AF	EA	EE	LA	ME	NA	OC	SA	WE	Total
AF (Africa)	14% (92)	22% (289)	26% (137)	19% (105)	23% (195)	16% (3944)	12% (692)	26% (306)	15% (183)	17% (7633)
EA (East Asia)	20% (332)	16% (1660)	19% (856)	15% (662)	18% (970)	12% (27447)	12% (3953)	25% (1416)	15% (10676)	14% (48023)
EE (East Europe)	11% (1285)	14% (5010)	13% (5278)	11% (2618)	14% (4325)	9% (114728)	10% (11473)	18% (4355)	10% (51088)	10% (201565)
LA (Latin America)	12% (127)	16% (523)	14% (540)	11% (985)	15% (493)	10% (17245)	9% (1888)	20% (499)	12% (6369)	11% (28868)
ME (Middle East)	16% (231)	25% (622)	16% (635)	17% (320)	17% (824)	13% (15881)	13% (1973)	26% (792)	15% (6494)	14% (27883)
NA (North America)	19% (2713)	20% (2713)	16% (2143)	20% (1352)	19% (2112)	13% (86346)	15% (8161)	25% (2049)	15% (23947)	14% (130919)
OC (Oceania)	14% (58)	18% (260)	26% (149)	26% (82)	19% (182)	12% (6656)	9% (1474)	24% (205)	15% (2303)	13% (11484)
SA (South Asia)	17% (2614)	23% (7729)	22% (4861)	19% (3599)	20% (5632)	16% (143699)	15% (18958	24% (10934)	18% (54710)	17% (254075)
WE (Western Europe)	13% (470)	17% (2070)	14% (1779)	14% (960)	15% (1927)	13% (38544)	14% (4250)	23% (1529)	13% (20111)	13% (72297)
Total	16% (5734)	19% (20935)	17% (16393)	16% (10702)	18% (16714)	13% (456106)	13% (52894)	23% (22113)	14% (177852)	

Failure factors from a study of 400.000 small projects

Predictor variable	Coefficient	p-value	Odds ratio	95% confidence interval	
				Lower	Upper
Constant	-2.90	0.00			
SatisfactionScoreProviderCat=Low	0.35	0.00	1.42	1.39	1.45
SatisfactionScoreProviderCat=No Scores	0.91	0.00	2.49	2.33	2.67
FailureRateProviderCat=Low	-0.66	0.00	0.52	0.51	0.53
FailRateProviderCat=No Projects	-0.34	0.00	0.71	0.67	0.76
SkillTestPassRateProviderCat=Low	0.07	0.00	1.07	1.02	1.12
SkillTestPassRateProviderCat=No Tests	0.58	0.00	1.79	1.74	1.85
SatisfactionScoreClientCat=Low	0.18	0.00	1.20	1.17	1.23
SatisfactionScoreClientCat=No Scores	0.25	0.00	1.28	1.23	1.33
FailureRateClientCat=Low	-0.64	0.00	0.53	0.52	0.54
FailureRateClientCat=No Projects	-0.63	0.00	0.53	0.51	0.56
PreviousCollaboration=Yes	-1.74	0.00	0.17	0.17	0.18
FocusLowPriceCat=Low	-0.19	0.00	0.83	0.81	0.85
FocusLowPriceCat=Medium	-0.08	0.00	0.92	0.89	0.95
FailureRateProviderRegionCat=High	0.27	0.00	1.31	1.28	1.33
FailureRateClientRegionCat=High	0.42	0.00	1.53	1.48	1.58
GeographicalDistance=Neighbor	-0.07	0.02	0.93	0.90	0.97
GeographicalDistance=Offshore	0.02	0.10	1.02	1.00	1.05
logProjectSize	0.71	0.00	2.03	1.99	2.06

Jørgensen, Magne. "Failure factors of small software projects at a global outsourcing marketplace." Journal of systems and software 92 (2014): 157-169.

PRODUKTIVITETSVEKST gjennom digitalisering (og annen teknologi) er trolig det som – om noe – vil kunne redde velstanden til en **aldrende befolkning** (nedgang i andel i lønnet arbeid), som jobber **færre timer**, med **høyere lønn**, og som investerer **mindre i forskning og utvikling** enn de fleste andre industriland.

Velstandsland = Lykkeland?
«World happiness report»

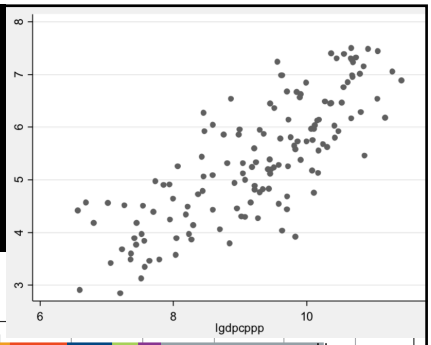
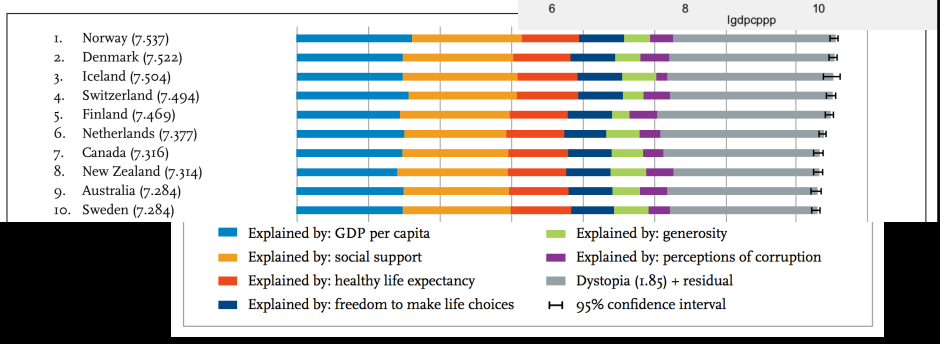


Figure 2.2: Ranking of Happiness 2014-2016 (Part 1)



MEN, da må digitaliseringen faktisk føre til økt produktivitet.

Gjør den det?

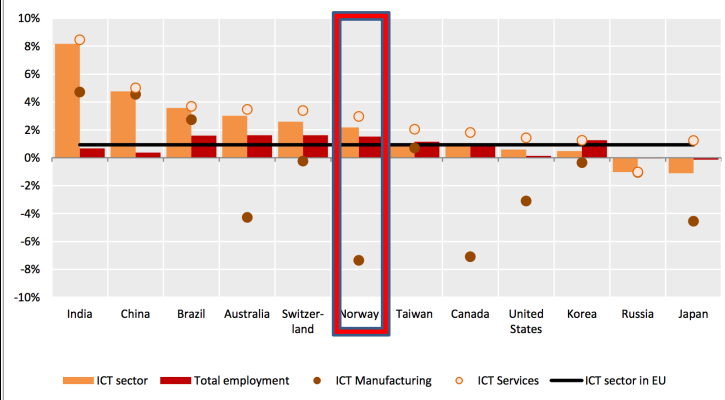
Hvor mye investeres?
Hvor mye får vi igjen?
Har dette forholdet endret seg over tid?

VI INVESTERER TROLIG STADIG MER I DIGITALISERING

(VANSKELIG Å FINNE GODE TALL …)

Antall ansatte i IKT-sektoren øker med ca. 2% i året (mer enn snittet i EU)

Figure 15: Mean annual growth rate of ICT sector and total employment in the European Union and other non-EU economies (2006-2014)

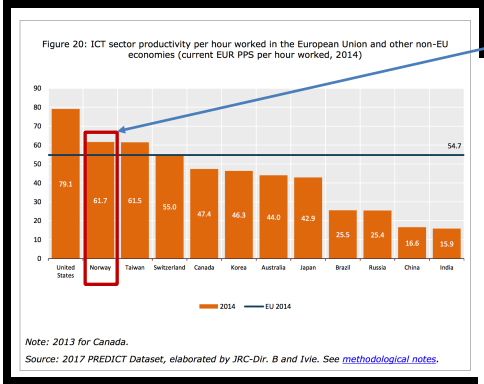


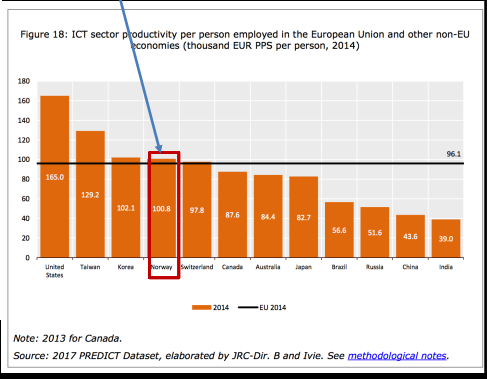
Source: 2017 PREDICT Dataset, elaborated by JRC-Dir. B and Ivie. See methodological notes.

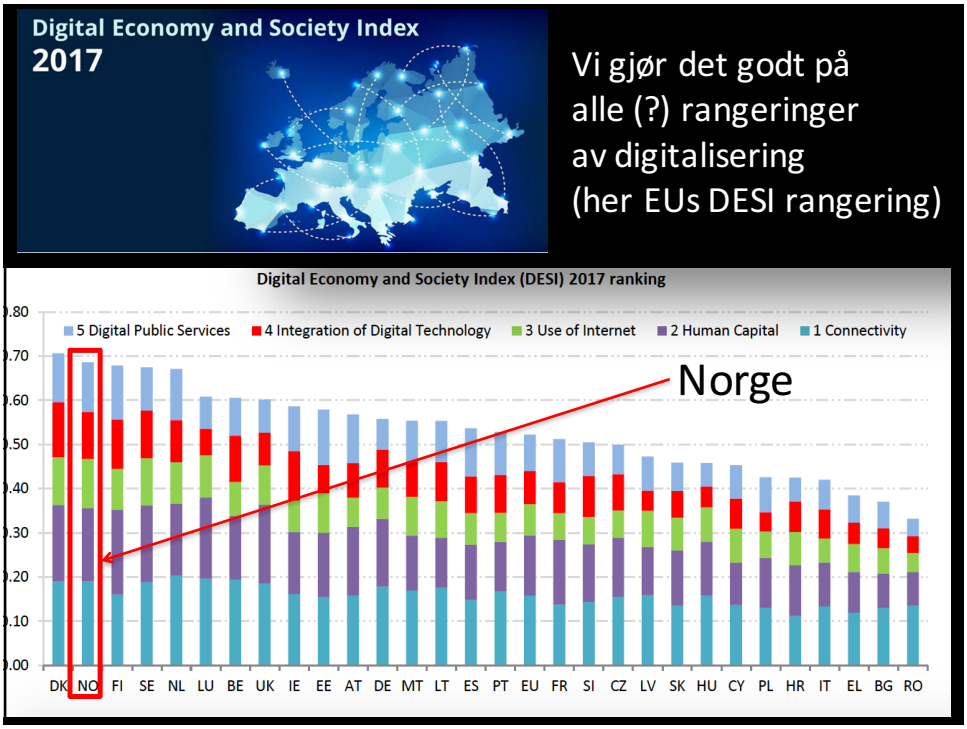
NORGE BLIR STADIG MER DIGITALISERT, OG VI GJØR DETTE RASKERE OG MER EFFEKTIVT ENN DE FLESTE ANDRE LAND

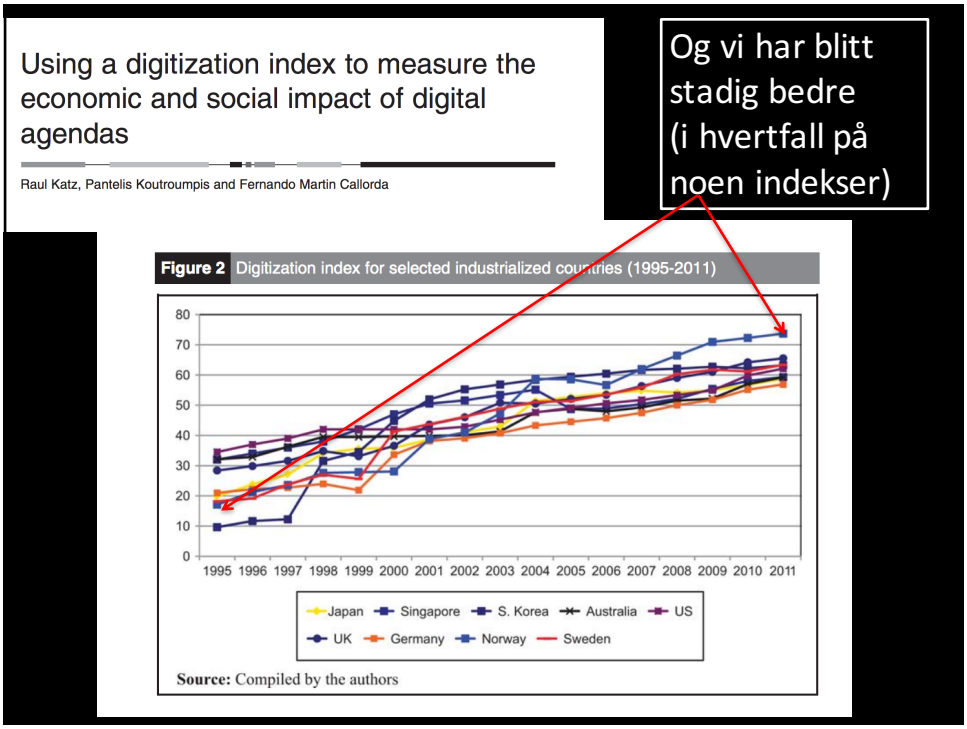
God produktivitet per arbeidstime (ikke så god per ansatt)

Norge






Digital Economy and Society Index
2017
Vi gjør det godt på
alle (?) rangeringer
av digitalisering
(her EUs DESI rangering)
Digital Economy and Society Index (DESI) 2017 ranking
5 Digital Public Services
4 Integration of Digital Technology
3 Use of Internet
2 Human Capital
1 Connectivity
Norge
DK NO FI SE NL LU BE UK IE EE AT DE MT LT ES PT EU FR SI CZ LV SK HU CY PL HR IT EL BG RO


Using a digitization index to measure the economic and social impact of digital agendas
Raul Katz, Pantelis Koutroumpis and Fernando Martin Callorda
Og vi har blitt
stadig bedre
(i hvertfall på
noen indekser)
Figure 2 Digitization index for selected industrialized countries (1995-2011)
1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011
Japan
Singapore
S. Korea
Australia
US
UK
Germany
Norway
Sweden
Source: Compiled by the authors

DERSOM ØKT DIGITALISERING GIR ØKT PRODUKTIVITET, SÅ SKULLE VI FORVENTE EN ØKT PRODUKTIVITETSVEKST DE SENERE ÅRENE

MEN, VI SER DET MOTSATTE. PRODUKTIVITETSVEKSTEN HAR GÅTT NED

NOU Norges offentlige utredninger 2015: 1

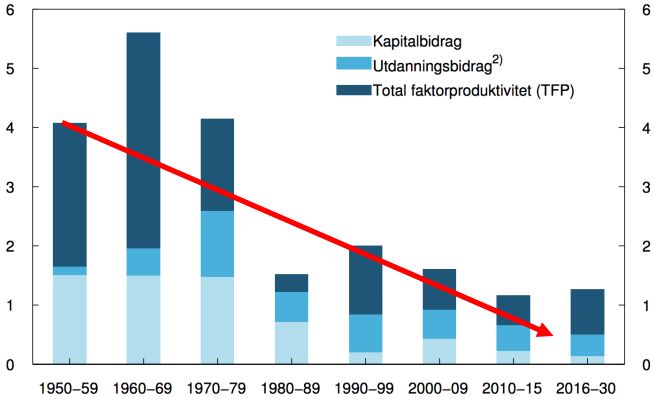
Produktivitet – grunnlag for vekst og velferd

Produktivitetskommisjonens første rapport

«Produktivitetsveksten i Norge har etter 1970 vært sterkere enn i de fleste andre land. Produktivitetsvekst har vært den viktigste forklaringen på den sterke velstandsveksten Norge har hatt i denne perioden. Produktivitetsveksten har falt etter 2005. Dette er bekymringsfullt dersom det er et uttrykk for et langsiktig fenomen, og ikke bare et utslag av midlertidige forstyrrelser.»

Fra årstalen til sentralbanksjef Olsen

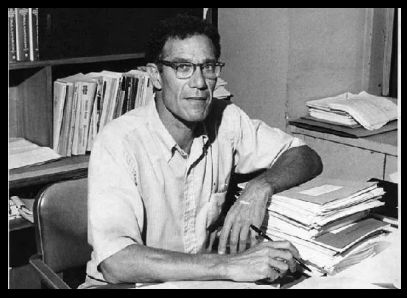
Figur 8 Produktivitetsvekst i Norge.
BNP per timeverk uten olje.[1)] Faste priser. Årsvekst. Prosent



1) Anslag 2016 – 2030.
2) Beregningen er basert på metode fra Hall, R. og C. Jones (1998) «Why do some countries produce so much more output per worker than others». *NBER Working Paper* 6564. Marginalavkastning av utdanning er satt til seks prosent.
Kilder: Statistisk sentralbyrå og Norges Bank

Det undersøkelser typisk viser

- Digitaliserings**andelen** av produktivitetsveksten var først høy (1990-tallet), ble så lavere (2000-tallet), for i det senere å ha vært høyere (**typisk 30-50%!)**.
- Enkel sagt: Digitalisering forårsaker en stor andel av produktivitetsveksten, men veksten blir bare lavere og lavere – selv om vi investerer mer og mer.



DETTE KALLES OFTE «PRODUKTIVITETSPARADOKSET» FØRST FRAMSATT AV SOLOW I 1987- OG DISKUTERES FORTSATT HEFTIG

Litt om metodikken (Solow-modellen):

$$\ln Y(t) = \ln B(t) + \alpha \ln K(t) + (1 - \alpha) \ln L(t)$$

Y er produksjonen, B er "total faktorproduktivitet", K er kapitalinnsats (som kan deles opp i investeringer i digitalisering og annet), L er arbeid.

Bruk av modellen (regresjonsanalyse) gir typisk at en 10% økning i digitaliseringskapital gir ca. 0.6% produktivitetsvekst (Y/L = arbeidsproduktivitet).

Problemer

- Kun direkte effekter og ingen interaksjonseffekter.
- Antar at modellen er riktig spesifisert og at regresjonsantagelser ikke brytes.
- Tar ikke med økt kvalitet (verdi = pris)
- Får ikke med produktivitetsvekst der verdiskapning ikke kan «prises», som digitalisering av offentlig sektor, gratis søketjenester, selvbetjening, delingsøkonomi og spill-over effekter.
- Tar ikke med forsinkede effekter

… og så har vi forskerproblemer (publikasjonsskjevhet)

ELSEVIER

Information Economics and Policy

Volume 38, March 2017, Pages 38-54

The productivity paradox: A meta-analysis

Petr Polák

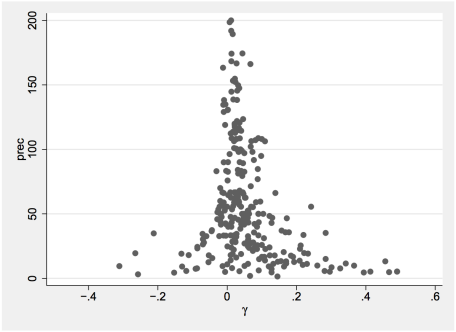


Figure 5: Funnel graph – before 2002

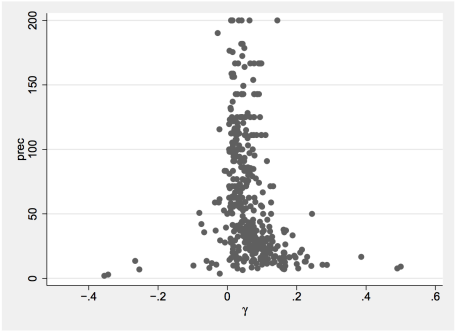


Figure 6: Funnel graph – after 2002

Trolig vesentlige spill-over effekter

Oxford Bulletin of Economics and Statistics

Published on behalf of the University of Oxford's Department of Economics

Explore this journal >

Original Article

Knowledge Spillovers, ICT and Productivity Growth†

Carol Corrado, Jonathan Haskel, Cecilia Jona-Lasinio

First published: 24 March 2017 Full publication history

"We find that (a) the marginal impact of ICT capital is higher when it is complemented with intangible capital."
"Intangible capital" er f eks søketjenester, gratis programvare, kompetanseøkning og merkevarebygging.

Hvor stor del av investeringene kan i det hele tatt gi oss produktivitetsvekst?

Information Systems Evolution Efficiency – Differences between the Public and the Private sector

John KROGSTIE, Tor Kristian VELD
IDI, NTNU, Trondheim Norway

- Totalt sett brukes ca en tredjedel (34%) til utvikling av ny funksjonalitet (nye systemer eller ny funksjonalitet i eksisterende)
  - Og da er det ikke inkludert det meste av investeringer i kompetanse, omstilling, utredninger, m.m.
  - Offentlig sektor med lavere andel enn privat sektor (30% vs. 40%).
- 17% brukes til utvikling av nye systemer, 41% til forvaltning, og 42% til brukerstøtte og drift. Stabilt siden minst 1993.
- Hvor stor del er "disruptiv" digitalisering? Vanskelig å si, men trolig en forsvinnende liten andel.

Noe investering gir reelt sett produktivitetsnedgang …

Store prosjekter har større risiko for å mislykkes enn små prosjekter.

Ca. 10% av alle IKT-prosjekter blir enten stoppet eller leverer svært lite kundenytte.

Ca. 50% får vesentlige problemer enten med nytte, kvalitet, kostnadskontroll, tidskontroll eller produktivitet

www.hitledelse.com

Gjør digitalisering oss mer lykkelige? (annet enn gjennom velstandsøkning)

Norwegian Institute of International Affairs

ICT, growth and happiness

Per Botolf Maurseth

Research Policy xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Research Policy

journal homepage: www.elsevier.com/locate/respol

ELSEVIER

Internet use and well-being: A survey and a theoretical framework

Fulvio Castellacci*, Vegard Tveito

TIK Centre, University of Oslo, Norway

ARTICLE INFO

JEL classifications:
O33
I31
D31

Keywords:
ICTs
Internet
Innovation
Well-being

ABSTRACT

How does Internet use affect well-being? This paper presents a survey of the literature investigating this question, and it develops a framework to analyze both positive effects and potential risks. We point out four distinct channels through which Internet can shape well-being: it changes time use patterns, creates new activities, facilitates access to information, and acts as powerful communication tool. We show how these four channels impact well-being in distinct domains of life. A central point emerging from the literature review is that the effects of Internet on well-being are mediated by a set of personal characteristics that are specific to each individual: psychological functioning, capabilities, and framing conditions (culture and beliefs). Hence, it is the interaction between human beings' activities in distinct domains of life and their own personal characteristics that explains why the use of Internet has stronger positive effects for some individuals and social groups than others.

Det meste av empiri tyder på at digitalisering (stort sett) har positive effekt på fornøydhet

- Unngår kjedelig arbeid som å fylle ut selvangivelse, mer spennende bruk av fritid, større sosiale nettverk, lettere tilgang på informasjon, ...

- **Bekymringer**: Jobb-trygghet, personvern, overforbruk/avhengighet, isolasjon, ...
- **Effekt avhenger av**: Kompetanse, utdannelse, helse, kultur, politikk, ...

Lykke-modellen (regresjonsanalyse, nok en gang) tyder på at økt internettbruk gir et lite direkte bidrag til økt lykke.

1%-poeng økning i antall internettbrukere gir et utslag på 0.006 på landets lykkesnitt

Table 6

	Random effects	
Variable	Helliwell et al. 2016	With internet
Ln(GDP per cap.)	0.494 (0.054)***	0.385 (0.064)***
Sosial support	1.530 (0.221)***	1.392 (0.239)***
Health	0.011 (0.007)	0.012 (0.008)
Freedom	0.963 (0.160)***	0.793 (0.179)***
Generosity	0.562 (0.141)***	0.478 (0.157)***
Perc. of corr.	-0.656 (0.159)***	-0.675 (0.169)***
Internetusers		0.006 (0.002)***
Unemployment		-0.019 (0.005)***
Year fixed effect	Yes	Yes
Number of countries	153	135
N	1108	890
R^2	0.74	0.76
R^2 (within)	0.16	0.17
R^2 (between)	0.79	0.80

Digitalisering lønner seg - noen mysterier består

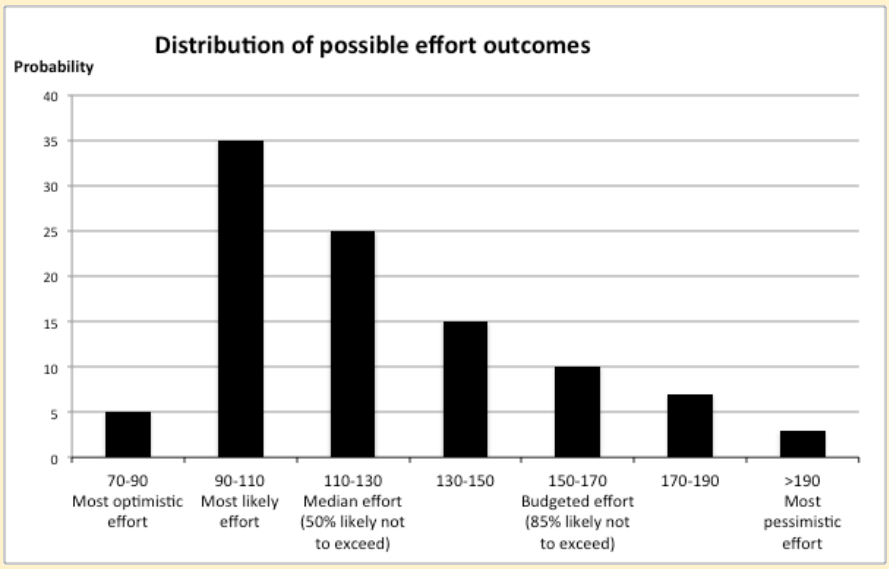
- Vi trenger digitalisering som gir produktivitetsvekst for å beholde vår velstand.
- Velstående land = lykkelige land. Digitalisering ser ut til å gjøre oss litt lykkeligere.
- Godt dokumentert at en stor (kanskje økende) andel av en stadig lavere produktivitetsvekst skyldes økte investeringer i digitalisering.
- Problematisk, men mulig, å forsvare mer digitalisering i en tid der produktivitetsveksten har gått ned, mens digitaliserings-investeringene har gått opp.
- Alt kan snus fra litt dystre til svært lovende utsikter med én stor innovasjon. Kunstig intelligens?

SPØRSMÅL?

Does the software industry know and communicate what they mean with an effort estimate?

(Do **you** know what an estimate is?)

An estimate is an estimate is an estimate?



Estimate =
Best case?
Most likely?
Median?
With contingency added?
Something else?

A proper communication of what we mean with an estimate requires a probabilistic understanding!

It's fine to give a single point estimate, as long as we tell where at the distribution we are, e.g., that we communicate a most likely or p50-estimate (median estimate).

A survey among software professionals

"You have just estimated the number of work-hours you think you need to develop and test four different software systems. Please select the description below that you think is closest to what you meant by your effort estimate in the previous four estimation tasks:

- *Number of work-hours I will use given that I experience no or almost no major problems.* [Ideal effort]
- *Number of work-hours I most likely will use.* [Most likely effort]
- *Number of work-hours where it is about just as likely that I will use more as it is that I will use less effort than estimated.* [Median effort/p50]
- *Number of work-hours where it is unlikely that I will use more effort than estimated.* [Risk averse effort/budgeted effort/…]
- *Number of work-hours based on my expert judgment/feeling of how many work-hours I will use. I find it difficult to decide about the exact meaning of the estimate.* [Don't know/gut feeling]
- *None of the above descriptions is close to what I typically mean by an effort estimate.*

Results (replicated in other surveys)

Interpretation (as claimed in hindsight)	Frequency of interpretation
Ideal effort	37%
Most likely effort	27%
Median effort (p50)	5%
Risk averse effort	9%
Don't know/gut feeling/other	22%

Similar problems (probably worse) with estimates of benefits ...

I analysed more than 100 cost-benefit plans of Norwegian IT projects. **None** were explicit about what they meant with their estimated benefits or estimated profit.

Sometimes software companies try to include uncertainty in their estimates.

Some provide and add uncertainty as shown below
Exercise: Find (at least) five problems

Activity	Minimum effort (best case, optimistic)	Estimate	Maximum effort (worst case, pessimistic)
Activity A	15 work-hours	20 work-hours	25 work-hours
Activity B	40 work-hours	60 work-hours	80 work-hours
Activity C	45 work-hours	50 work-hours	55 work-hours
SUM effort	**100 work-hours**	**130 work-hours**	**160 work-hours**

1. **Not communicating of what is meant** by minimum, estimate (most likely?) and maximum
2. **Too symmetric intervals.** The outcome distribution is typically right-skewed.
3. **Too narrow intervals.** Strong tendency towards too narrow effort intervals to reflect, for example, a 90% confidence inerval.
4. **Incorrect additions.** It is only the mean values that can be safely added, not the most likely, the minimum or the maximum effort. Adding most likely estimates leads to underestimation in a right-skewed world. (For benefits, which may be left-skewed (?), this may lead to over-estimation.)
5. **No dependencies**. Most projects have dependencies between activities, e.g., testing is 40% of development. Not including this, leads to even more underestimation.

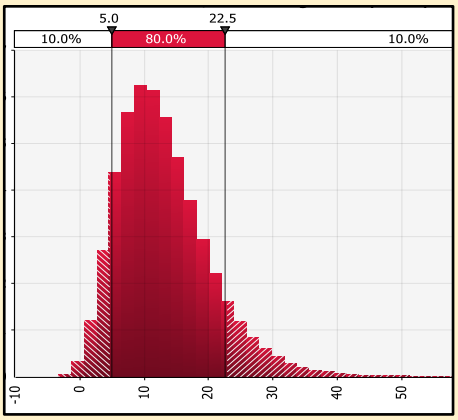
A brief side-track on adding estimates in a right-skewed world

Assume project X

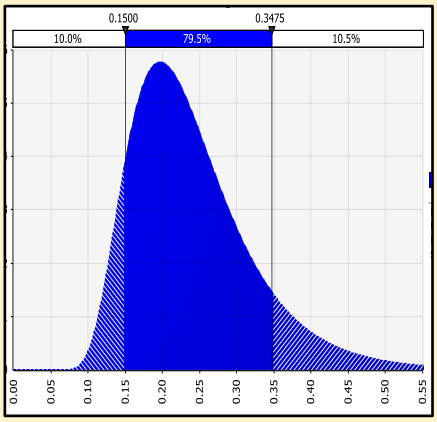
- Ten user stories, where all have the same (right-skewed) effort outcome distribution
  - Minimum (p10): 5 hour
  - **Most likely: 10 hours**
  - Maximum (p90): 22.5 hours
- Add-on activities (dependencies): 5 activities calculated as proportions of the sum of the ten user stories (administration, system test, ...). All of them have the same right-skewed effort outcome distribution
  - Minimum (p10): 15% of the effort on the user stories
  - **Most likely: 20% of the effort on the user stories**
  - Maximum (p90): 35% of the effort on the user stories

The effort distributions (log-normal, right-skewed)

A user story



An add-on activity

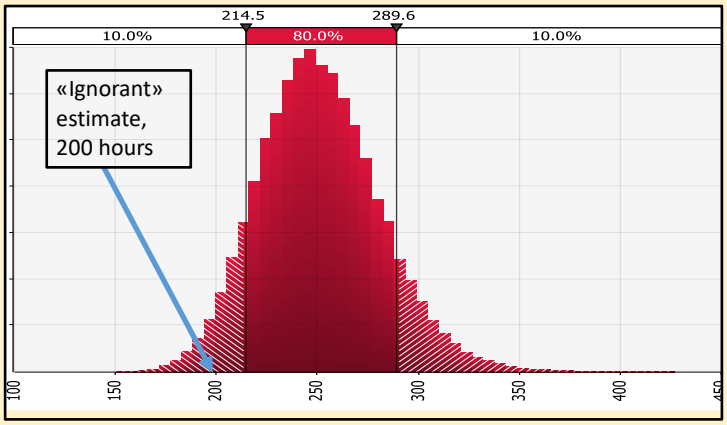


«Ignorant» adding of activities

- SUM User stories
  - 10 x most likely (10 hours) = 100 hours
- SUM Add-on activities
  - 5 x 0.2 x SUM User stories = 100 hours
- SUM TOTAL
  - 100 tv + 100 tv = 200 hours
- Gives a STRONG underestimation!
- **Very unlikely to use 200 hours or less!**

The sum of the most likely effort is NOT the most likely sum

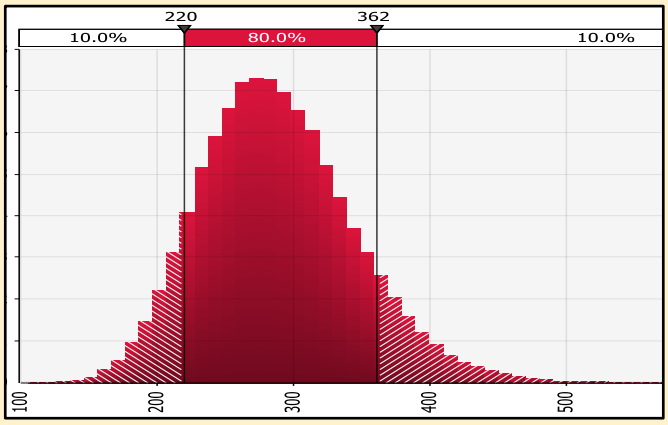
Sum of the effort distributions



P10 = 215
Most likely = 246
P50 (median) = 250
Expected value (mean) = 251
P85 = 282
P90 = 290

Still not optimal, not right-skewed.
Central limit theorem + no dependencies → symmetric

Sum of effort distributions with dependencies (add-on dep.)



P10 = 220 (up from 215)
Most likely = 265 (up from 246)
P50 (median) = 284 (up from 250)
Expected value (mean) = 288 (up from 251)
P85 = 345 (up from 282)
P90 = 361 (up frok 290)

Large increase in most likely estimates, nd it is right-skewed.
Median estimate (50% likely) is 284 (vs. “ignorant“ of 200 and “no dependencies“ of 250)

The most ”advanced” companies do it with asymmetric and wider intervals, and the use of the ”PERT”-formula. Still problematic?

Activity	Minimum effort (p10)	Most likely (ML) effort	Maximum effort (p90)	Mean effort PERT effort = (Min+4ML+Max)/6)	Variance of effort PERT variance = (Max – Min)²/36
Activity A	15 work-hours	20 work-hours	40 work-hours	23 work-hours	17
Activity B	50 work-hours	60 work-hours	100 work-hours	65 work-hours	69
Activity C	45 work-hours	50 work-hours	150 work-hours	66 work-hours	306
Sum			Expected value =	**154 work-hours**	**392 (stdev = 20)**
Uncertainty	p85 (85% conf. not to exceed) equals ca. exp. value + stdev			**154 + 20 = 174 wh**	

- The assumption of the PERT-formula is the unrealistic assumption that min=p0 and max=p100. Does not affect mean effort much, but the variance get much too small. Should divide variance (assuming p10 as min and p90 as max) by approx. $2.65^2 = 7.0$ instead of 36! PERT gives much too narrow intervals.
- No support for knowing what a p10 and p90 estimate should be (No diff betwen 75%, 80%, 90% and 98% confidence intervals.)

Experiment: Are software developers and managers able to give min-max with X% confidence?

- Participants: 62 professional software developers (from Ukraine)
- Estimated the same task.
- Asked for min-max intervals

Group98%: How accurate do you think your effort estimate is?
I am 98% confident that I will use between _____ (minimum) and _____ (maximum) work-hours.
NB: A 98% confidence means that you think that the actual effort will be within the minimum-maximum interval in about 98 out of 100 times in similar situations.

Group80%: How accurate do you think your effort estimate is?
I am 80% confident that I will use between _____ (minimum) and _____ (maximum) work-hours.
NB: An 80% confidence means that you think that the actual effort will be within the minimum-maximum interval in about 80 out of 100 times in similar situations.

TABLE II: MINIMUM, MAXIMUM AND RWIDTH

Group	Minimum effort (median)	Maximum effort (median)	RWidth (median)
Group98%	110 work-hours	210 work-hours	0.44
Group80%	112 work-hours	176 work-hours	0.40

Rwidth = (Max – Min)/Most likely

16

What to do? A long way to go …

A simple approach leading to more realistic effort uncertainty asessments

1. Estimate the most likely effort of the new project or task.
2. Identify the ”reference class” (similarly estimation complexity of projects or tasks).
3. Recall the estimation error distribution of the reference class.
4. Use the estimation error distribution to find p10, p50 (plan), p80 (budget), p90 or whatever estimate you need.

Example:

- You estimate the most likely effort a new project to be 1000 work-hours and want to find the p90-estimate (which will be your maximum effort).
- In the reference class of similar projects you find/know that 90% of the projects had an effort overrun of 60% of less (= 10% had more than 60% overrun).
- Your p90-estimate should consequently be 1000 + 60% of 1000 = 1600 work-hours.

We have evaluated and implemented this approach in real-world contexts

Experiment

- Nineteen estimation teams of software professionals within one company.
- Estimation of the most likely effort of a project, which had just started.
- Estimation of the uncertainty in terms of 90%-confidence intervals (p5 and p95).
- Two groups:
  - Group A: Uncertainty assessment ”as usual”. Give 90% prediction intervals. No support for minimum and maximum judgements.
  - Group B: Create the error distribution of the reference class. Provide minimum and maximum effort.
- **Results**: The teams in Group B had much more realistic views of the real uncertainty of the project. Especially for the minimum effort, understanding that the world is right-skewed.
- Two replications in real-world contexts (controlled field experiments) confirm the results of improved realism using this method.

20

So what …

- Poor communication of what is meant by effort and benefits estimates is typical in software estimation contexts.
- Poor use of uncertainty assessment methods, if used at all, is even more common.
- Too narrow and too symmetric effort intervals gives ”garbage in – garbage out” even when using proper uncertainty assessment methods.
- Looking back on previous estimation error is a ”simple” and effective way of getting realistic effort prediction intervals.
- This requires compentence and mindsets based on probabilities and distributions.
- A long way to go before the IT industry are able to identify the real uncertainty of software projects...

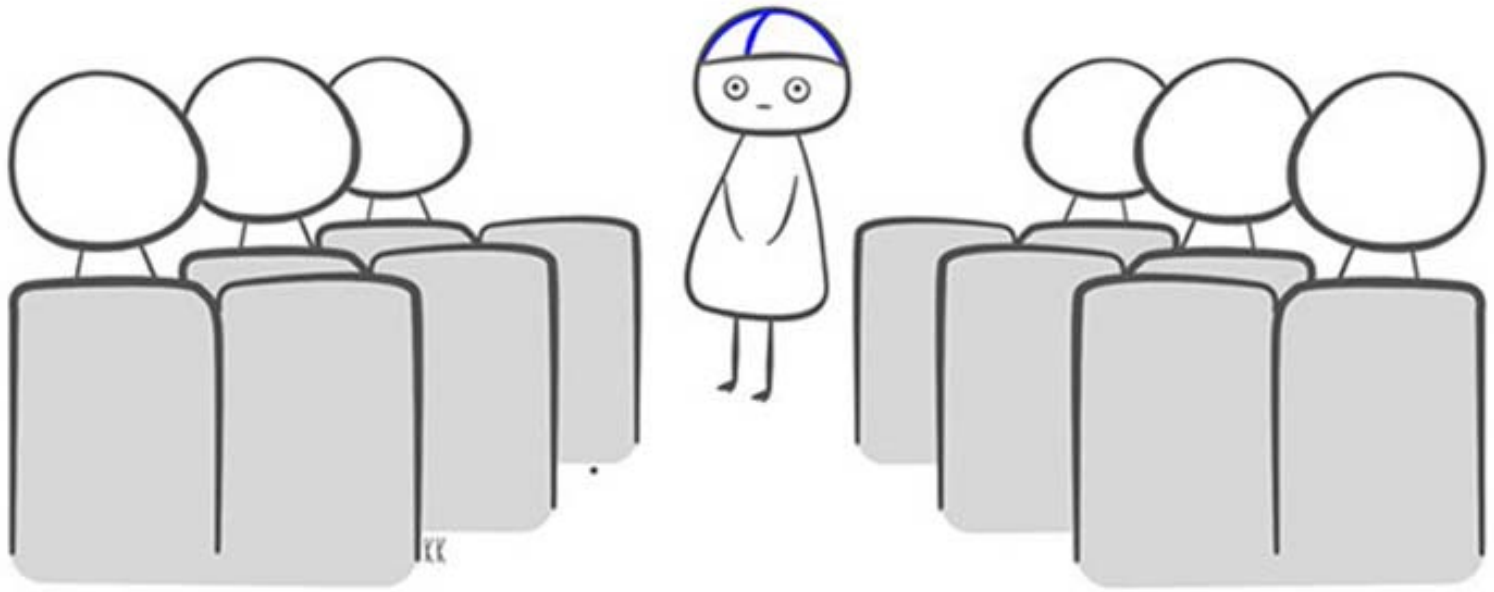
Aalto University
School of Science

Large-Scale Agile? Experiences from the Finnish Telecom Industry

Prof. Casper Lassenius
16.10.2018

FINNISH NIGHTMARES

FINNISHNIGHTMARES.BLOGSPOT.FI - FACEBOOK.COM/FINNISHNIGHTMARES - TWITTER.COM/FINN_MATTI



THE BUS IS "FULL".

Agenda

- Large-Scale Agile
- The cases
- Scaling to many teams
- Scaling the Product Owner
- Dealing with Product Management
- Infrastructure
- Building the Agile Mindset

Large-Scale Agile

- Can mean many things
  - Agile in large organizations
  - Large efforts
    - $$$
    - Long time
    - Many people involved
  - Here: >5 teams, 50 people in the same undertaking
    - Basic planning and coordination mechanisms break down

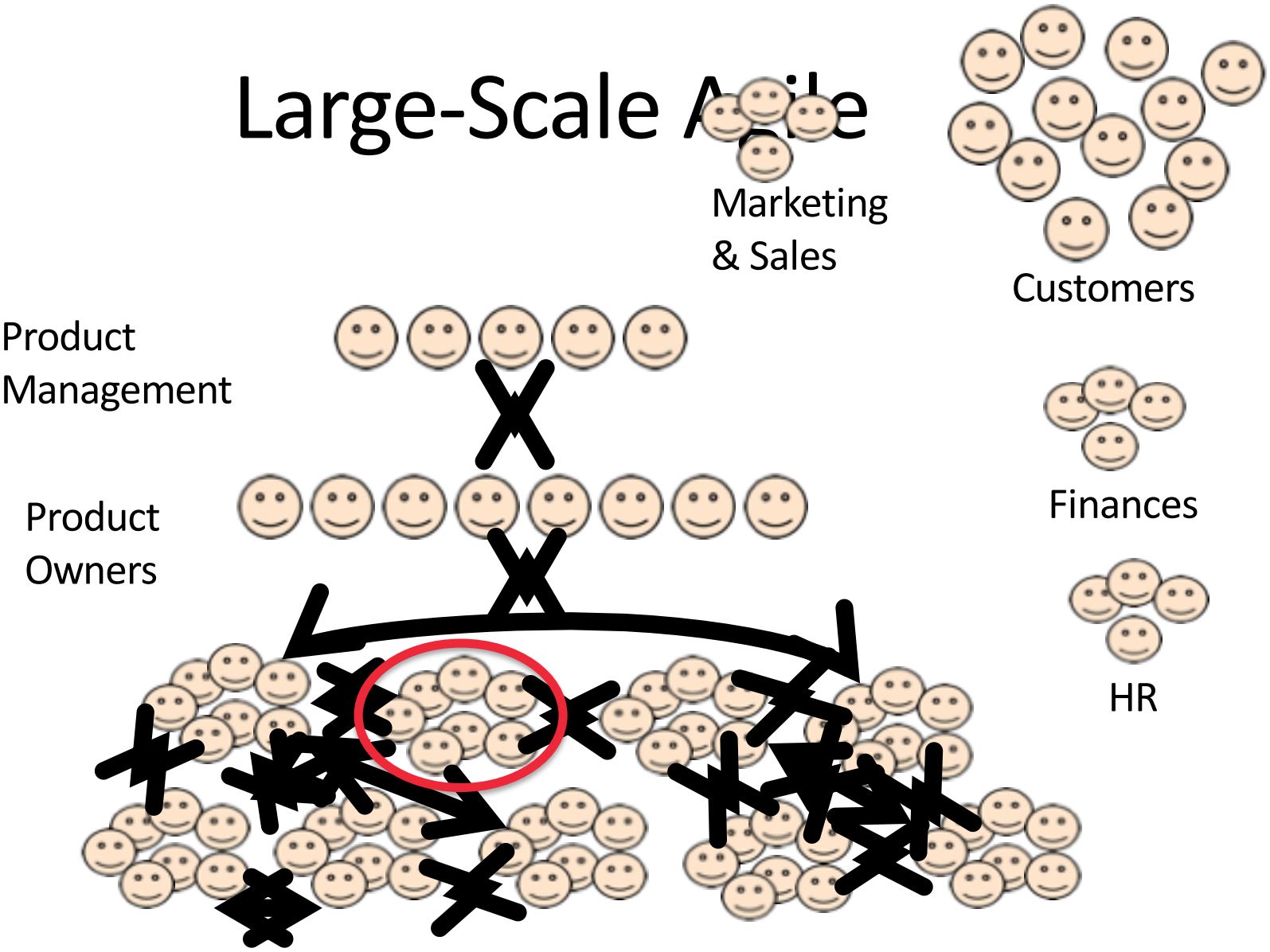
Cases

Organization	Product	# teams	# people	# sites
Ericsson A	Network element	25-40	250-400	3
Ericsson B	Service Platform	5-6	100	3
Nokia A	Network element	20	300	4
Nokia b	Network software	7	200	3
F-Secure	Security software	14	150	1

Some Issues in Large-Scale Agile

- Scaling to many teams
  - Finding a balance between team alignment and autonomy
    - How much freedom can be given to teams, and what must be common?
  - Inter-team coordination
- Sharing knowledge between roles that are now scattered in teams rather than in their own function
  - E.g. embedded architects
- Scaling the Product Owner function
- Dealing with the rest of the (non-agile) organization
- Building the agile mindset

Large-Scale Agile

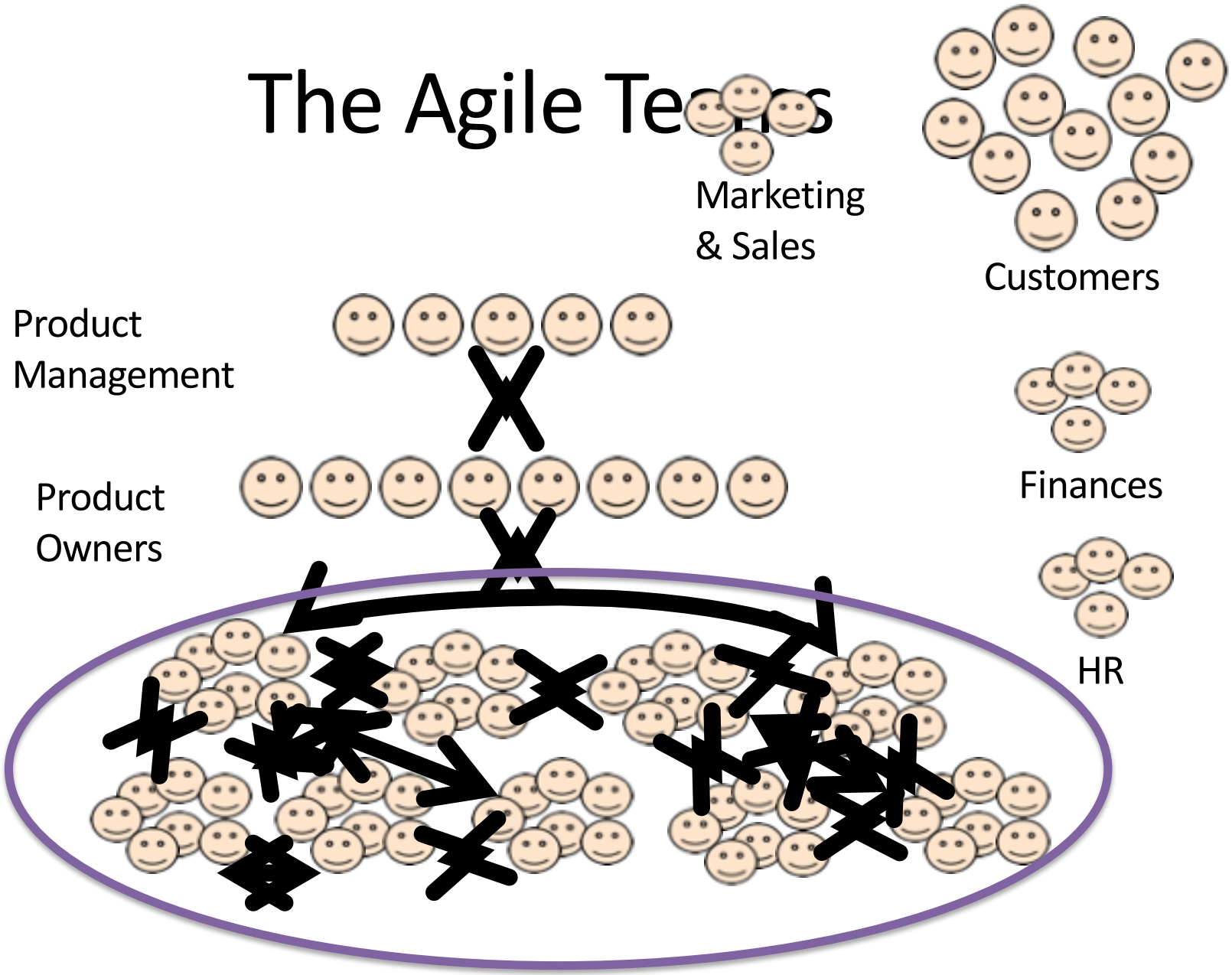


The Agile Team

Considerations:

- Cross-functional
- Self-organizing?
- What can a single team decide?
- What should be common with other teams?
  - Synchronization points with other teams?
- Expert roles in teams?
  - E.g. architects
- Common model to start with
- Needs coaching

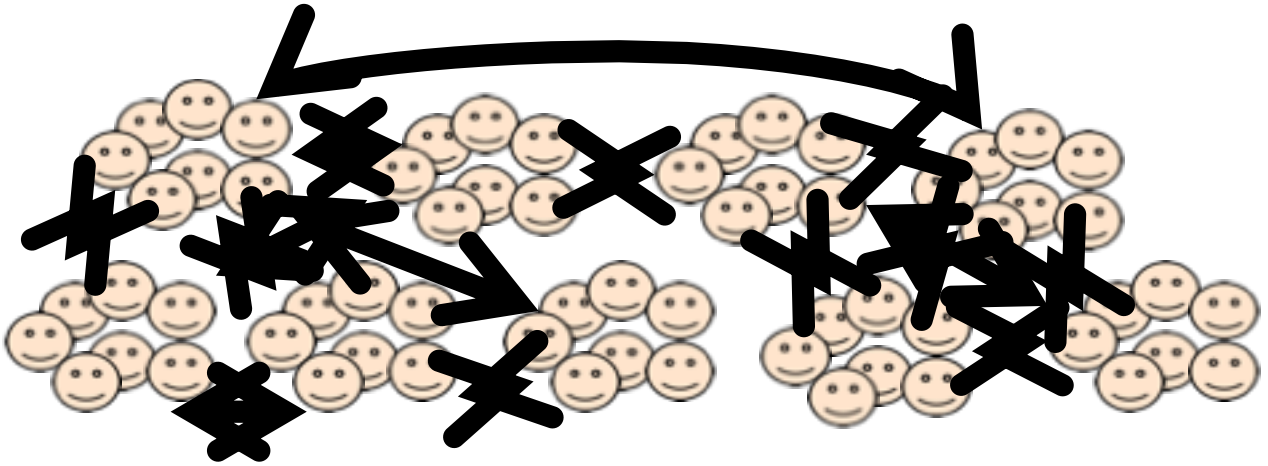
The Agile Teams



Inter-team Coordination

Inter-team coordination:

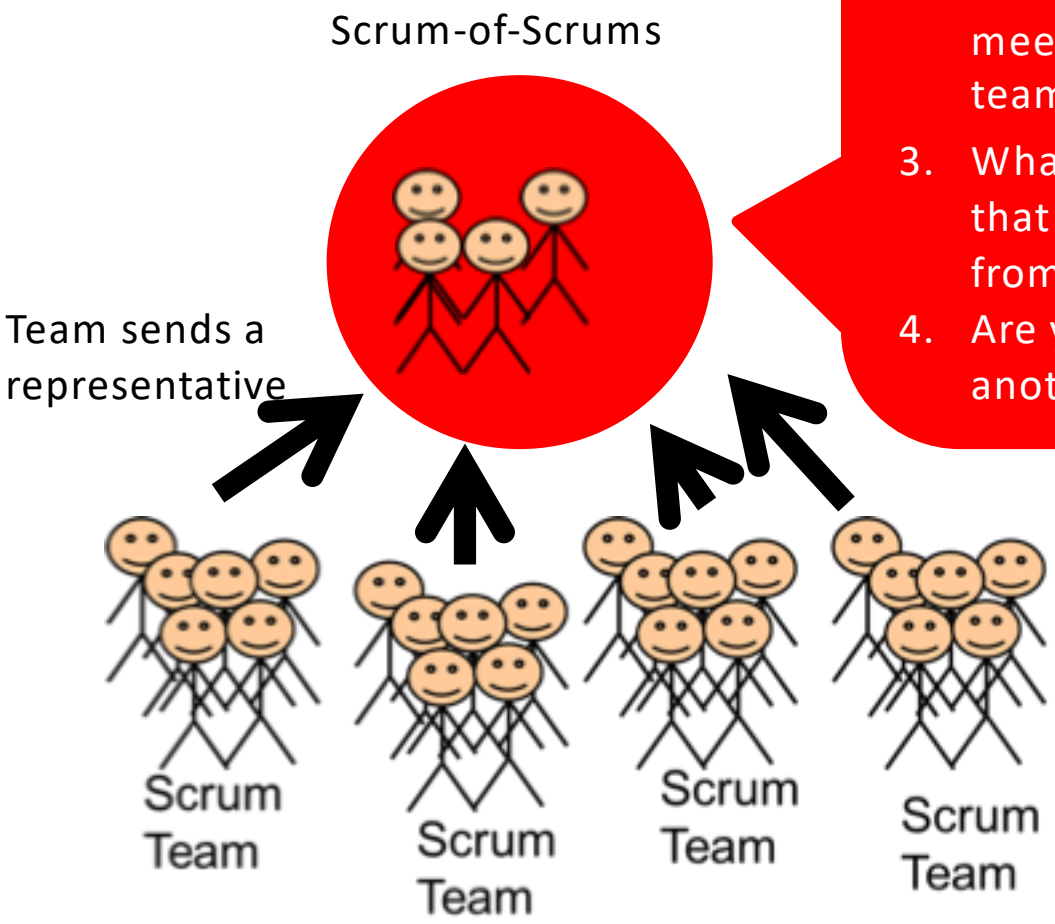
- Team responsibility – a huge mindset change
- Possible structures
  - Scrum-of-Scrums (SoS)
  - Joint Release Planning / PI planning
  - Communities of practice (CoP)
  - Common Demo & Retrospective



Literature: Scrum-of-Scrums (SoS)

SOS meeting:

- 2-5 / week
- 15-30 min

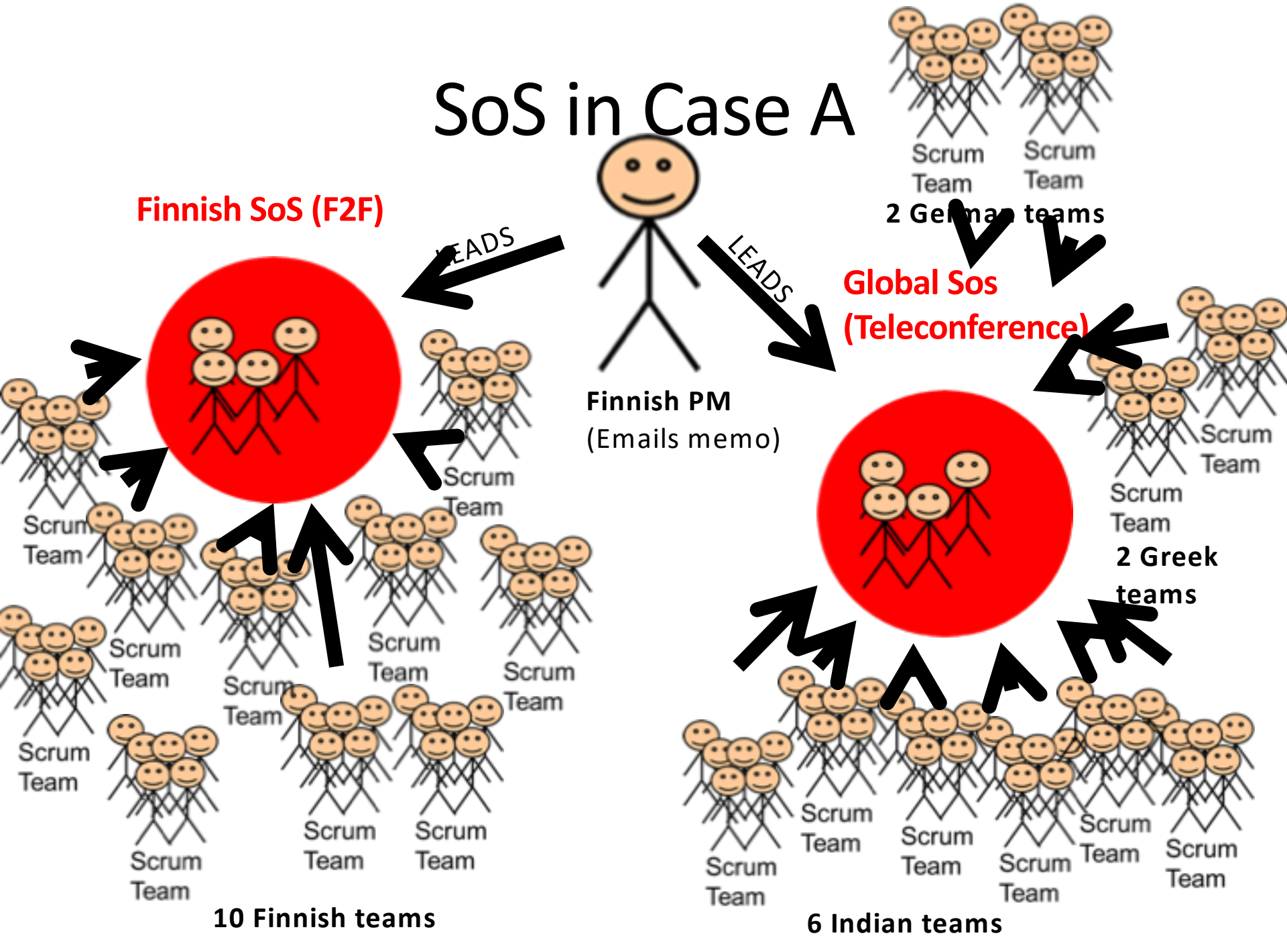


1. What did your team do since the previous meeting that is relevant to some other team?
2. What will your team do by the next meeting that is relevant to other teams?
3. What obstacles does your team have that affect other teams or require help from them?
4. Are you about to put something in another team's way?

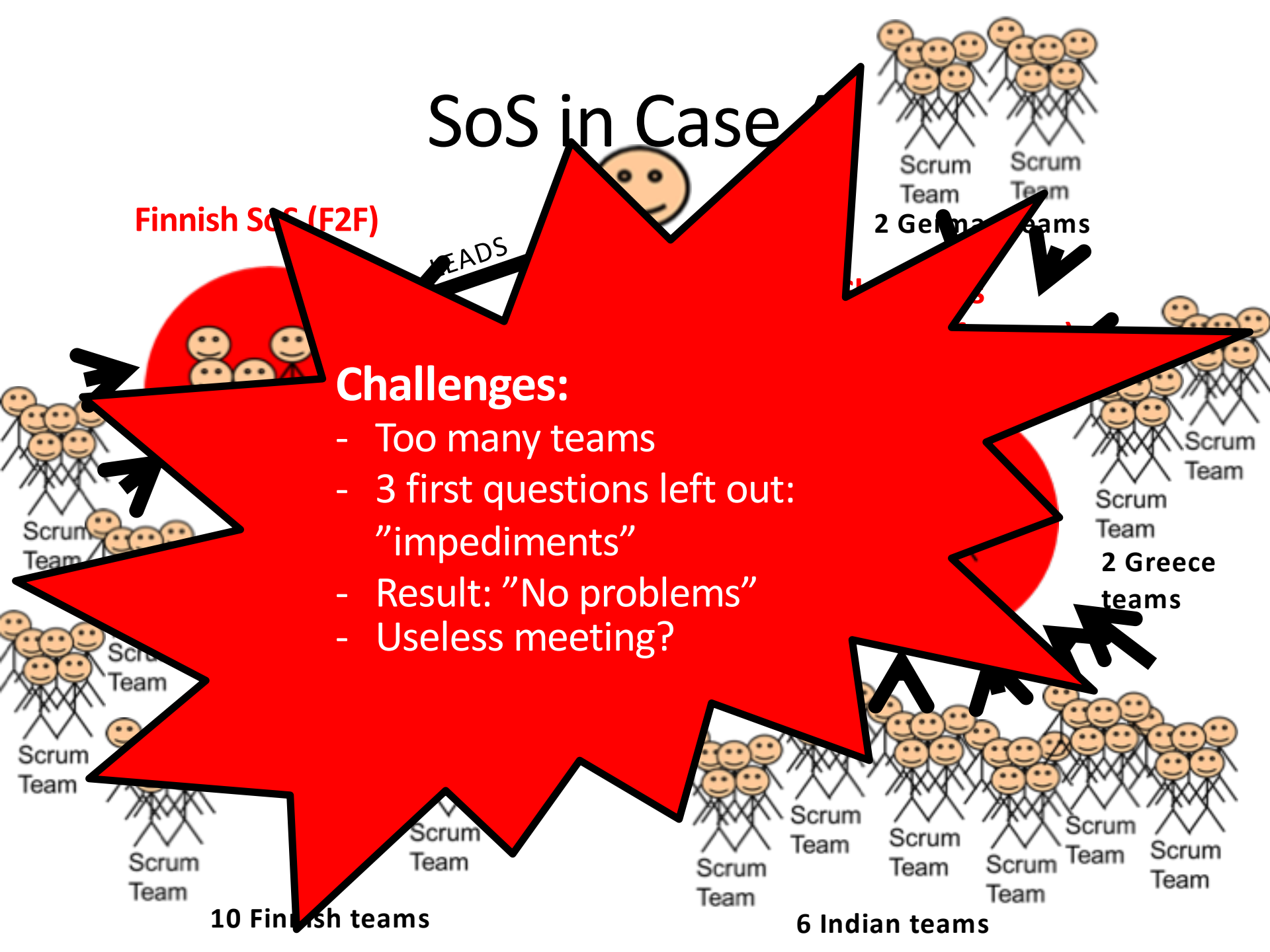
More teams?

Nested Scrum meetings: Scrum-of-Scrum-of-Scrum (SoSoS)

SoS in Case A


Finnish SoS (F2F)
LEADS
LEADS
Finnish PM
(Emails memo)
Global Sos
(Teleconference)
Scrum Team
2 German teams
Scrum Team
2 Greek teams
10 Finnish teams
6 Indian teams

SoS in Case


Finnish SoS (F2F)
LEADS
Scrum Team
Scrum Team
2 German teams
Challenges:
- Too many teams
- 3 first questions left out: ”impediments”
- Result: ”No problems”
- Useless meeting?
Scrum Team
Scrum Team
2 Greece teams
Scrum Team
Scrum Team
Scrum Team
Scrum Team
Scrum Team
Scrum Team
Scrum Team
Scrum Team
Scrum Team
Scrum Team
10 Finnish teams
6 Indian teams

Case A

“. . . it would be good if people would really talk about the problems there. Sometimes it feels like everybody just says ‘No problems’, that everything is going ok, but later on comes up that this and this does not work. (. . .) And many are fighting with the same problem at the same time.”

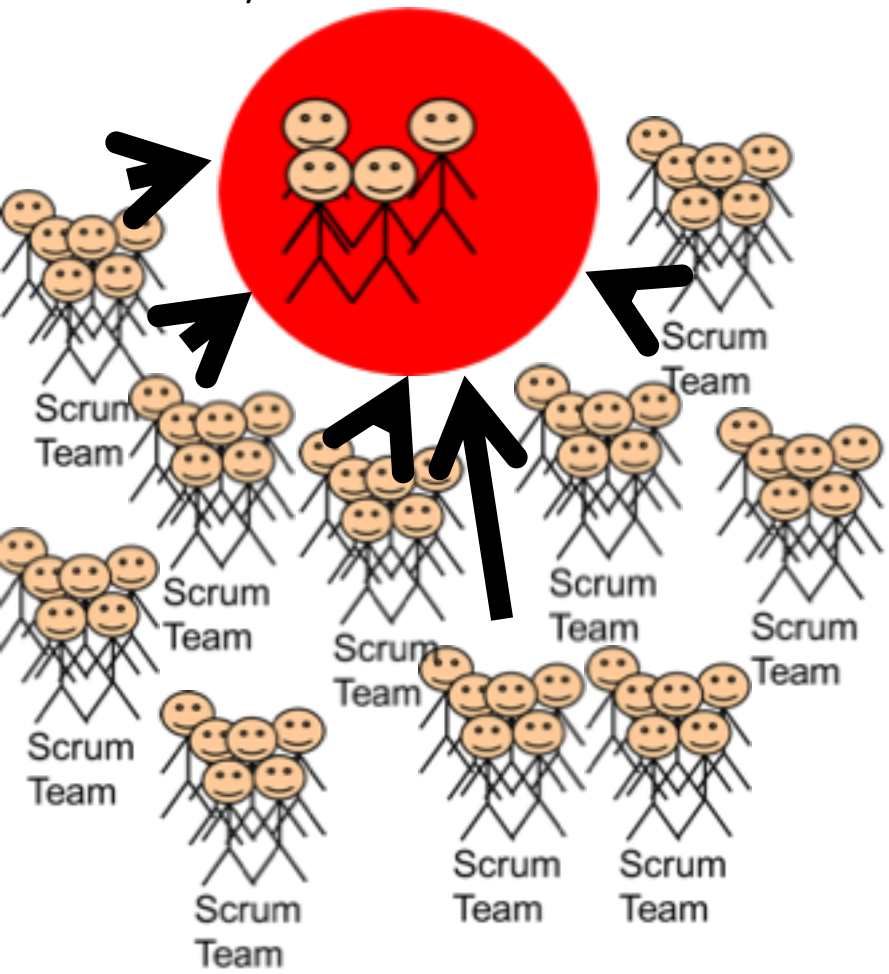
Tester

“Maybe part of the reason is that in general you don’t get solutions there anyway. Now we are scattered around the world, so we don’t have an absolute Scrum of Scrums.”
— Developer

Developer

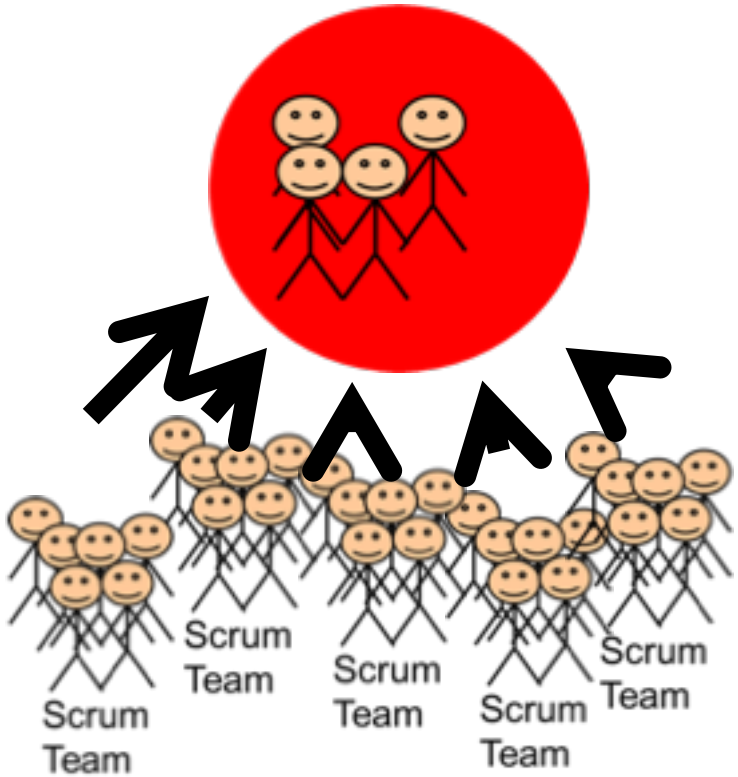
SoS in Case B

Grande SoS
- videoconference
- 1-3 / week



25 teams from 2 sites

Feature SoS
-videoconference
- 1 / week
- separate meetings for each feature



3-5 teams from 1-2 sites

SoS in Cas

Grande SoS

- videoco
- 1-3 / we

ure SoS

oconfe
we
etings for

Challenges:

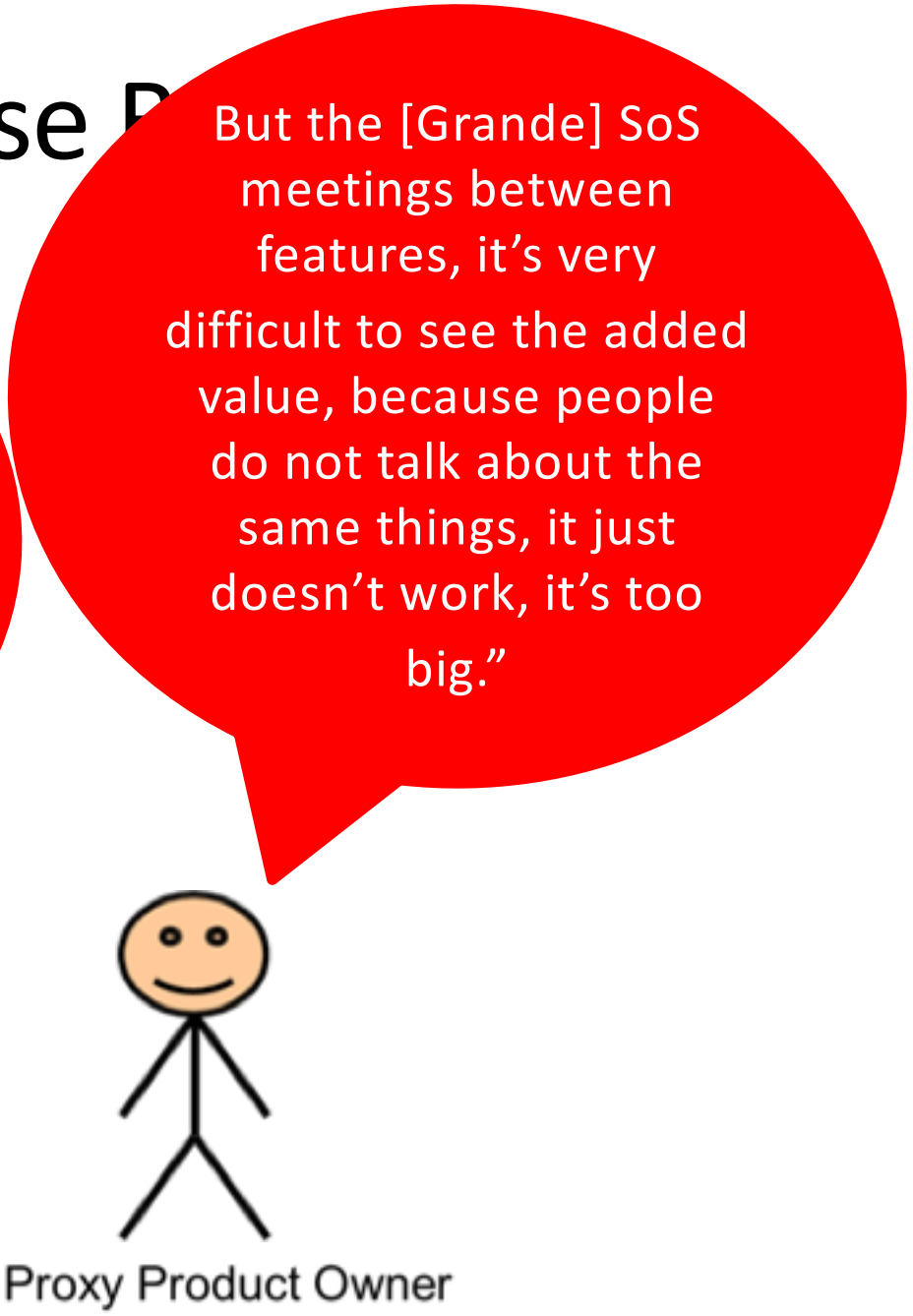
- In Grande SoS teams report ”what they find important to share”
- What to share?
- ”Nothing to share”
- ”Too big”
- ”No added value”
- Feature SoS a solution?

25 te ms from 2 sites

3-5 teams from 1-2 sites

Case P


”Feature SoS meetings are pretty good, because people there do the same things, talk “the same language” and have a common goal.”
Proxy Product Owner


But the [Grande] SoS meetings between features, it’s very difficult to see the added value, because people do not talk about the same things, it just doesn’t work, it’s too big.”
Proxy Product Owner

Community of Practice

a group of people who share a concern, a set of problems, or a passion about a topic, and who deepen their knowledge and expertise in this area by interacting on an ongoing basis

- Often used to solve problems related to organizational boundaries
  - E.g. role-specific CoPs
- Suggested as a mechanism to try in large-scale agile development by the LeSS framework
  - Can be used for a variety of purposes, including role-based concerns (as guilds), and for coordination and planning

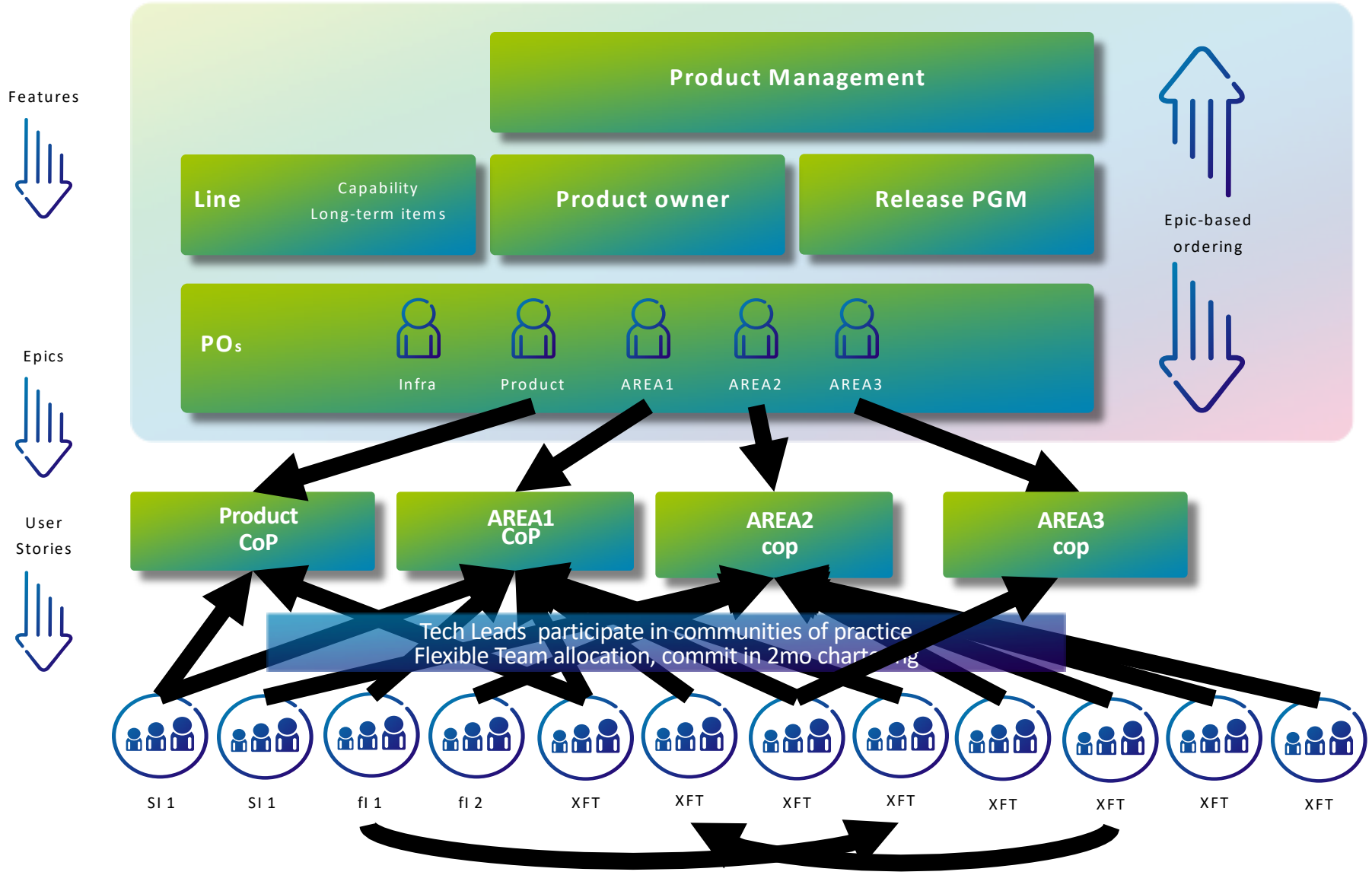
Community-based decision making

- The community (developers, SMs, POs) have decision making authority related to
  - Technical decisions
  - Ways of working
  - Scheduling of work (in collaboration with product management)

- The role of management has changed from monitoring and controlling to supporting the organization

- This is a huge cultural / mindset change compared to the traditional way of working in Ericsson

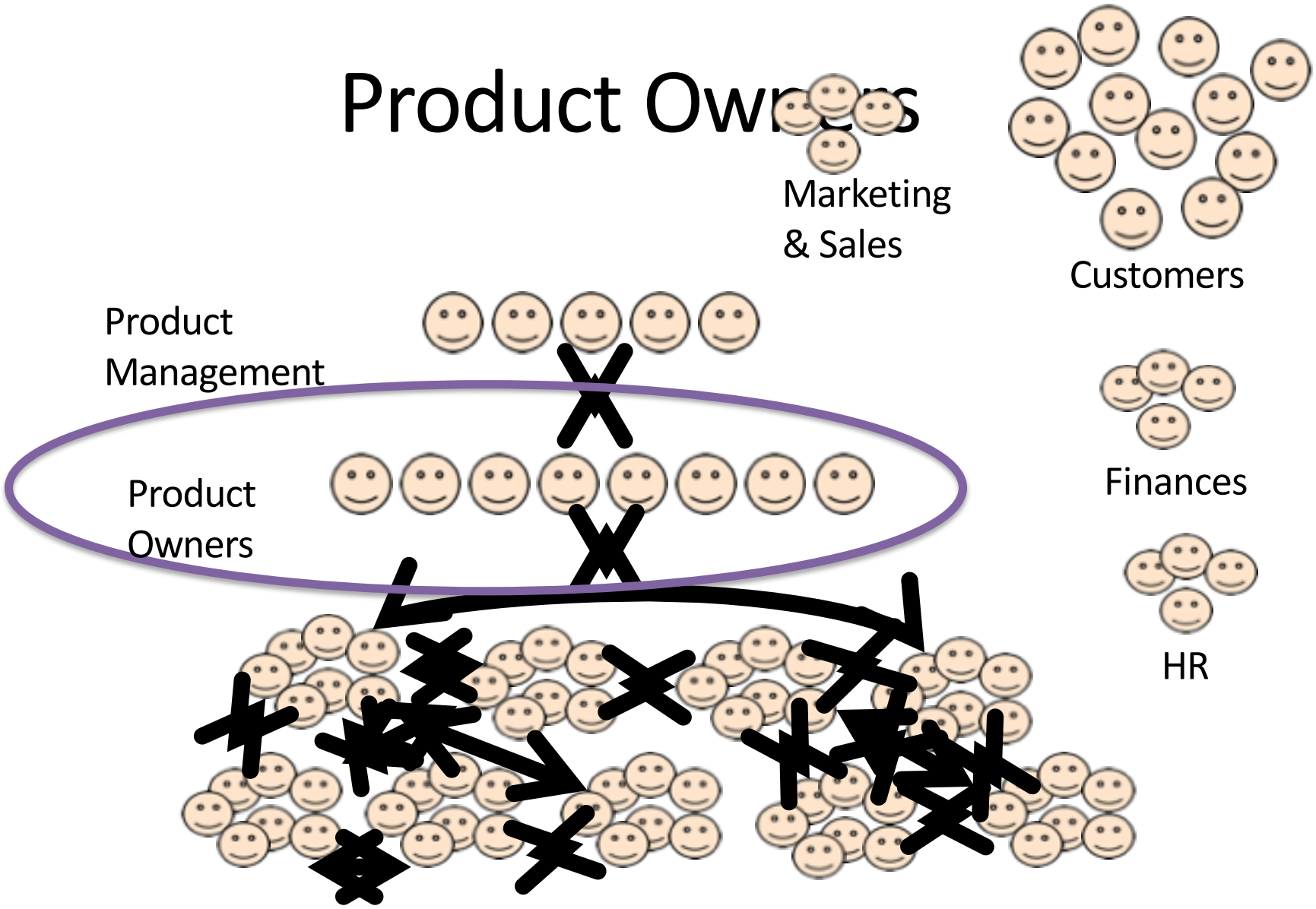
Ericsson: Current STRUCTURE



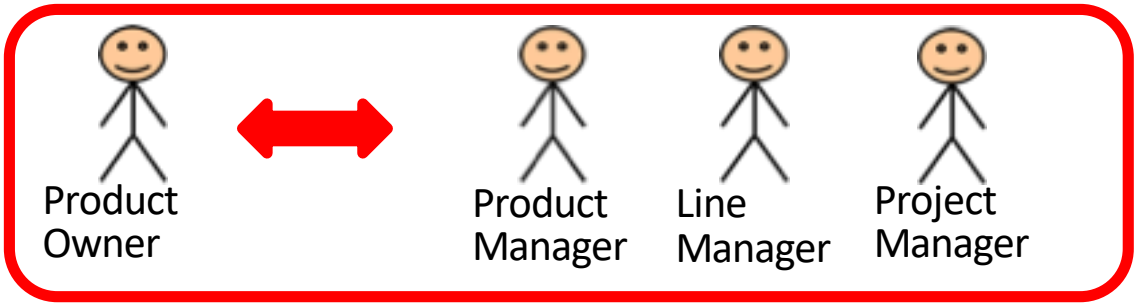
Do’s and Don’ts

- Aim at getting the teams to coordinate amongst themselves
- Use synchronized Sprints if the organization is not mature
- Allow teams autonomy to decide things themselves that do not affect other teams
- Have the teams decide jointly on things that affect them all
- Easily add extra “coordination roles”
- Add extra coordination meetings (have the teams make that decision)

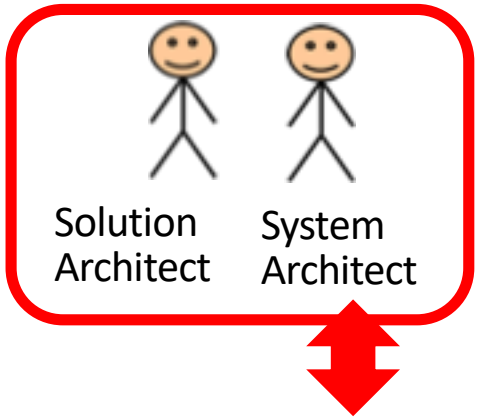
Product Owners



Product Owner Structure in Nokia

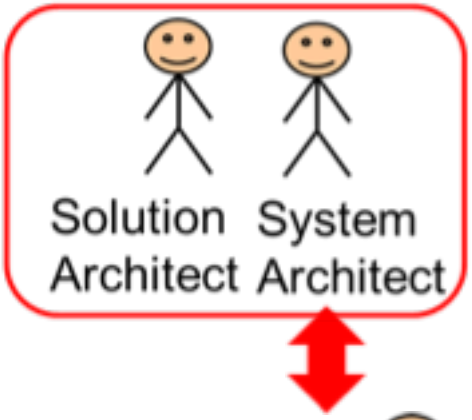


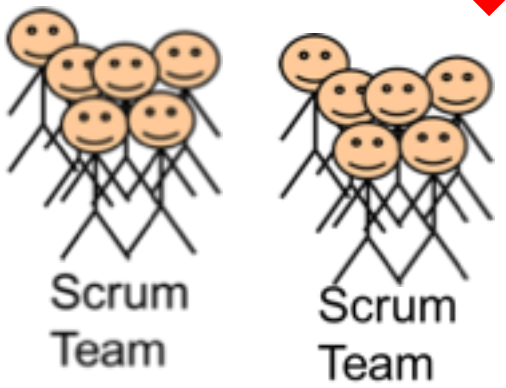
Area Product Owner (APO)



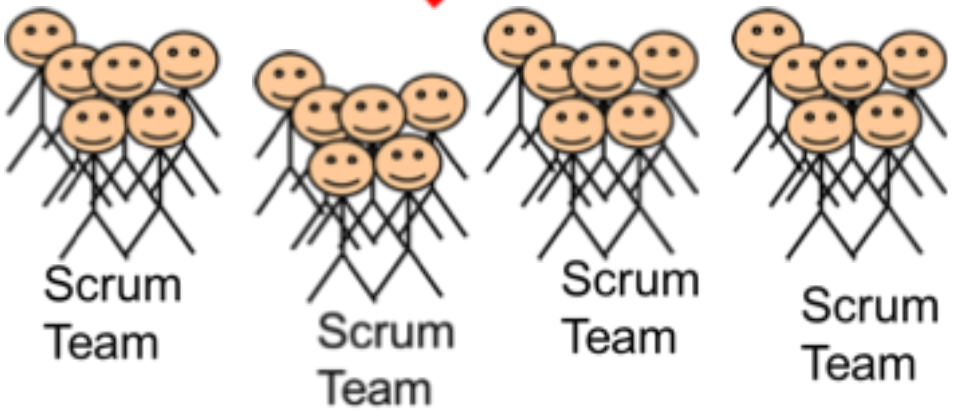
1-n APOs

Area Product Owner (APO)

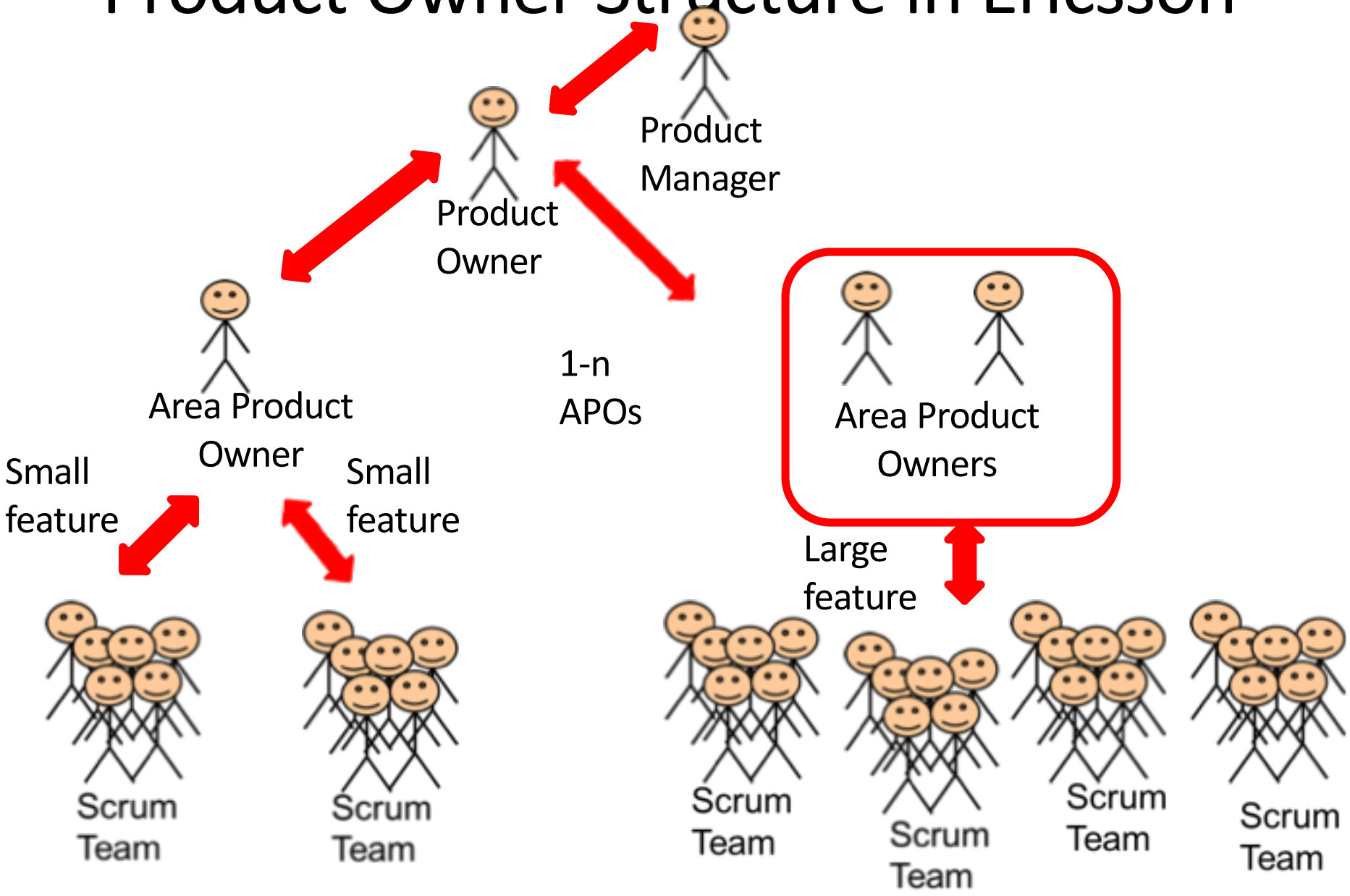




1-n teams per APO



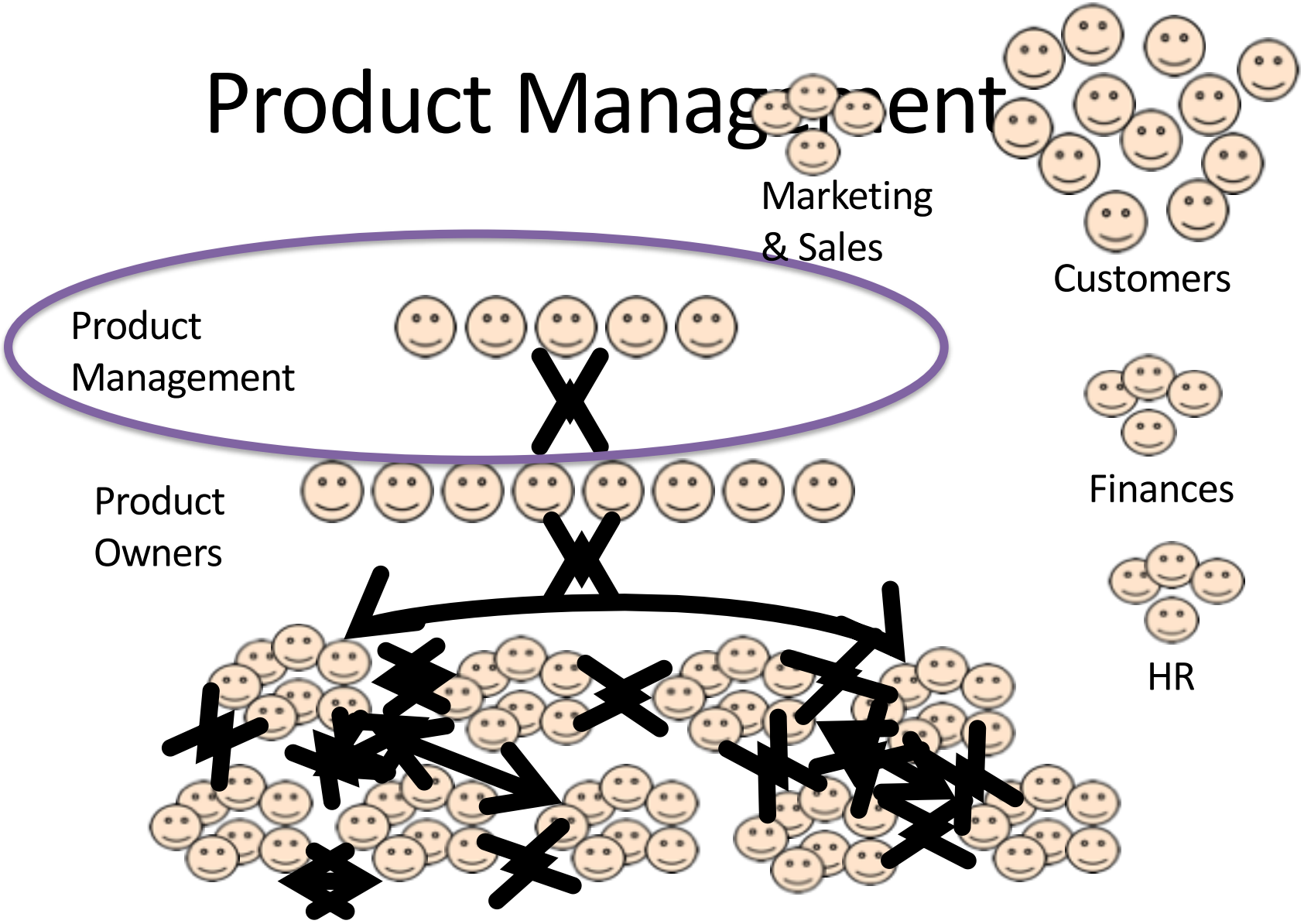
Product Owner Structure in Ericsson



Do's and Don'ts

- Do
  - Consider building a PO team
  - Have one PO work with 1-3 teams
  - Locate the POs close to the teams
  - In a distributed project, PO on the same site as the team if you have site-specific teams
  - Think carefully about the competence needed: technical vs. business
- Don't
  - Split the PO role to be a committee – the PO is always a person

Product Management



Product Management

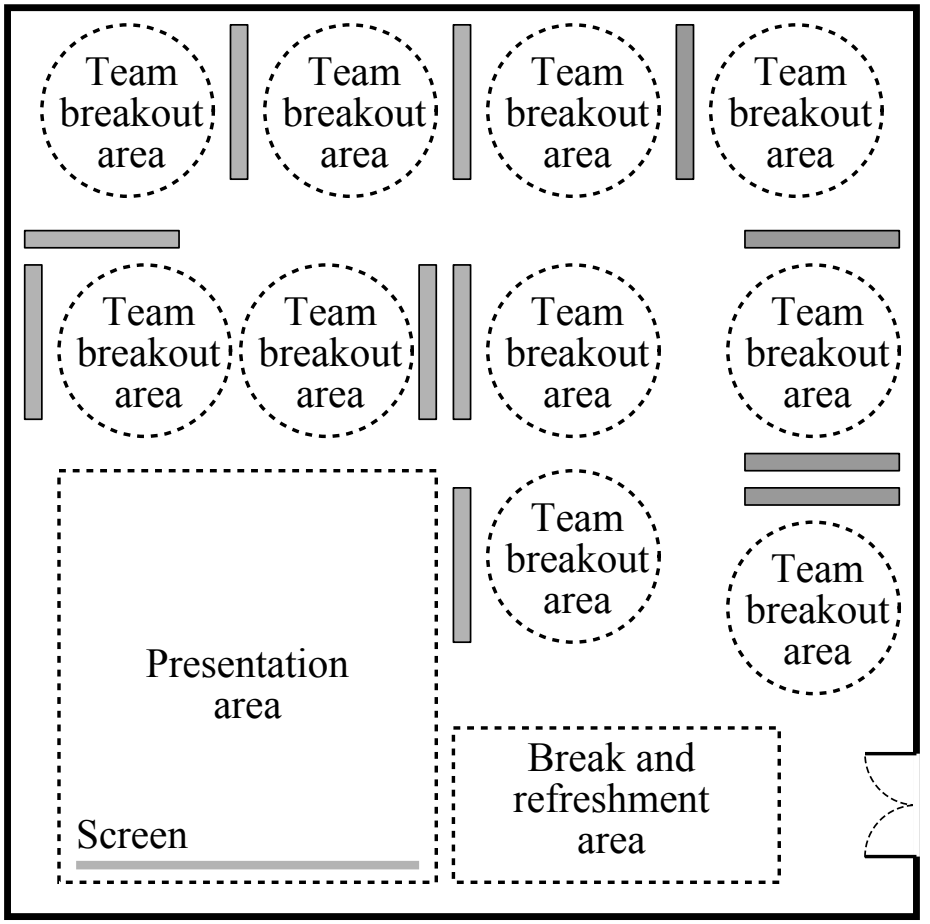
Product Management

Product Management:

- Huge mindset change
  - Cannot continue in waterfall mode
- Releases: When? What? Role of roadmaps?
- How to plan continuously?
- Involve teams in planning
- Lightweight plans

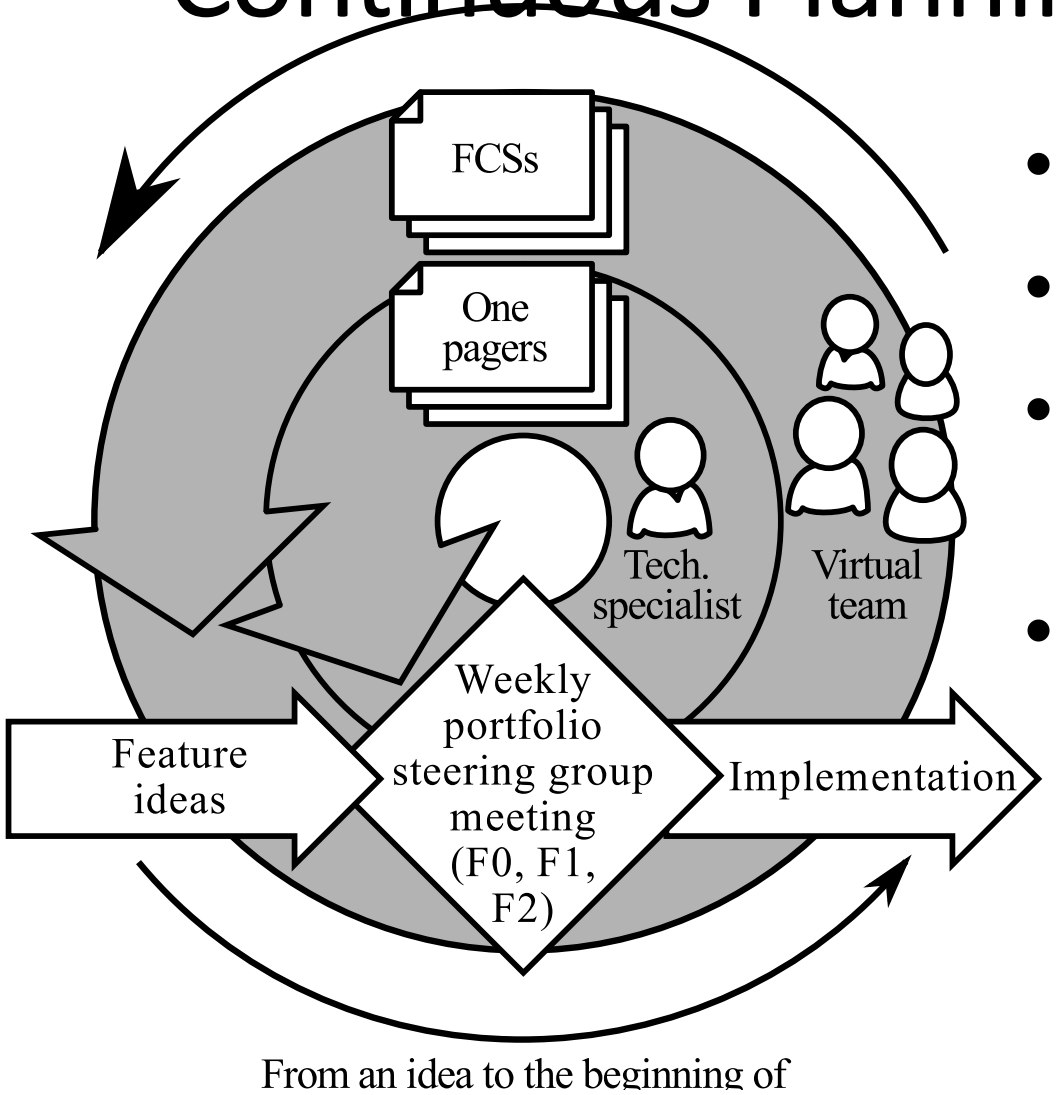
Joint Release Planning at F-Secure

- 10-14 teams
- 2-3 days planning
- 2-4 month release
- Videoconference



Joint Release Planning at F-Secure

Continuous Planning at Ericsson



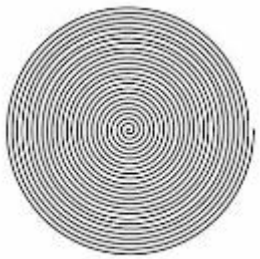
From an idea to the beginning of

- 1-2 week cycle time
- One pager
- FCS= Feature Concept Study
- F0, F1, F2 decision points

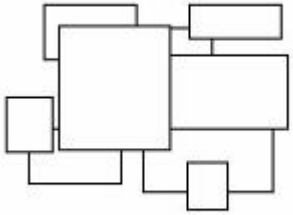
Workspaces and Infrastructure

- Spaces
  - Team spaces
  - Product Owner team
  - Global connections
- Infra
  - Continuous integration
  - Automated tests

Continuous integration



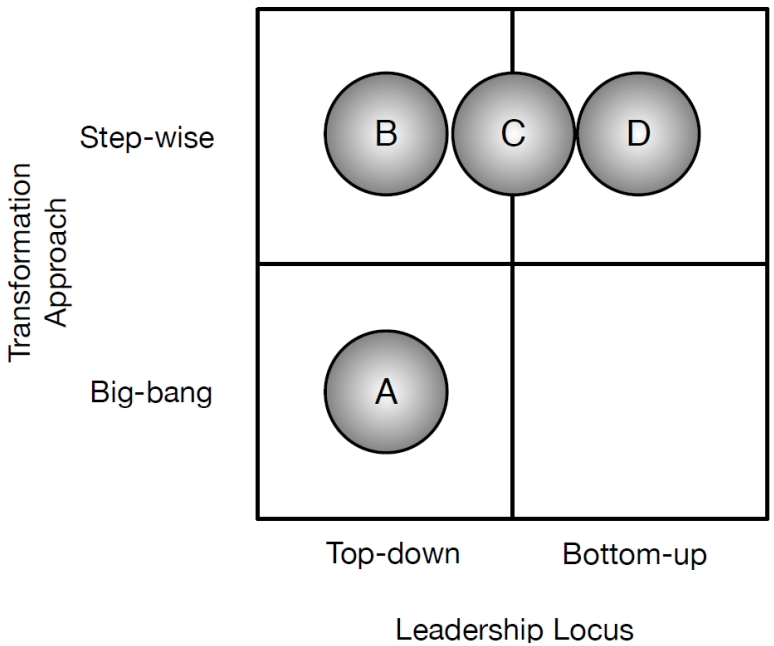
growing



building

Practices for Scaling Lean & Agile Development
Craig Larman and Bas Vodde

Transformation: Be Agile about Going Agile



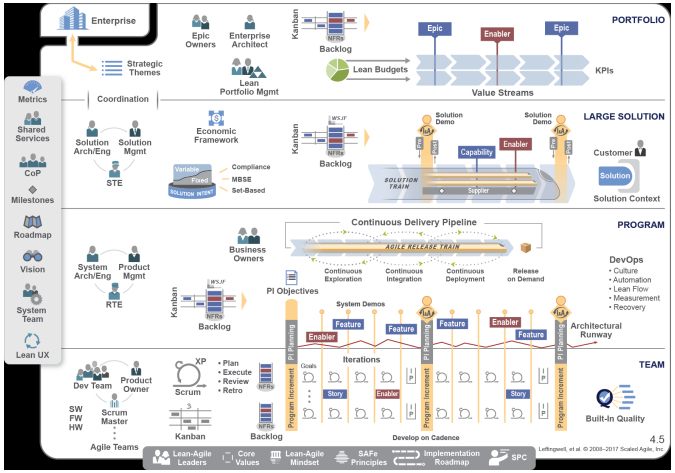
- Use a “sandwich” approach (C)
- One product / product area at a time
- Tailor specifically to each product setting
- Take an experimental approach
  - Try what seems to make sense. If it does not, change it
- Training and coaching
- It is a journey, not a project

Agile is a Mindset

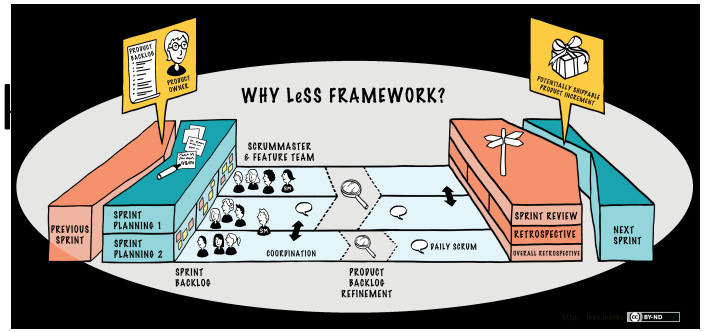
- Agile is not primarily a set of practices or processes
- Agile is a mindset, a new way of thinking about software development, customer collaboration, and management
- Experimental, empirical way of working rather than planning, controlling to stick to the plan
  - Relates both to product development and
  - Organizational development

Agile Frameworks

- Scrum most often used “process”
  - Tailor carefully



- Scaling frameworks
  - LeSS (Large Scale Scrum)
  - SAFe (Scaling Agile Framework)
  - Use with care
    - No replacement for actually “getting” agile



Summary

- Agile implementation at scale is a journey, not a project
- Mindset is the most important aspect – to build an experimental mindset both for agile implementation and product development
- Use frameworks wisely – as a source of ideas, beware of by-the-book implementation

Contact

Casper Lassenius

casper@simula.no

Ericsson Finland R&D

- 40 years of R&D experience
- 400 R&D professionals
- 100 patents filed every year

- Main responsibility areas
  - Media Resource Functions
  - Device Connectivity Platform
  - Radio Network Transmission pro
  - Network Security Solutions
  - Research

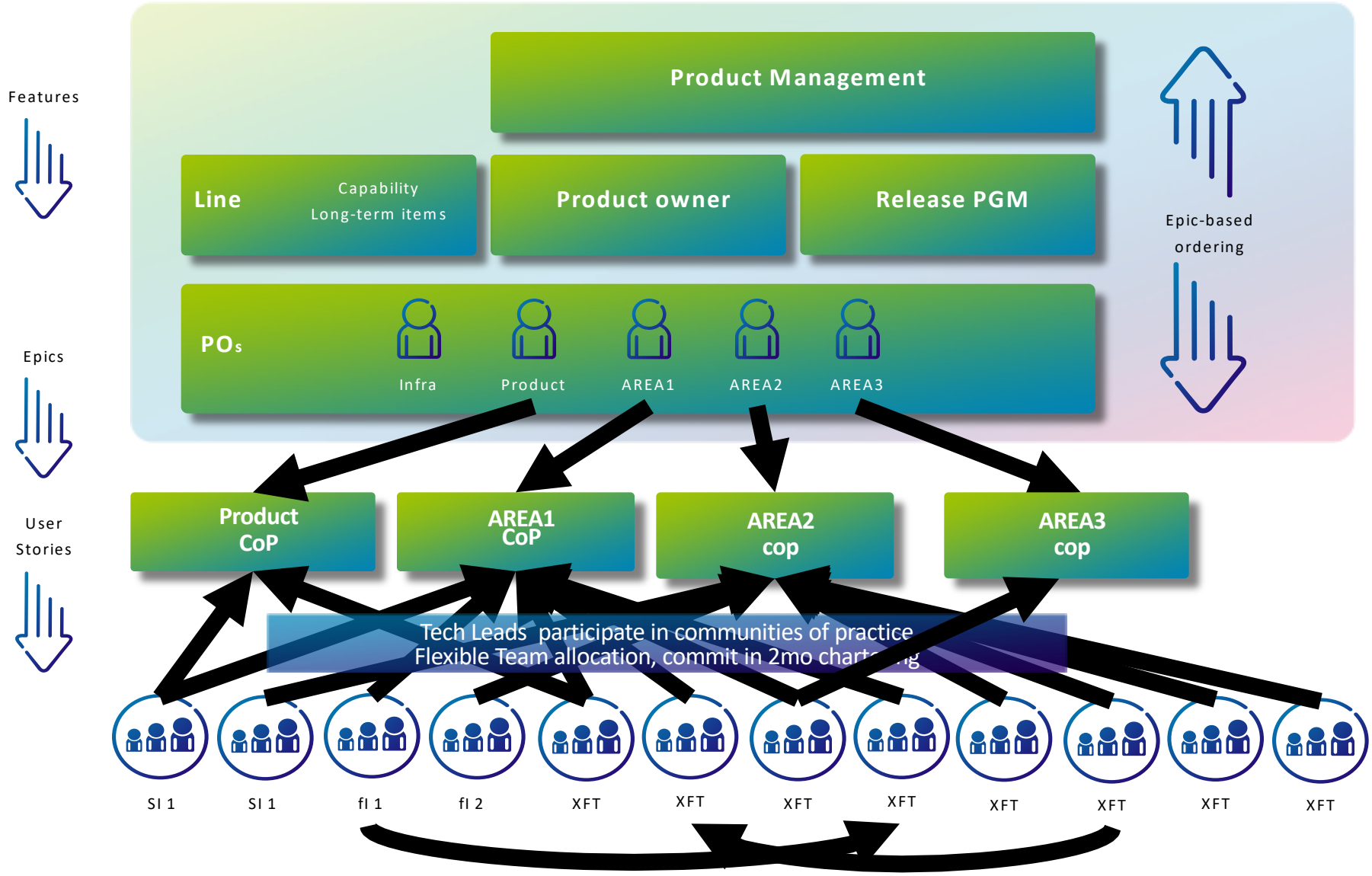
- Forerunner within Ericsson in large scale agile & lean adoptio

Big Changes

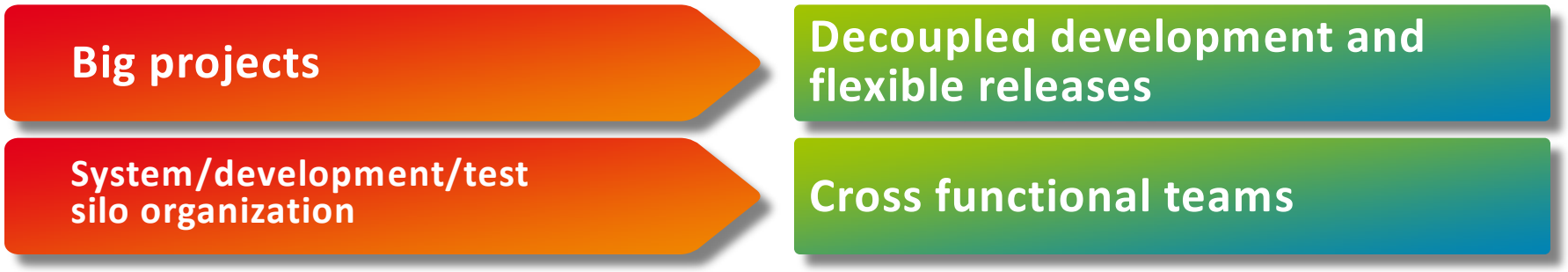
Big projects

Decoupled development and flexible releases

Current STRUCTURE

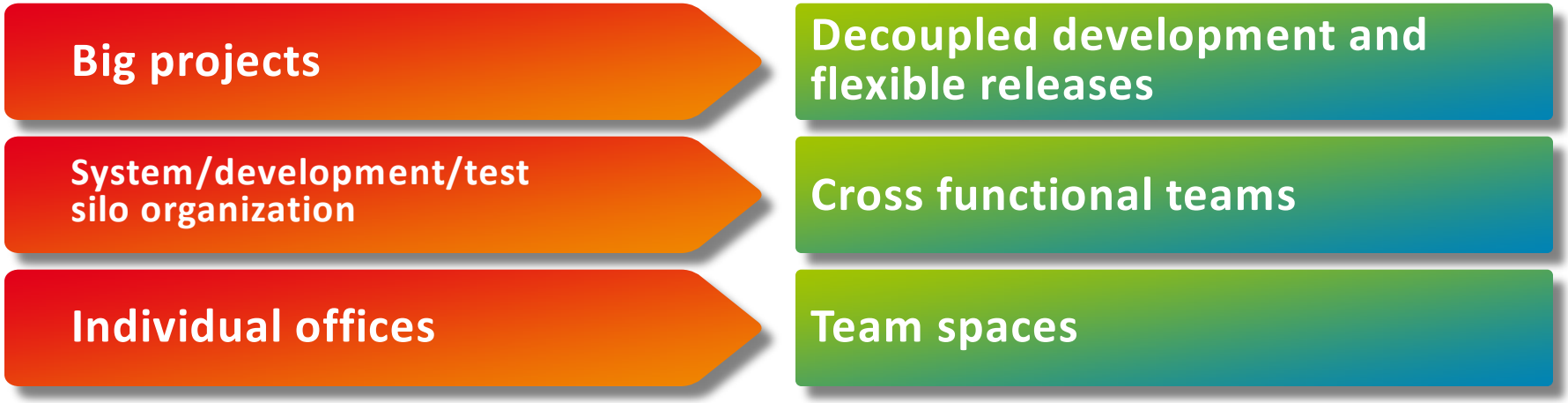


More Big Changes



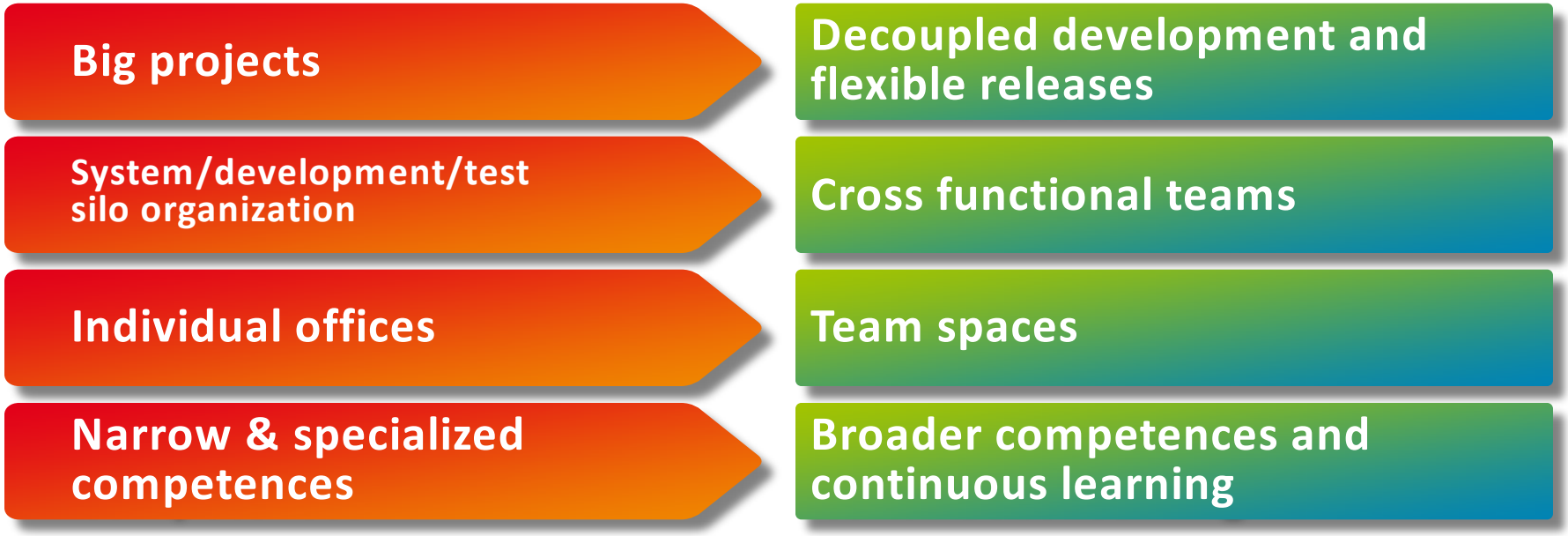
Team Forming session

Visible Changes

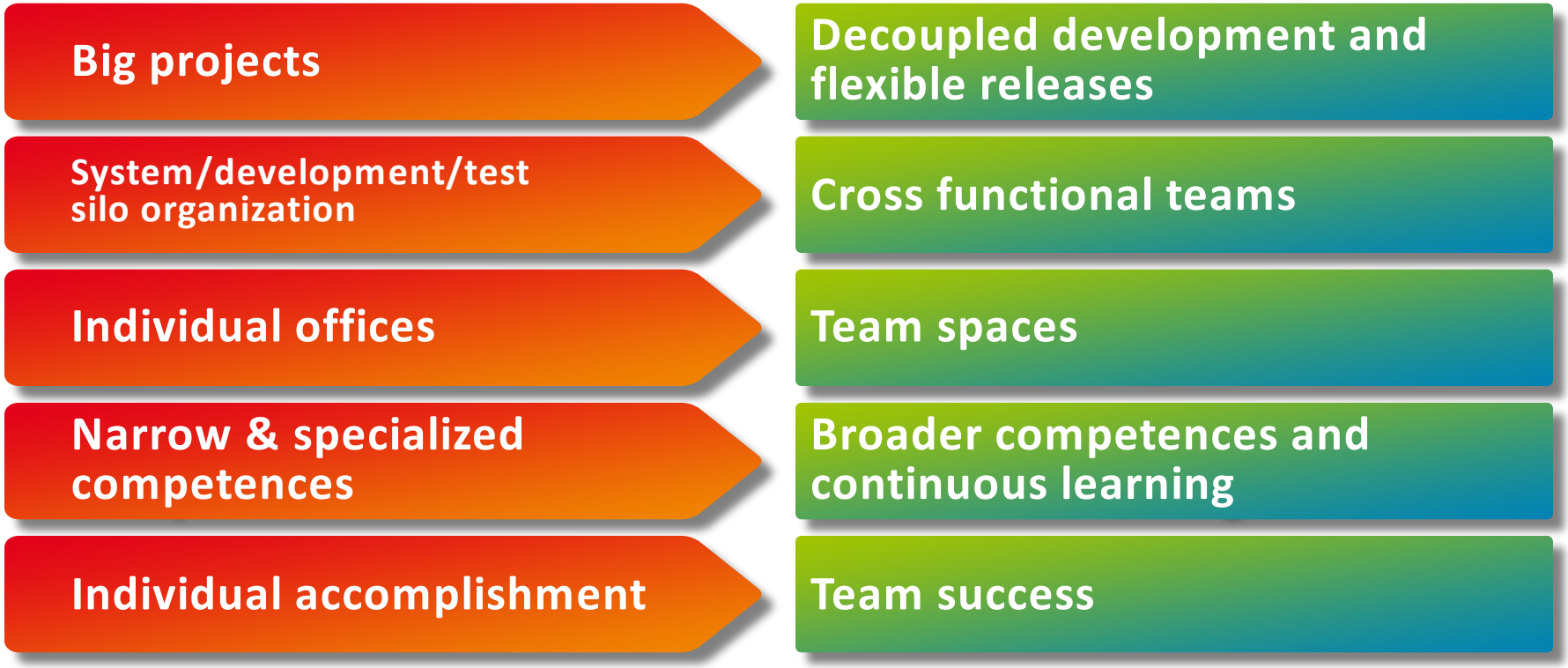


The team space

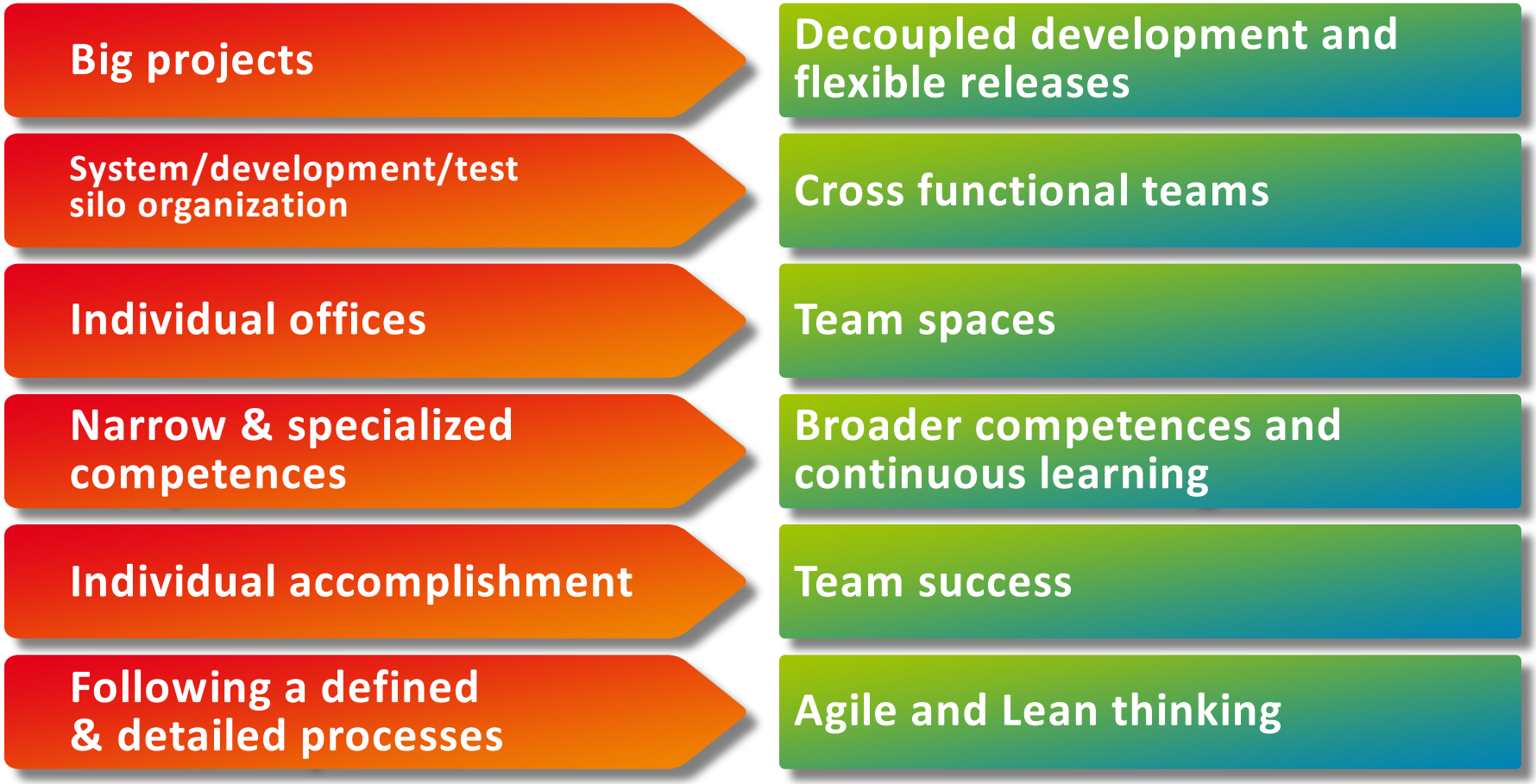
Changing learning



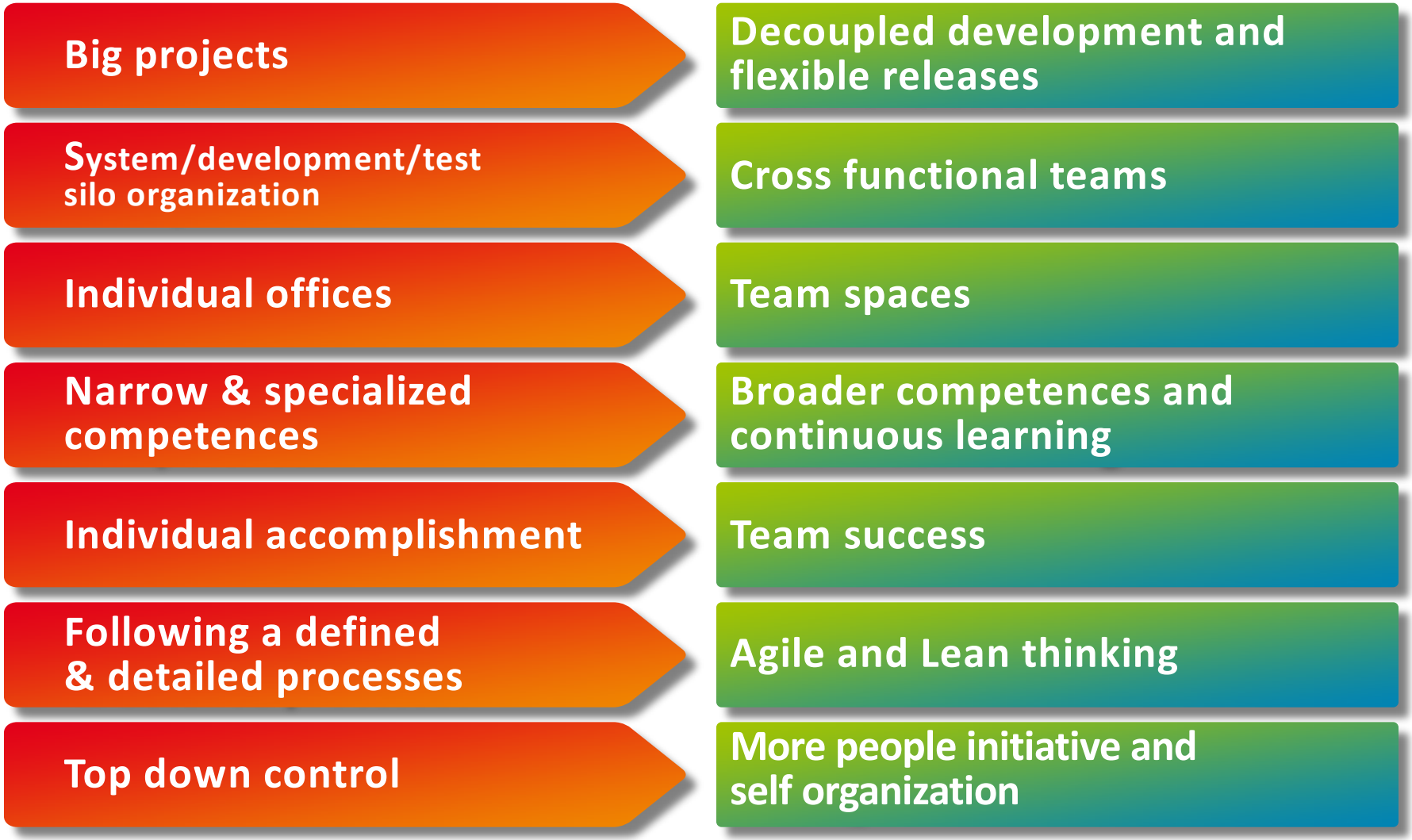
Changing culture



Changing thinking



Changing leadership



Large scale and Distributed Agile & Lean

"Hard aspects" of SW development
Languages, tools, techniques
architectures, modeling, processes, etc.

"Soft aspects" of SW development
Interaction between people,
engagement, team dynamics,
creativity, self-organization, etc.

Case Organization - Comptel

- A Finland based global telecom company - cares for more than 20% of all mobile usage data in the world
- Customers: 300 service providers across 90 countries serving 2 billion end-customers
- Recently acquired by Nokia
- 2 business lines developing 4 prod

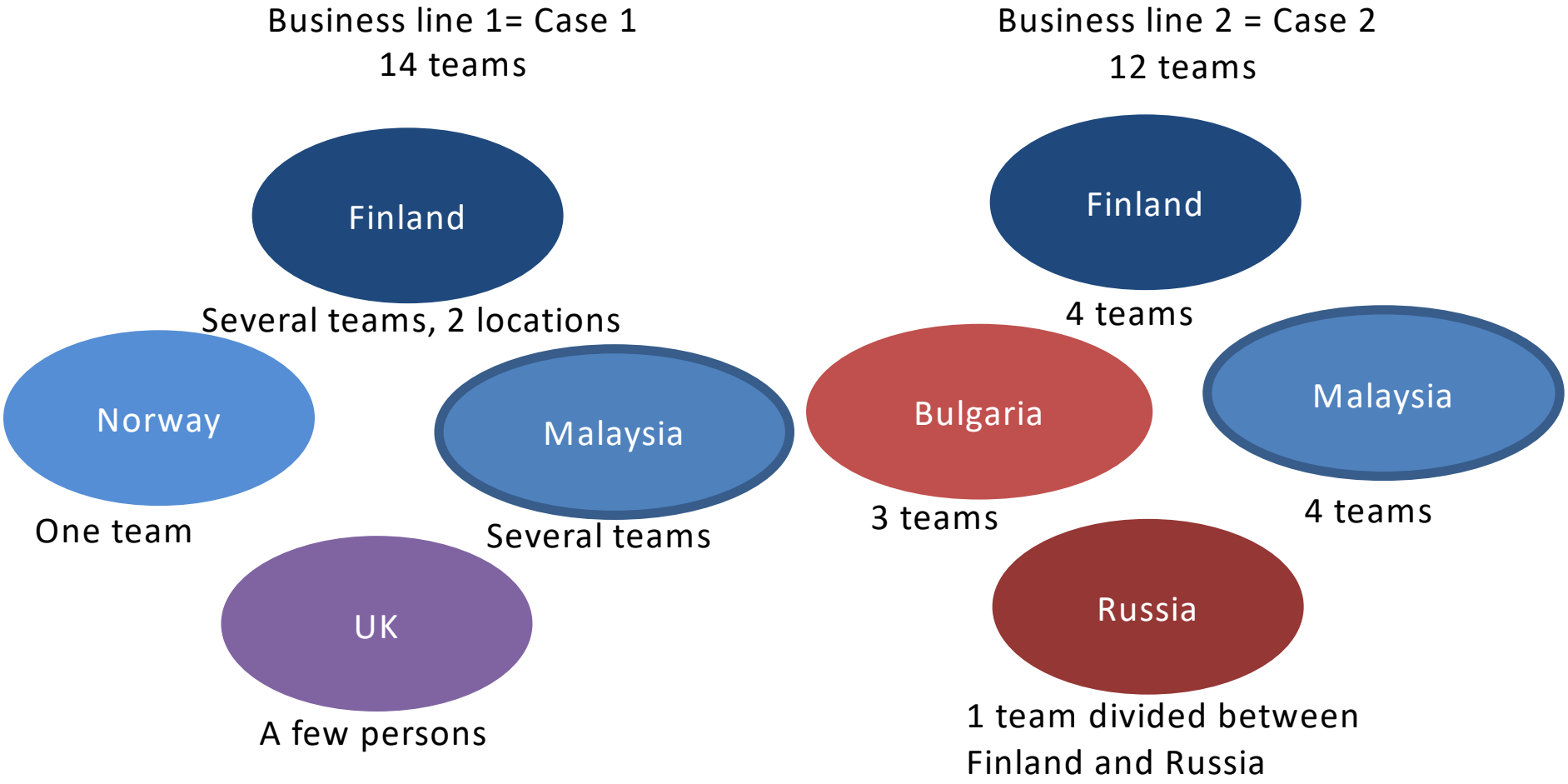
SW Development processes:

- Initially waterfall
- 2008 Scrum
- 2015 SAFe

Why SAFe?

- Inter-team collaboration
- Remove product management silos
- High-level prioritization
- Faster reaction to market changes

Global Distribution of Comptel Software Development



Methodology

RQ: How did the SAFe adoption differ between the business lines?

Data collection: 11 interviews (fall 2016), 1-2 hours each

Role	Case 1 (Finland + Malaysia)	Case 2 (Finland + Malaysia)
SM/team member	2	1
Product Owner	1*	1*
RTE	0 + 1	1
Managers	2	2 + 1
Total	**6**	**6**

* A person from platform organization supporting both business lines

The SAFe Adc

- First major change: Program Increment (PI) planning
  - 10 week increments
  - 2 day events
  - Skype-for-Business between sites
- Closer communication and collaboration
  - Between development teams: Scrum-of-Scrum meetings
  - Between POs and PMs
  - Between PMs
  - PO community meetings
- RTE facilitates

Day 1 - Agenda
Breakfast
Business vision
Architecture vision
Planning in teams
Scrum-of-Scrums
Planning in teams

Day 2 - Agenda
Breakfast
Planning in teams
Plans presented
Confidence vote
Retrospective

	Case 1	Case 2
SAFe training	After problems emerged	Managers on SAFe courses, developers internally
Change resistance & engaging people	Lack of training and communication -> change resistance	Trainings, communication, successful PI planning
Change agents	Few internal, leading change part-time	Several internal, external coach, RTE leading change
External coaches	Not in the beginning	Supported from the start
Release train engineers	Part-time RTE (Malaysia)	Full-time RTE (Finland)
1st PI planning event	Light preparation, chaotic event	Intense preparation, RTE supported by a coach, successful event
Continuous improvement	Items found in retros, but not much reacted	Items collected, responsibilities given, followed up by RTE
Satisfaction	Work satisfaction lowered after SAFe	70% of team members satisfied with SAFe in fall 2016

7 Success Factors for SAFe Adoption

1) Training the personnel well in advance

2) Informing and engaging people

3) Involving change agents

4) Hiring an experienced external consultant to train, advice and support

5) Preparing well for the first PI planning event

6) Having a full-time RTE

7 Success Factors for SAFe Adoption

1) Training the personnel well in advance

2) Informing and engaging people

3) Involving change agents

4) Hiring an experienced external consultant to train, advice and support

5) Preparing well for the first PI planning event

6) Having a full-time RTE

Success factors of large-scale agile transformations (SLR by Dikert et al. 2016)

7 Success Factors for SAFe Adoption

1) Training the personnel well in advance

2) Informing and engaging people

3) Involving change agents

4) Hiring an experienced external consultant to train, advice and support

5) Preparing well for the first PI planning event

6) Having a full-time RTE

7 Success Factors for SAFe Adoption

1) Training the personnel well in advance

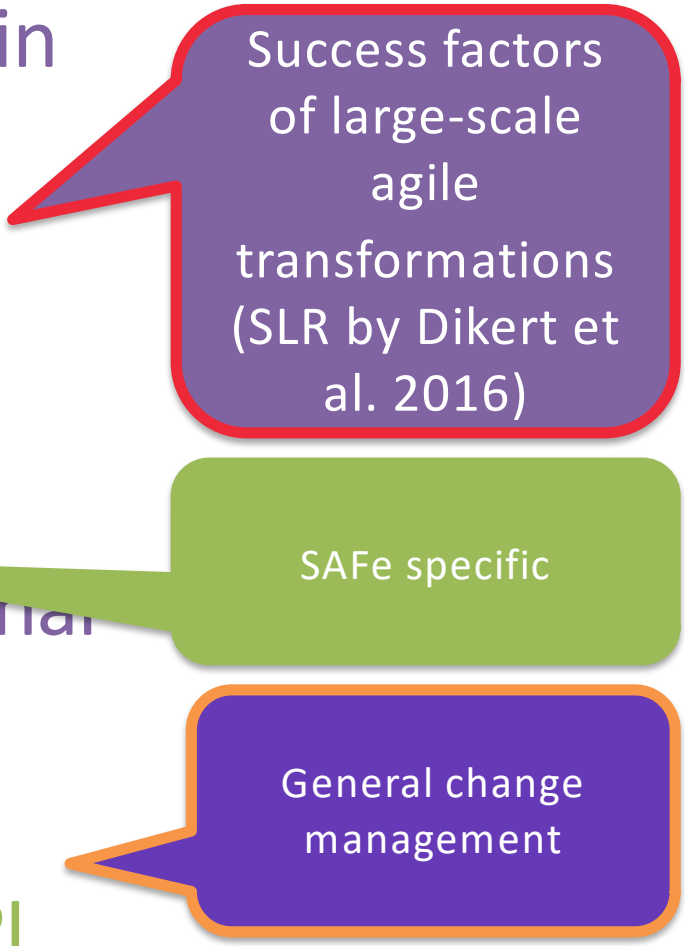
2) Informing and engaging people

3) Involving change agents

4) Hiring an experienced external consultant to train, advice and support

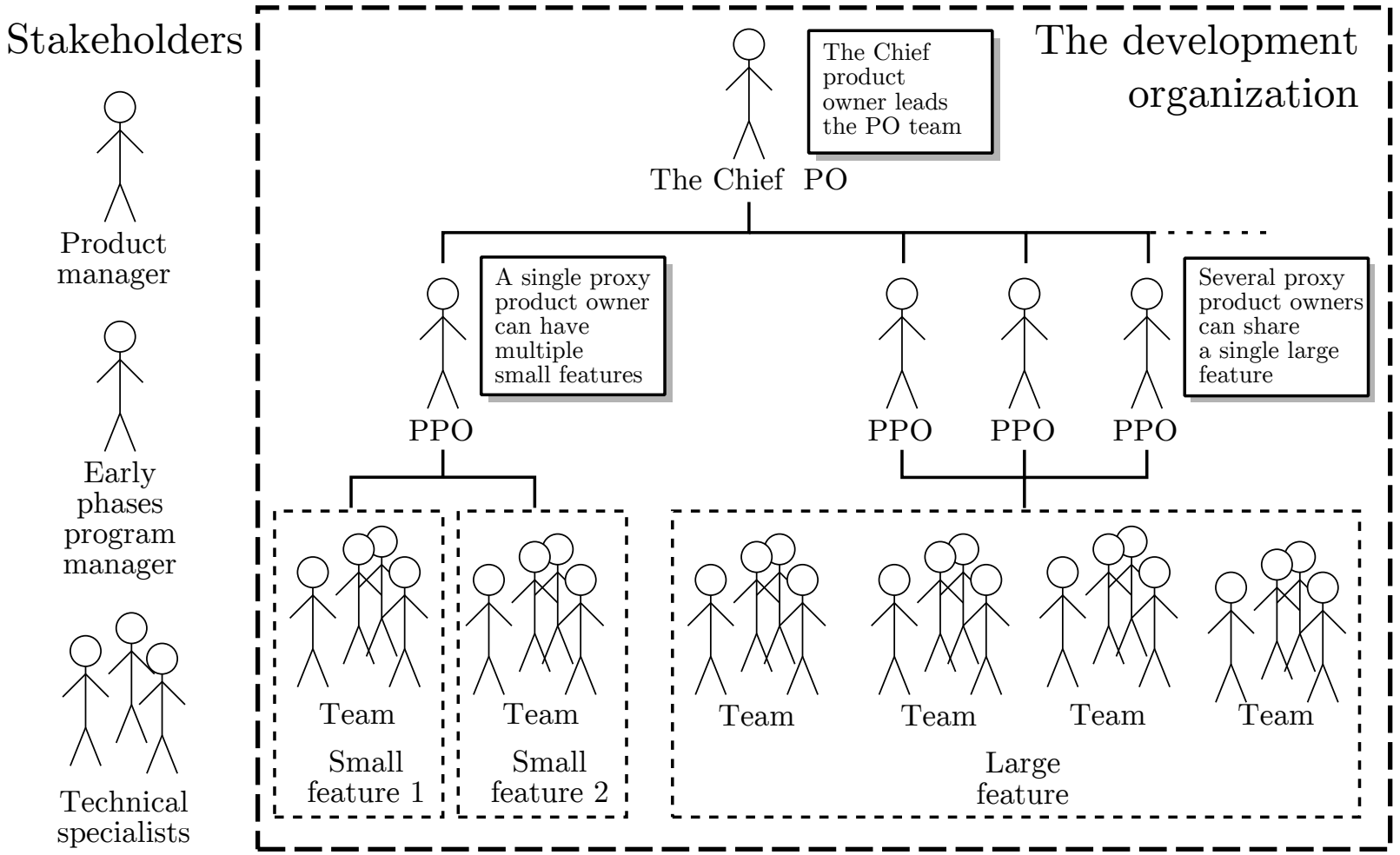
5) Preparing well for the first PI planning event

6) Having a full-time RTE



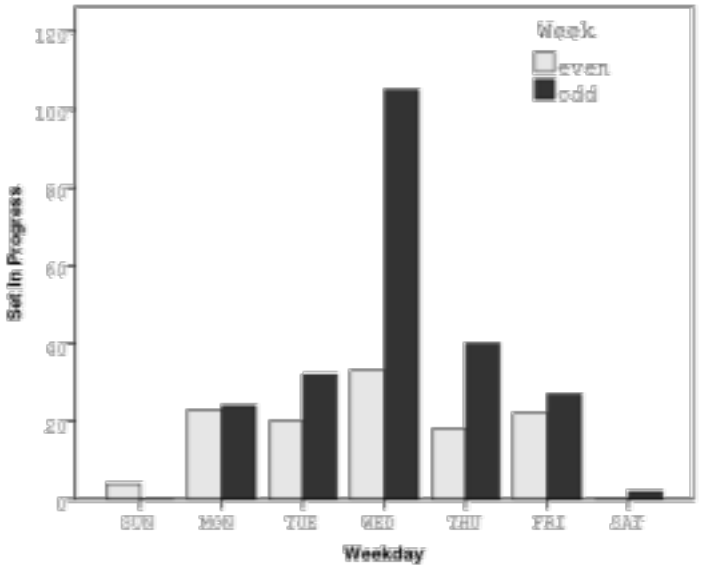
Backup material

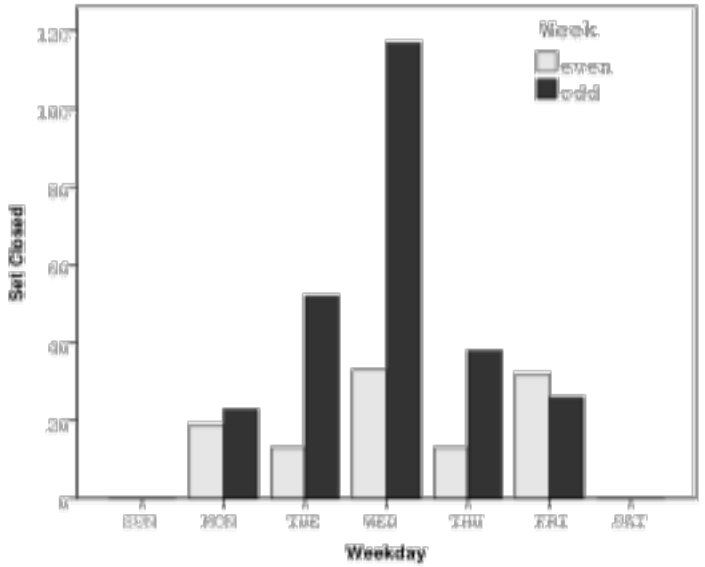
Results: Backlog Management



Results: Starting and Closing Stories

- Scrum: Stories are opened after sprint planning and closed at the sprint review
- Most often started during the bi-weekly sprint planning and closed during the bi-weekly sprint review day.
- Less than a third (30%) started during the sprint planning day and approximately a third (32%) were closed during the sprint review day.
- Clear mismatch with Scrum
- The utilization of developers was kept high by starting new user stories mid-sprint
  - Optimizing resource usage, not ”flow”
- Developers wanted to demonstrate and close stories as soon as they were ready
  - Became “part of the legacy”

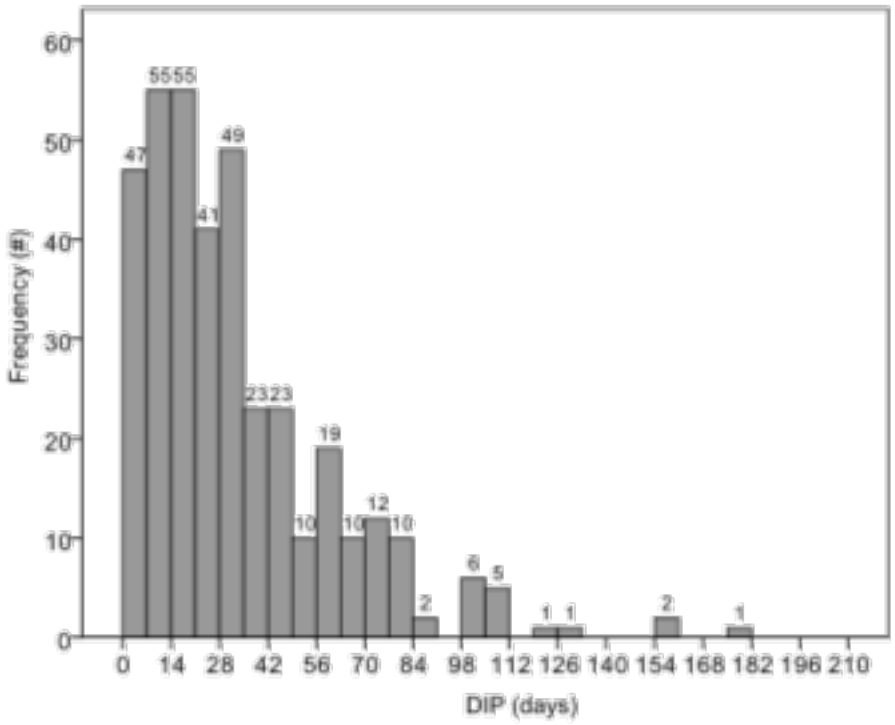




Results: User-story development time

- Scrum: Expected development time: 1 sprint (14 days)

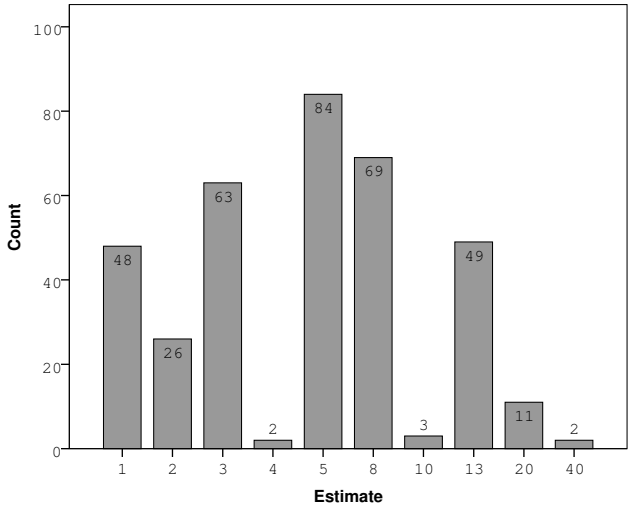
- Actual mean development time: 27 days
- Less than a third completed in one sprint
- Clear mismatch with Scrum
  - One-sample Wilcoxon Signed-Rank Test confirms ($p < 0.001$)
- Inter-team and external dependencies postpone user story closing
- Large and complex system – difficult to create small end-to-end user-stories

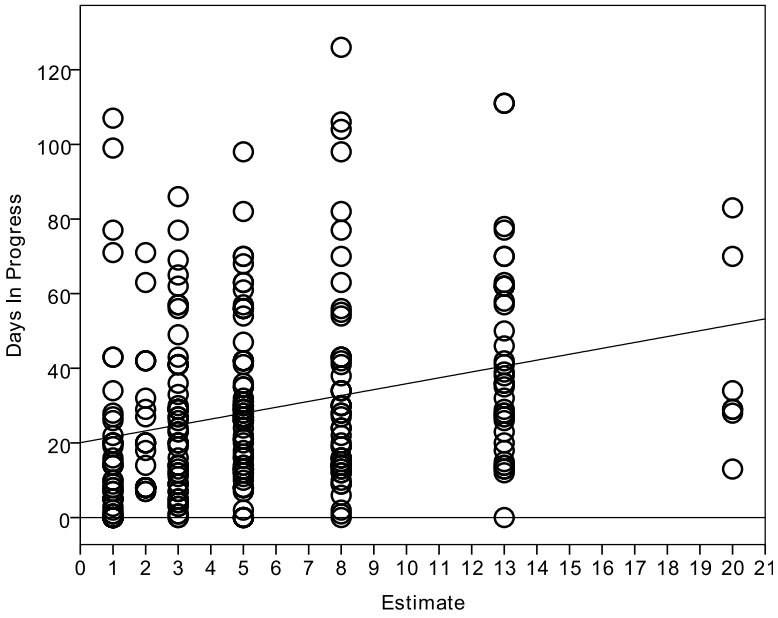


Results: Estimates and the development time

- Scrum: Estimate and DIP should not correlate, as all stories are opened and closed at the same time regardless of size (in this case: DIP should be 14 days)

- No notable correlation overall
  - Kendall's tau-b = 0.255 ($p < 0.001$)
- Most teams have no notable correlation
- Three teams with moderate significant correlation
  - $\tau_B = 0.576, 0.661, 0.484$ ($p = 0.010, 0.002, 0.014$)
- No explanation for the result from these teams





USIKKERHETSVURDERING FOR NYTTEPOENG OG KOSTPOENG

JO HANNAY

SIMULA RESEARCH LABORATORY

«Nytte og kostnadsstyring av IT-prosjekter i en usikker verden»

HiT-frokostseminar 7. mars 2018

simula

Suksess med IT i offentlig sektor (SMIOS)

Nyttestyre

- Hva da?
- Hva vil det si?

Lønnsomhetsanalyse

Produktelementer

Tiltak/Prosjekt

= produktelement

Kravspesifikasjon

(Minste) levedyktige produkt

(Minimal) marketable feature

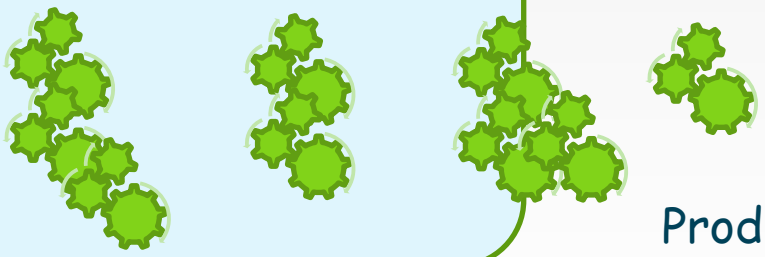
Konstruksjon

(Minste) levedyktige endring

(Minimal) viable change

Epos/Epic
Historier/stories

Gevinst



Produkt

simula
Suksess med IT i offentlig sektor
(SMIOS)

Lønnsomhetsanalyse

Nytte/kost-indeks

Nyttestyre:
Maksimere nytte i forhold til kost tidlig

Tiltak/Prosjekt

= produktelement

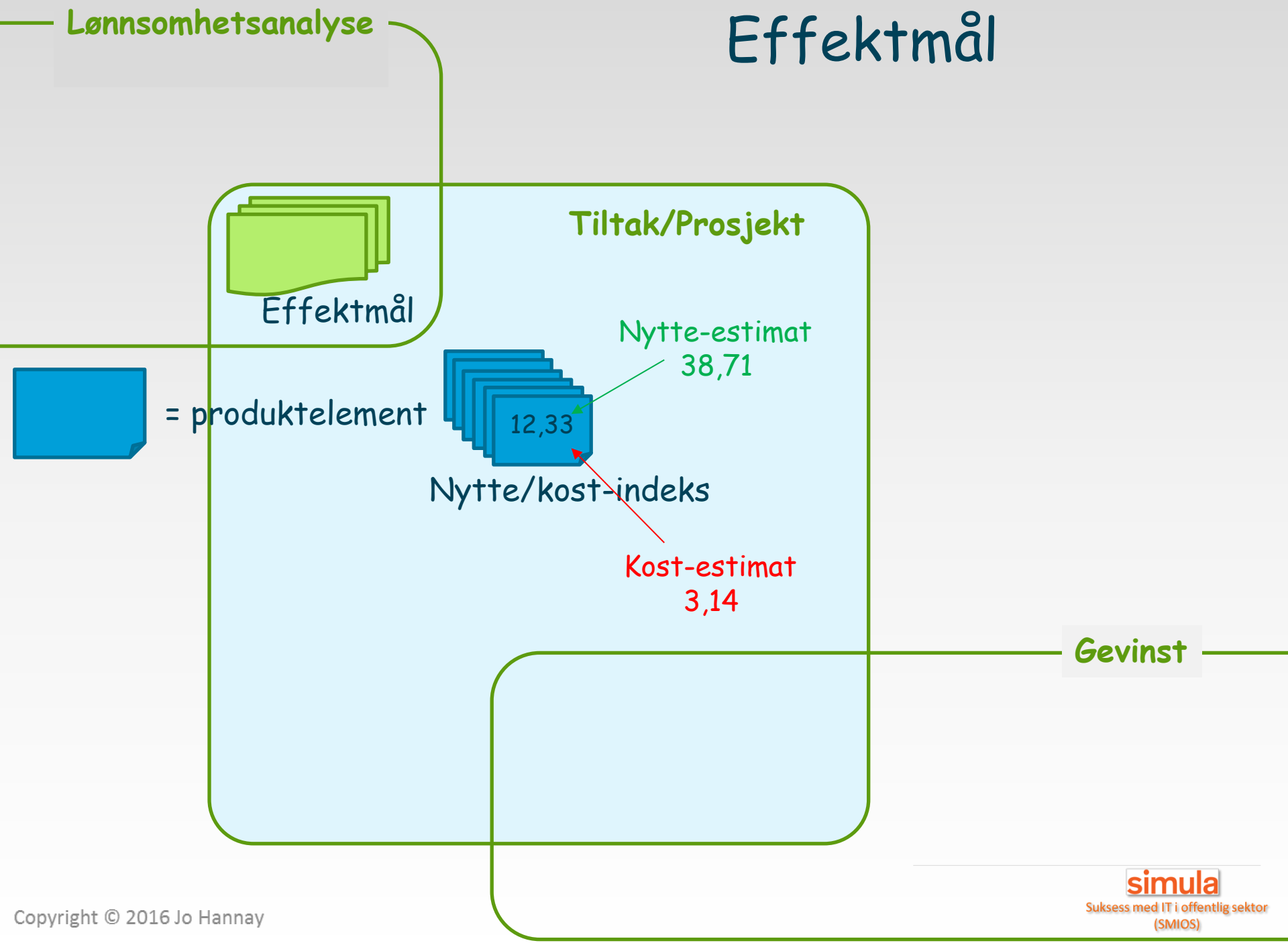
Nytte-estimat
38,71

Nytte/kost-indeks

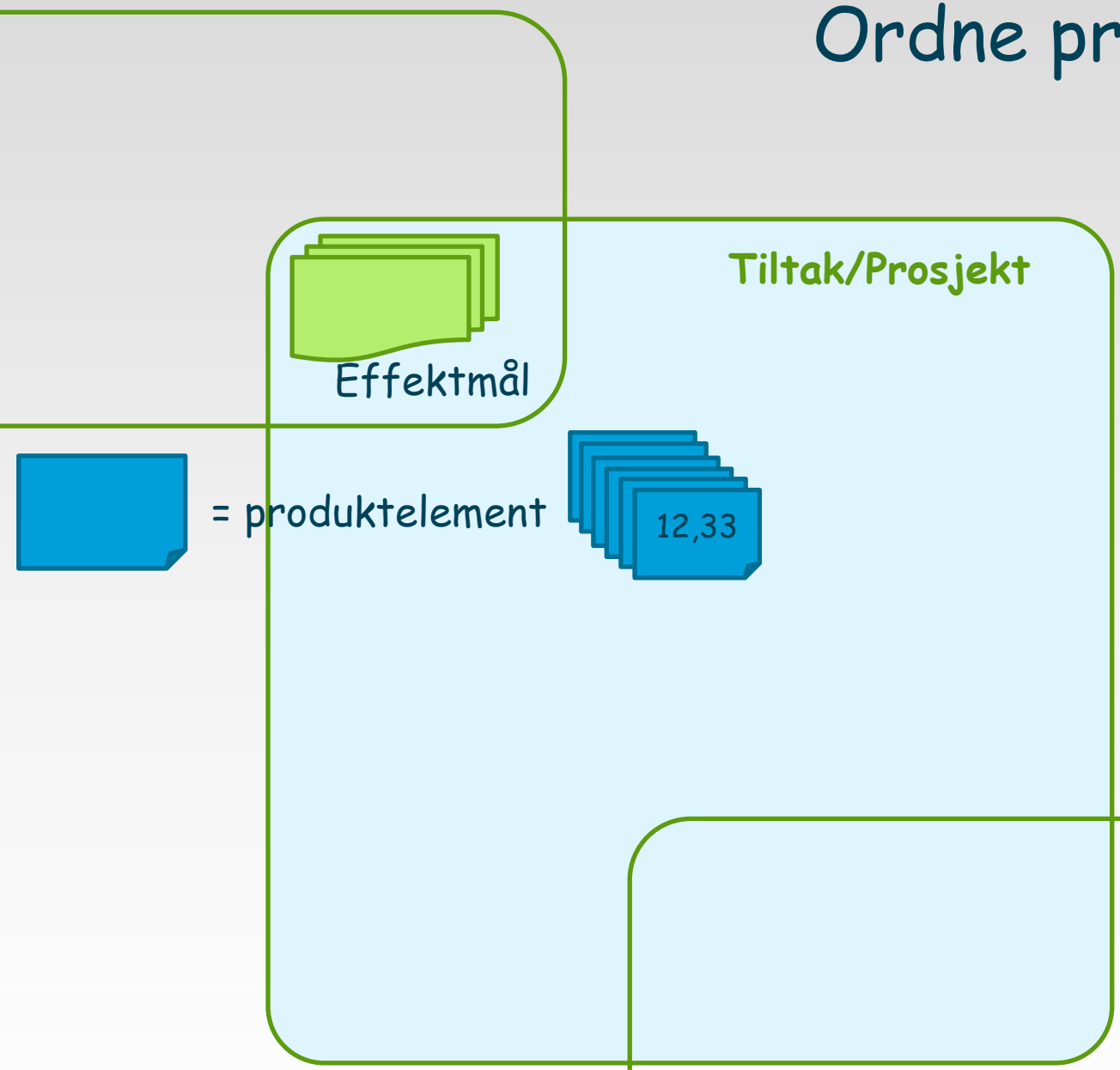
Kost-estimat
3,14

Gevinst

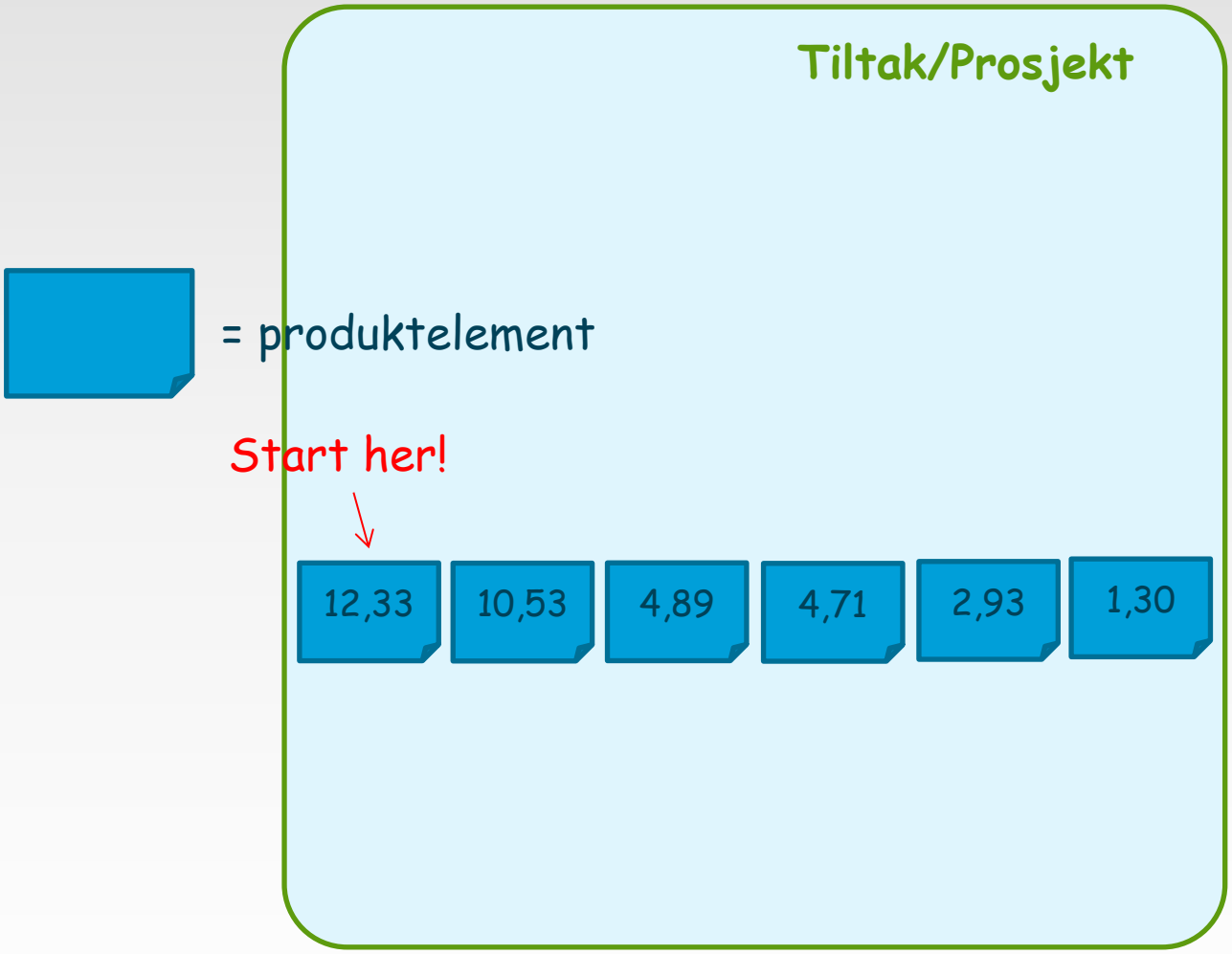
Effektmål



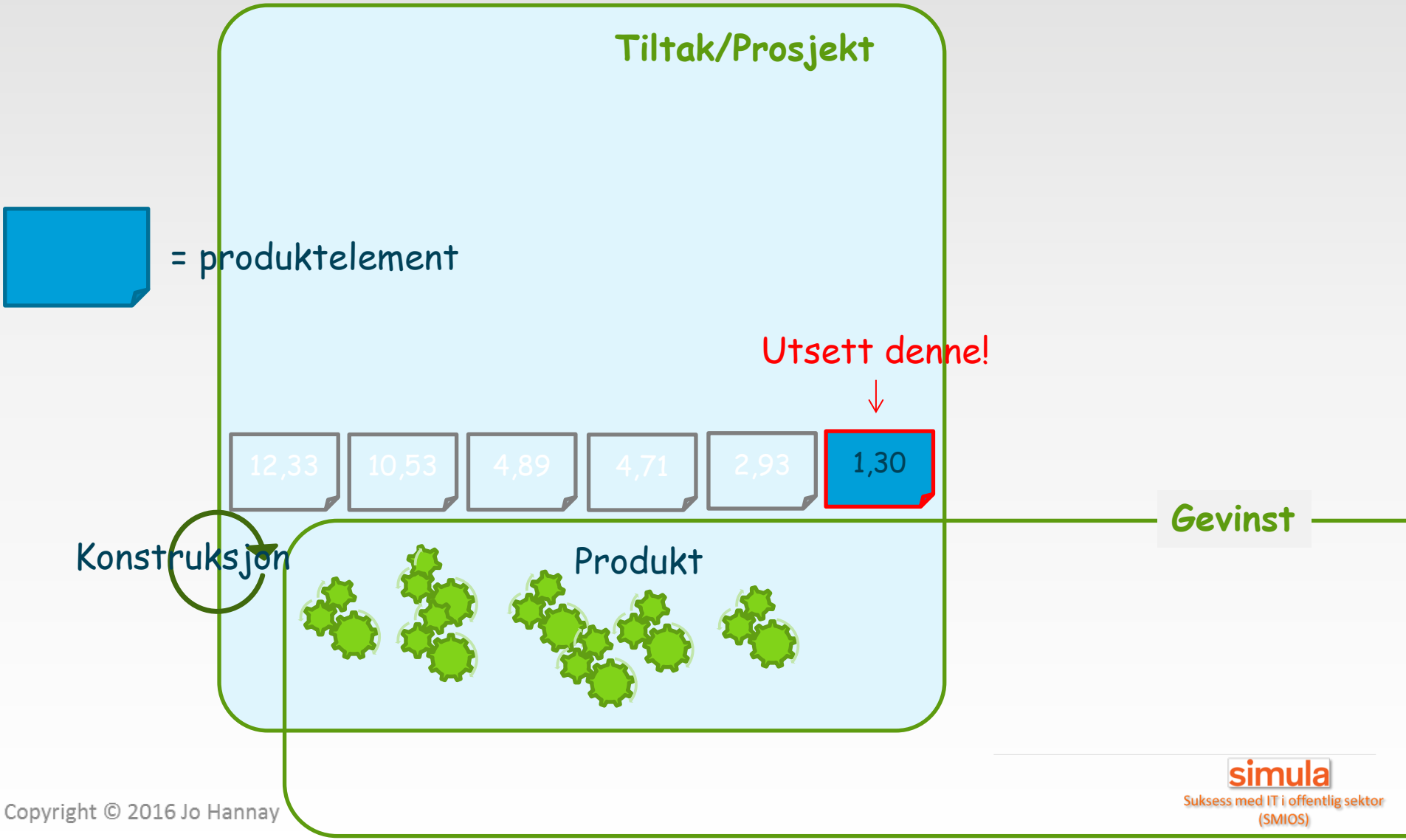
Ordne produktkø



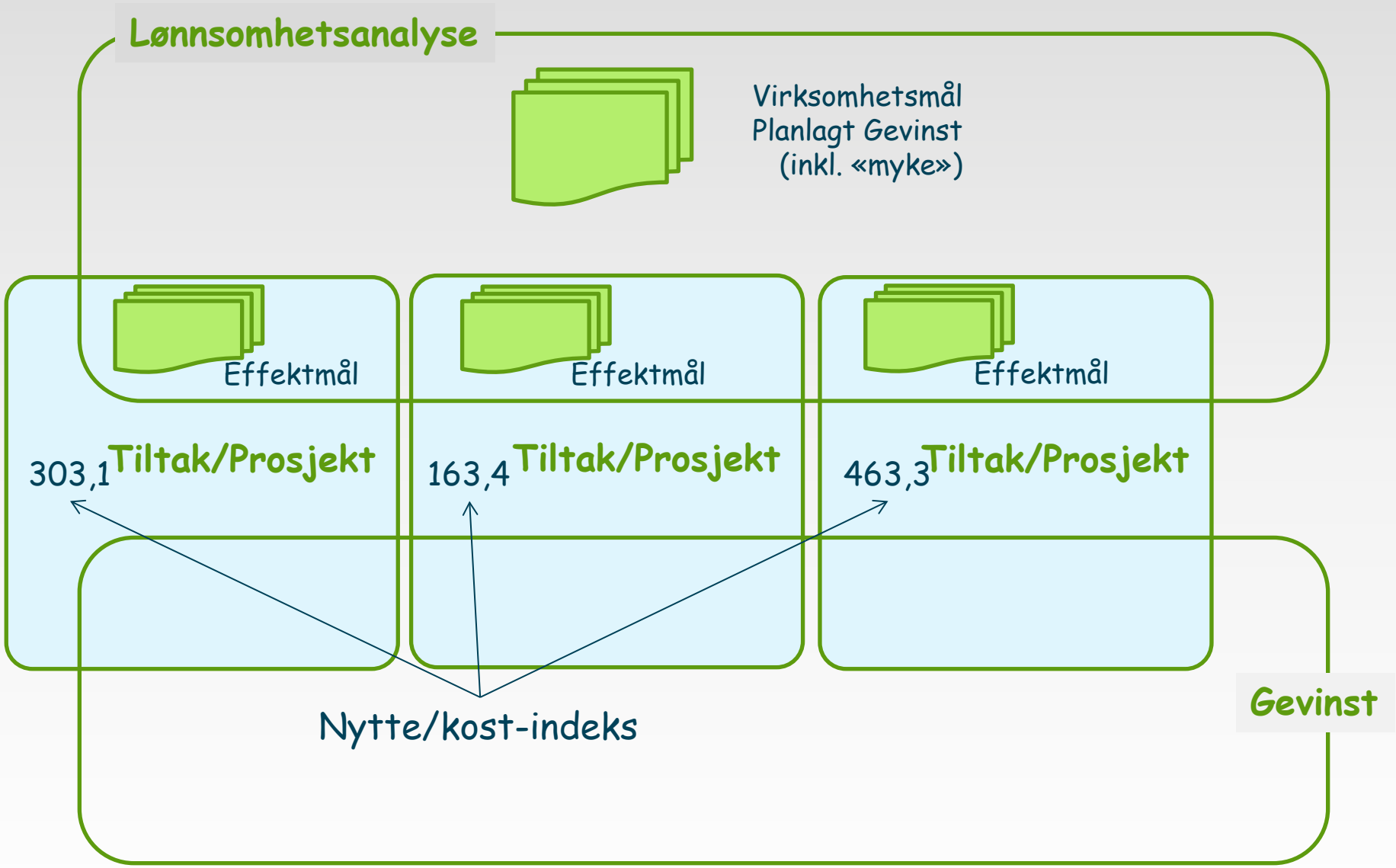
Ordne produktkø



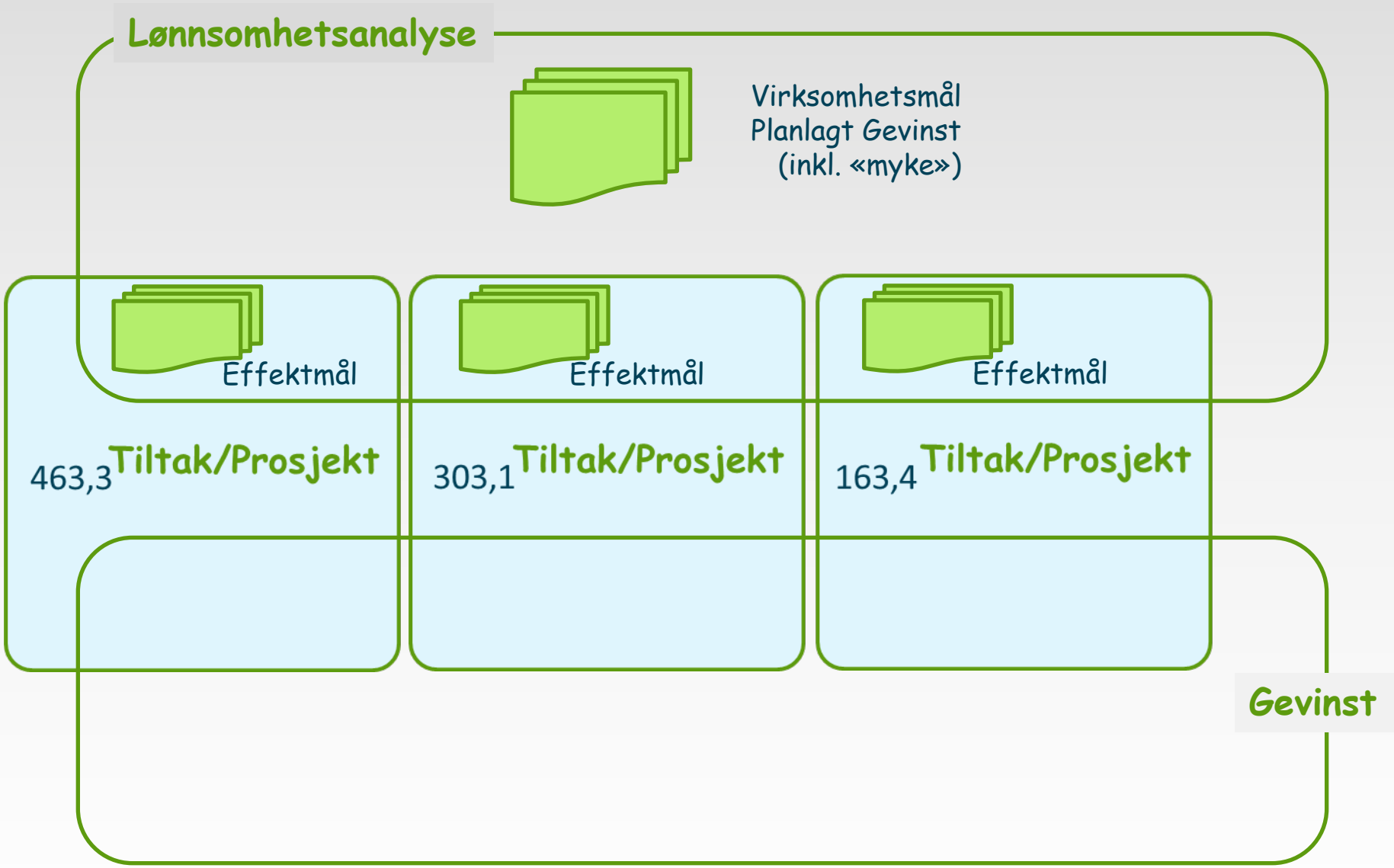
Stoppe i tide



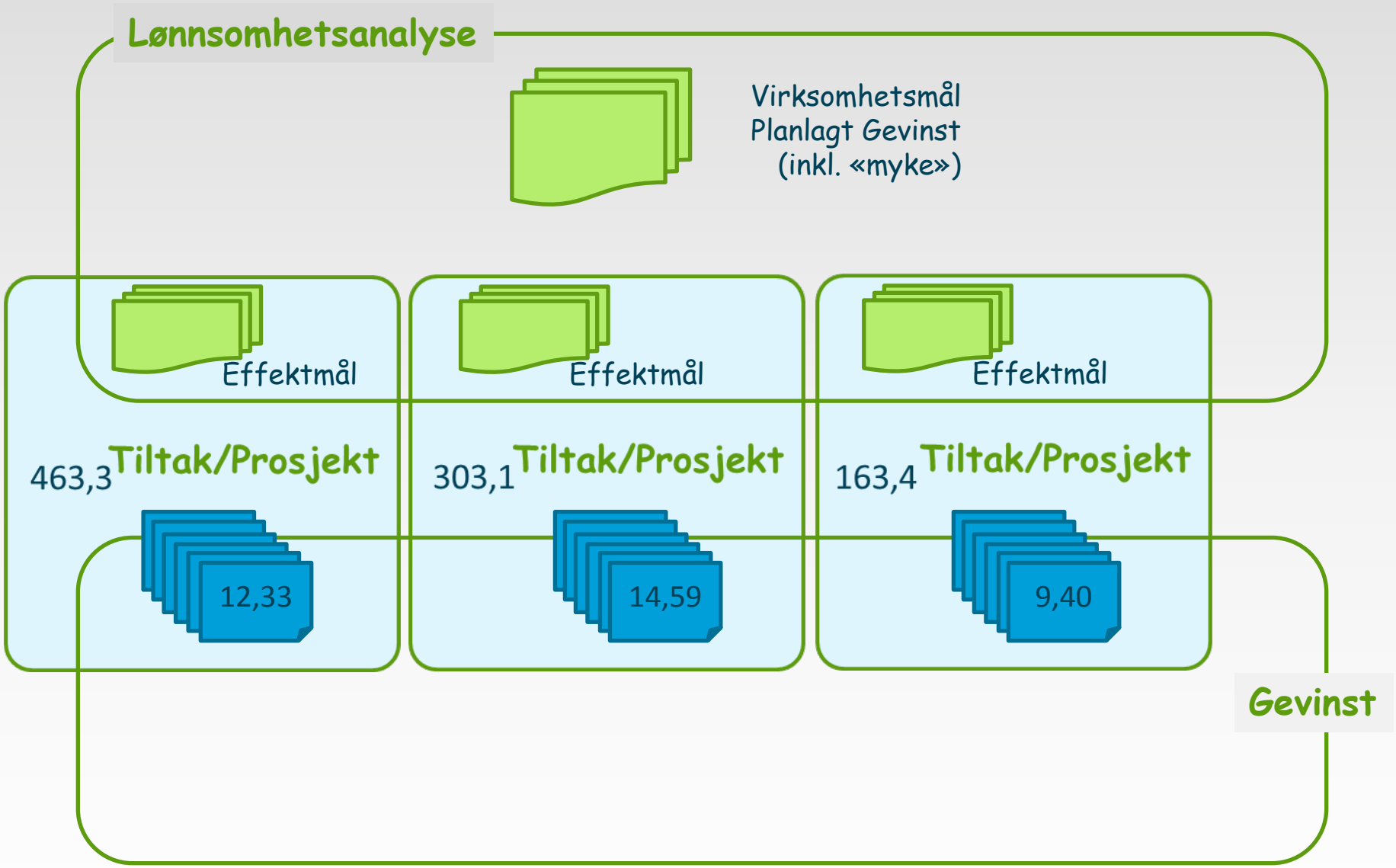
Nytte/kost-indeks for tiltak



Ordne porteføljen

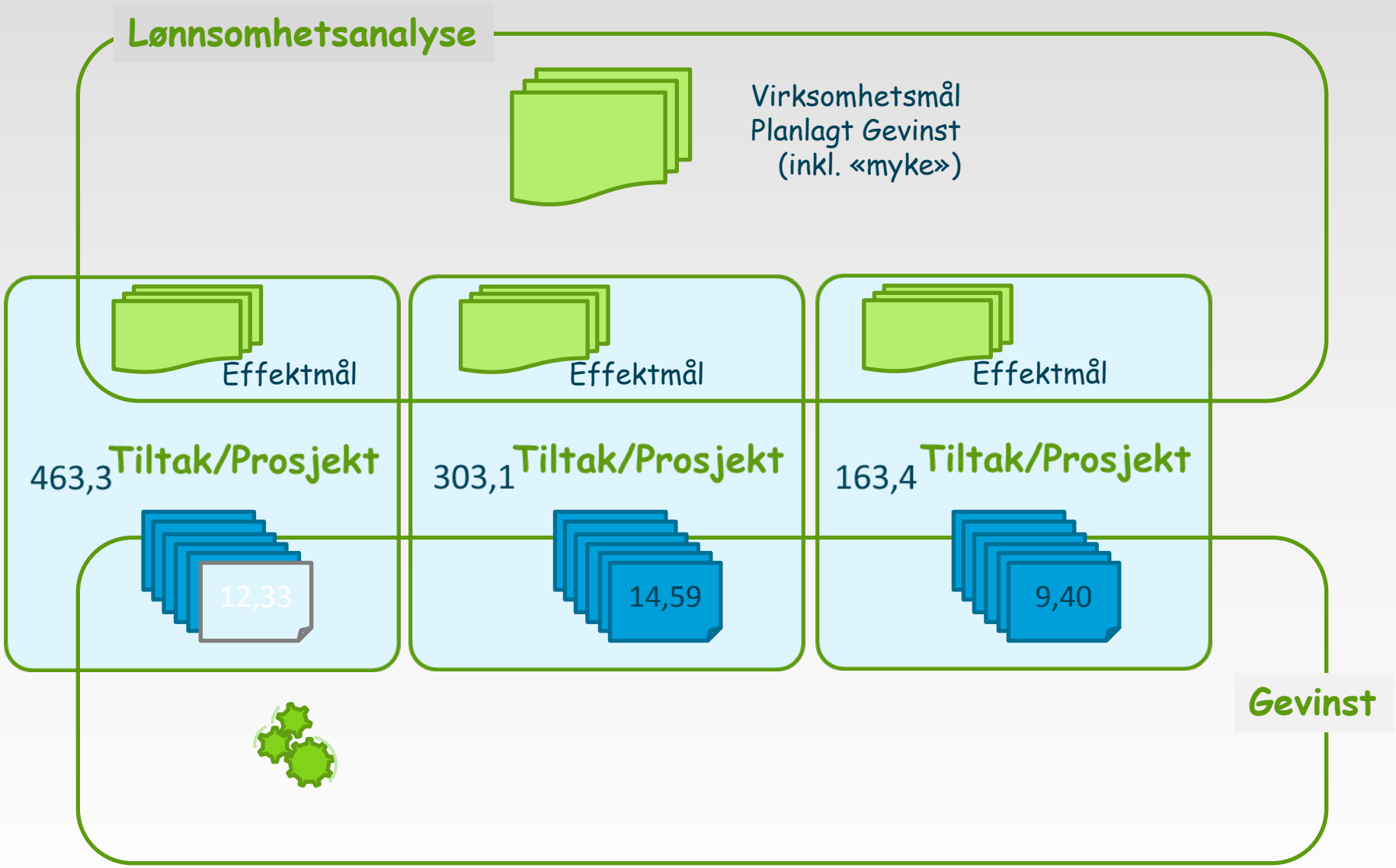


Konstruksjon i porteføljen

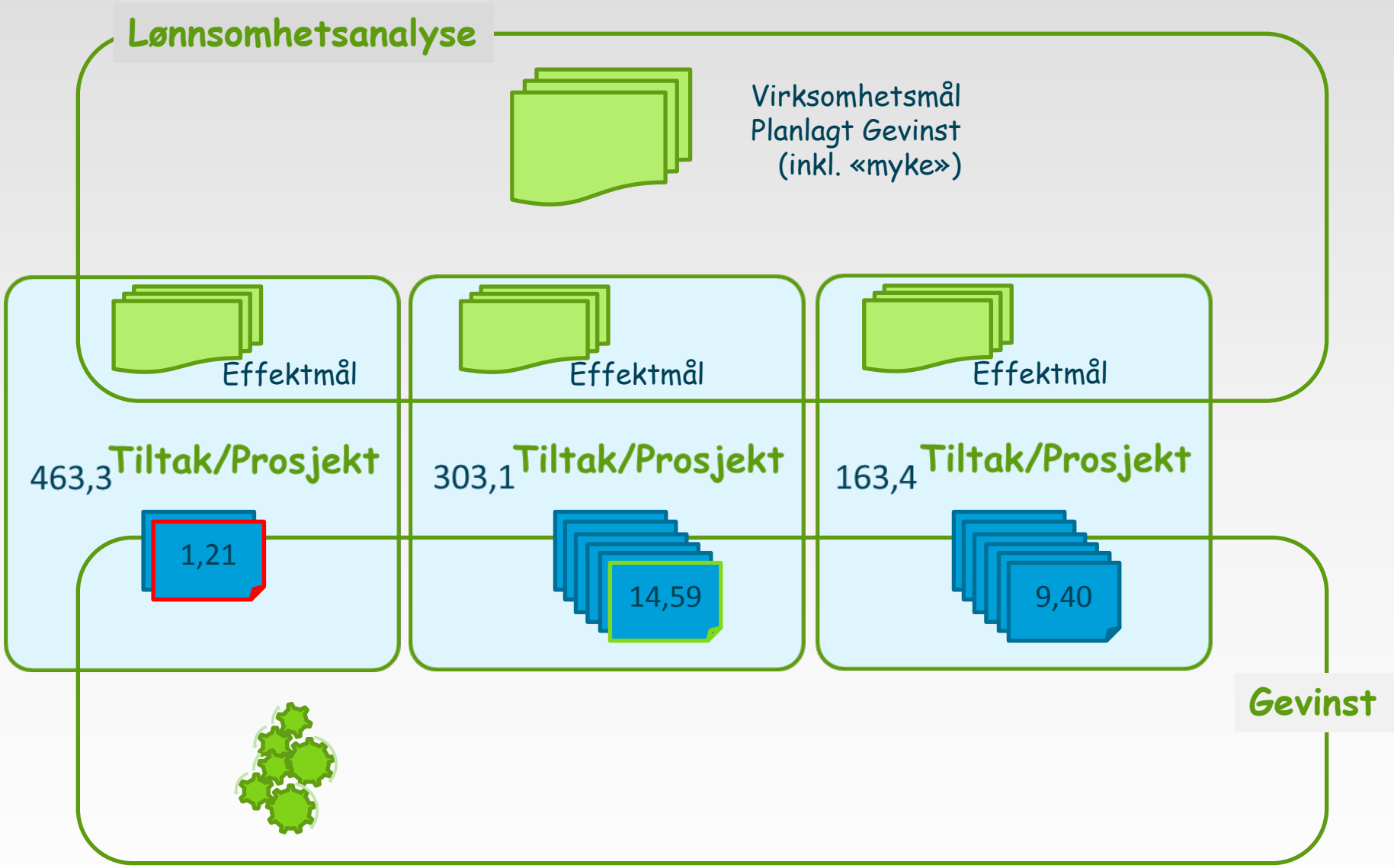


simula
Suksess med IT i offentlig sektor
(SMIOS)

Konstruksjon i porteføljen

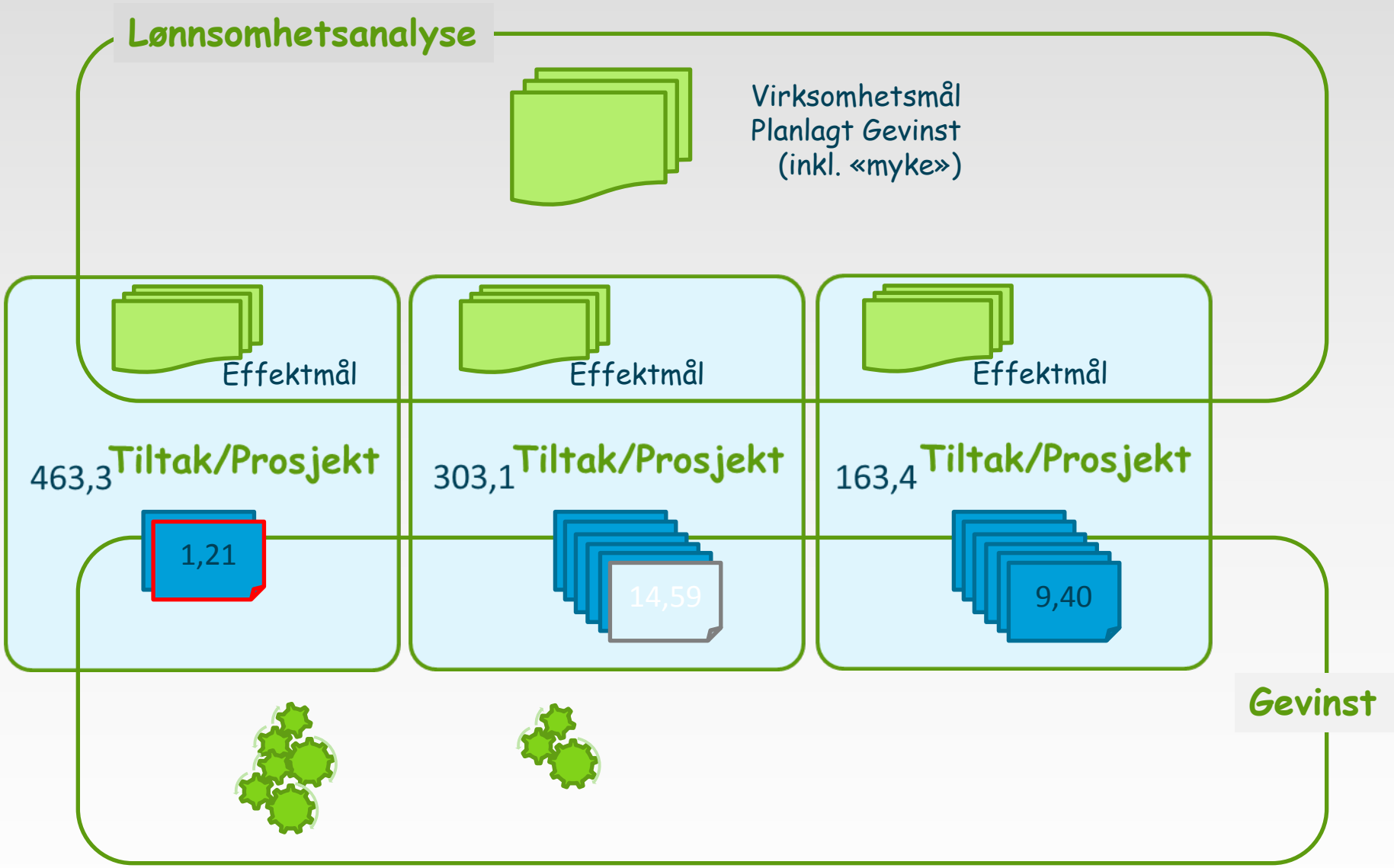


Konstruksjon i porteføljen



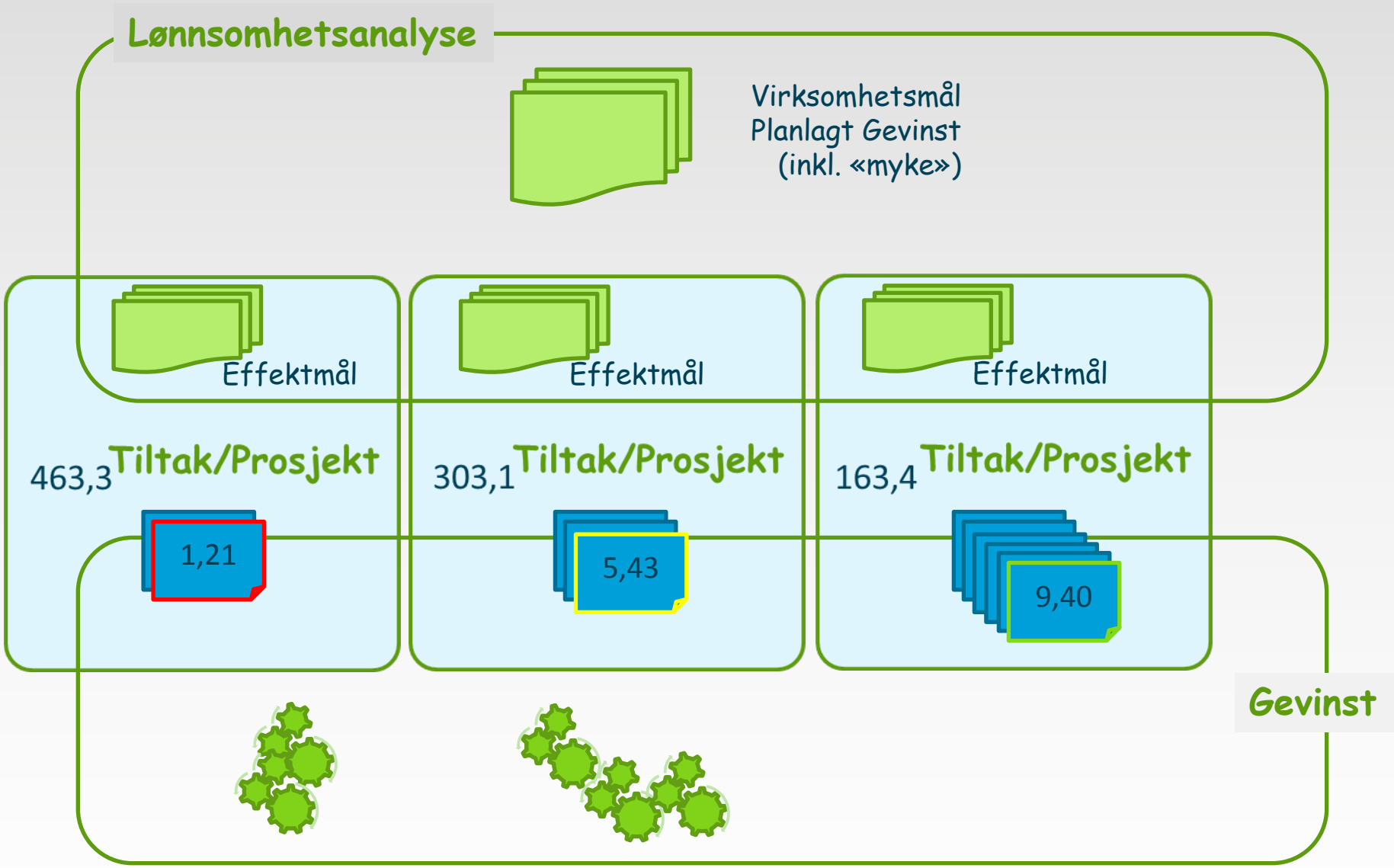
simula
Suksess med IT i offentlig sektor
(SMIOS)

Konstruksjon i porteføljen



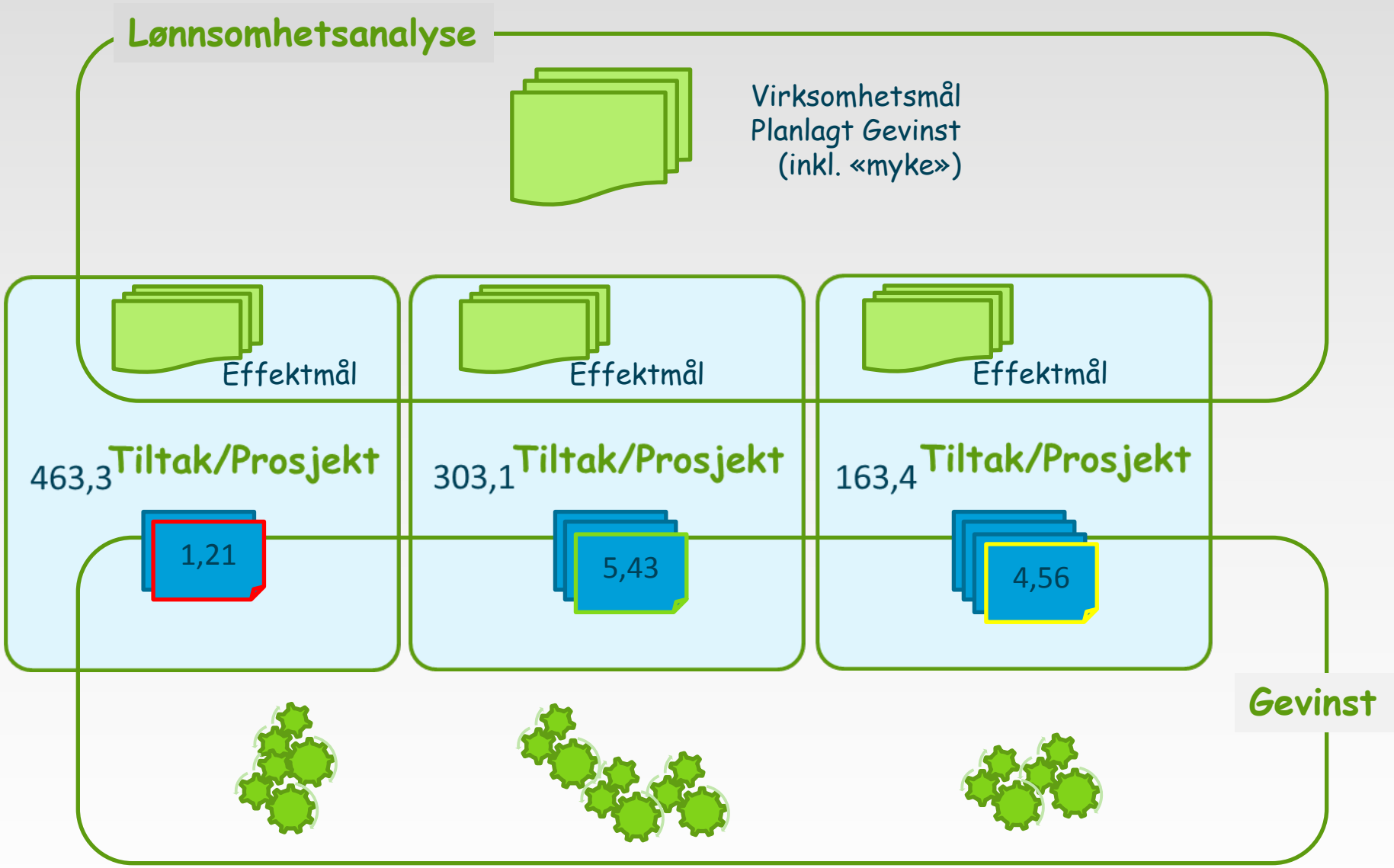
simula
Suksess med IT i offentlig sektor
(SMIOS)

Konstruksjon i porteføljen



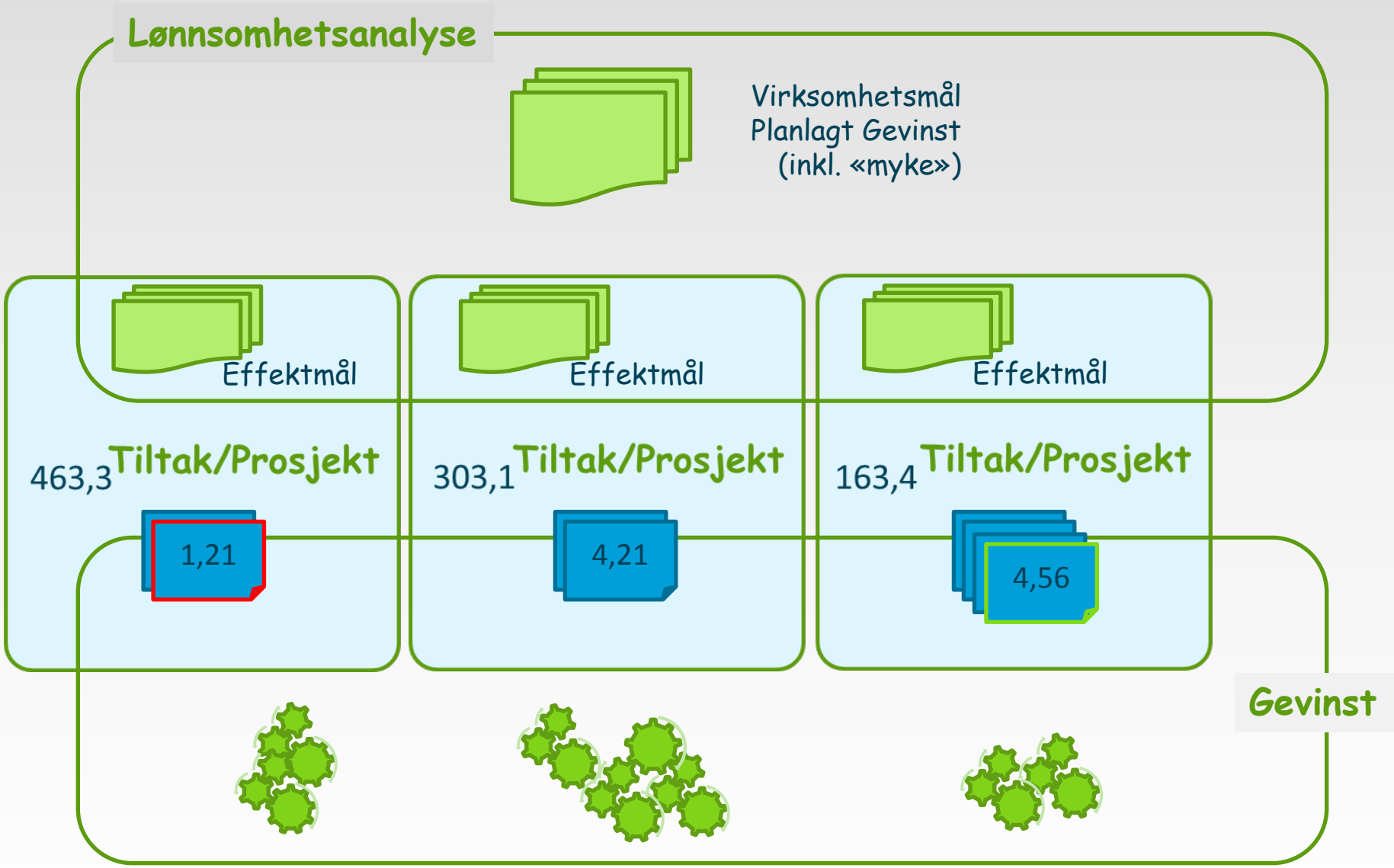
simula
Suksess med IT i offentlig sektor
(SMIOS)

Konstruksjon i porteføljen



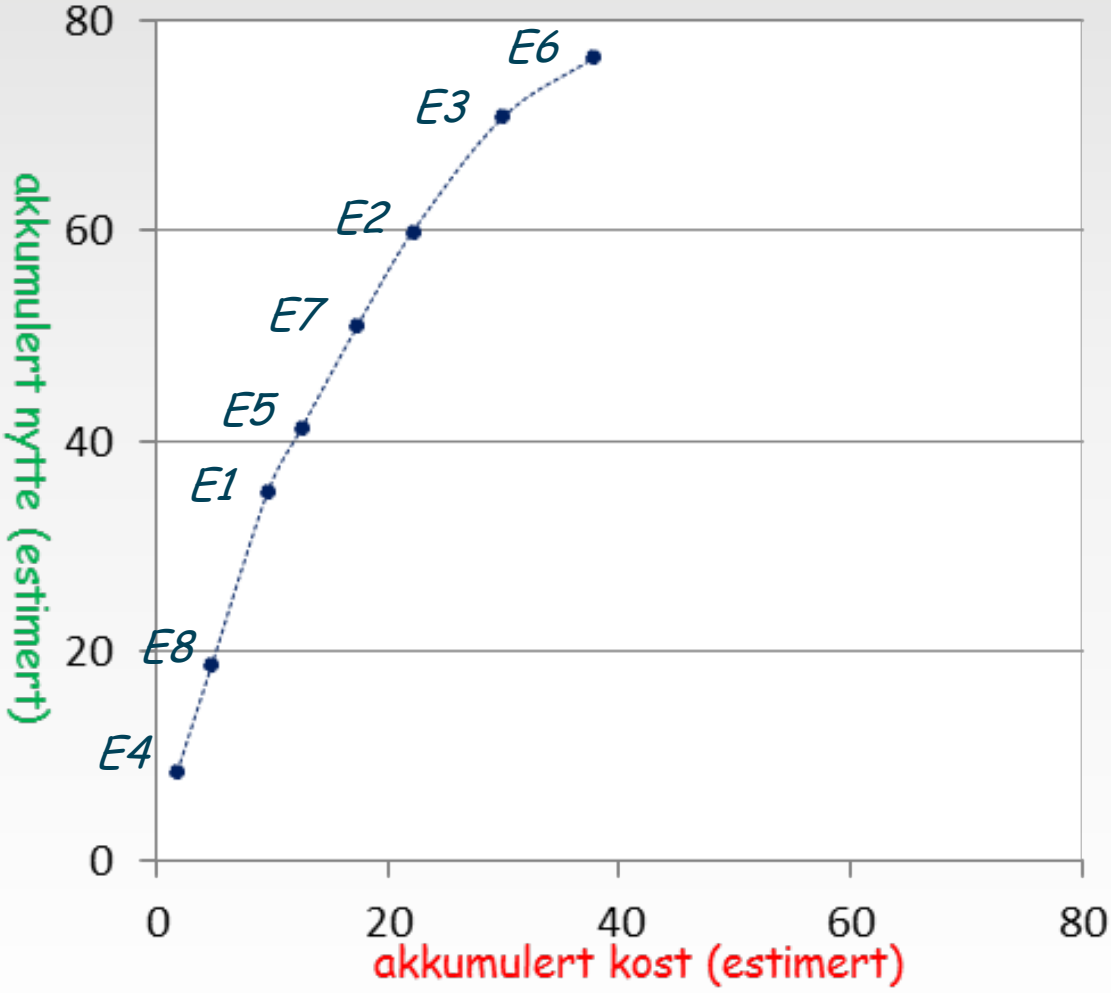
simula
Suksess med IT i offentlig sektor
(SMIOS)

Konstruksjon i porteføljen



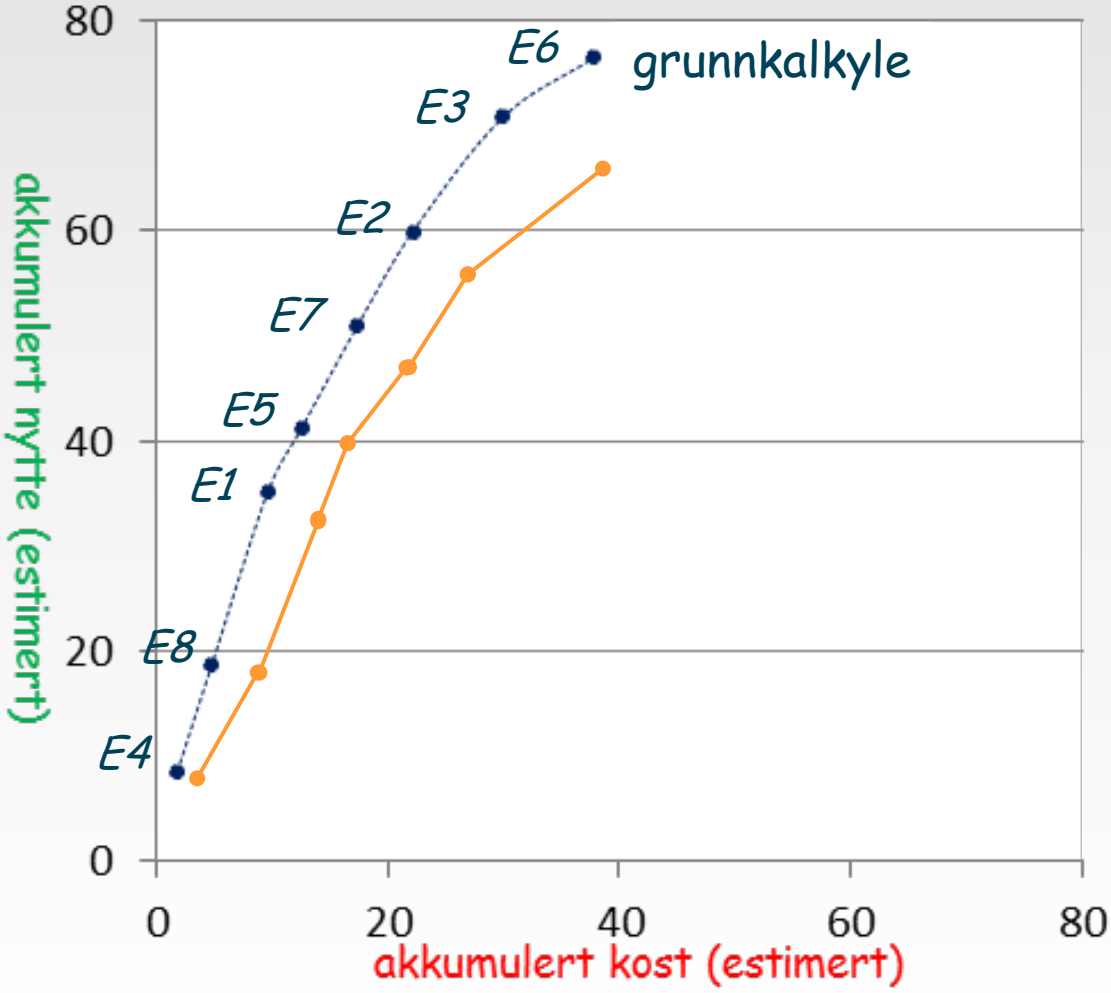
simula
Suksess med IT i offentlig sektor
(SMIOS)

Inntjent forretningsverdi (Earned Business Value)



Med nytteestimater og kostestimater på produktelementene kan man planlegge for å optimere nytte/kost tidlig

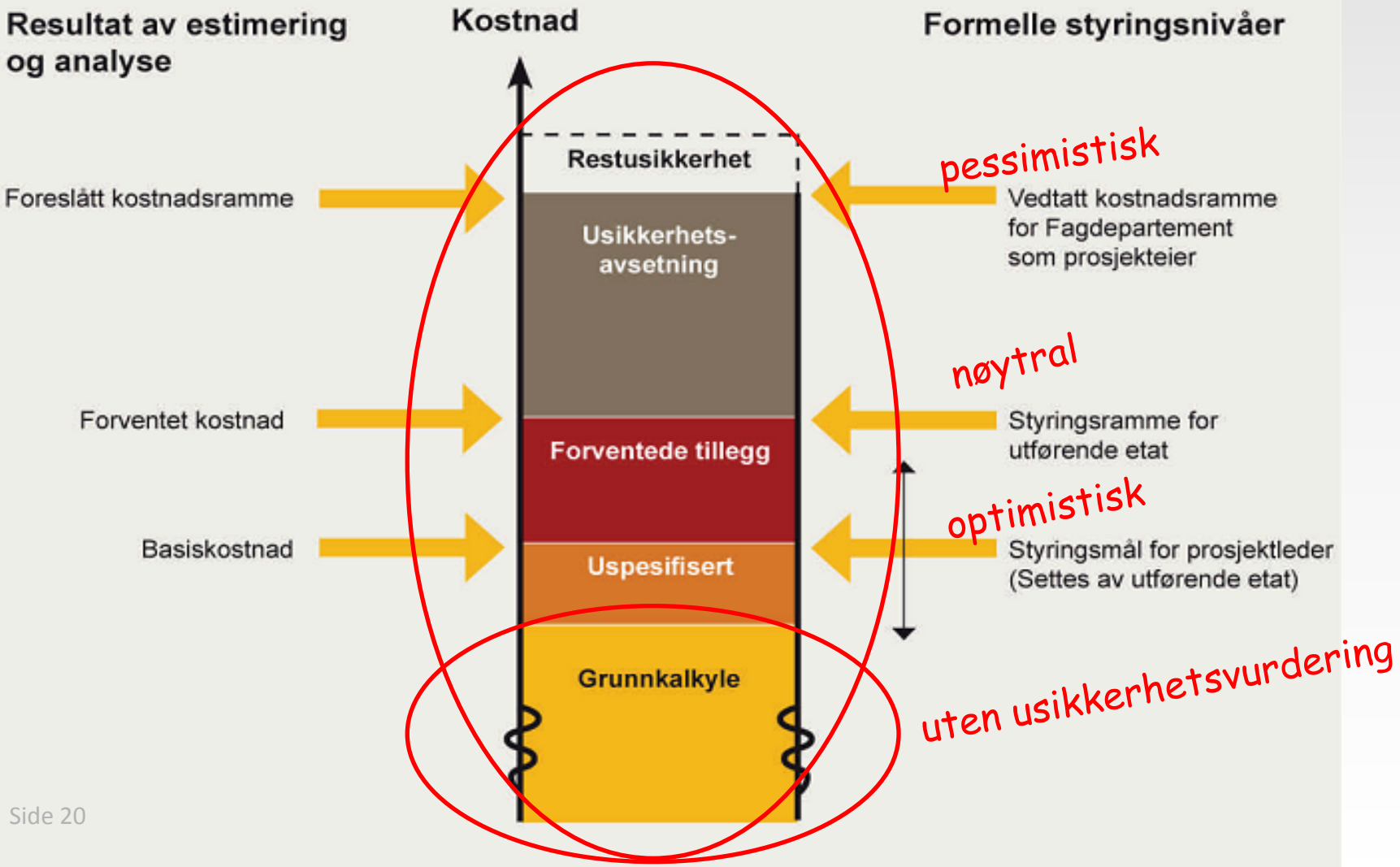
Inntjent forretningsverdi (Earned Business Value)



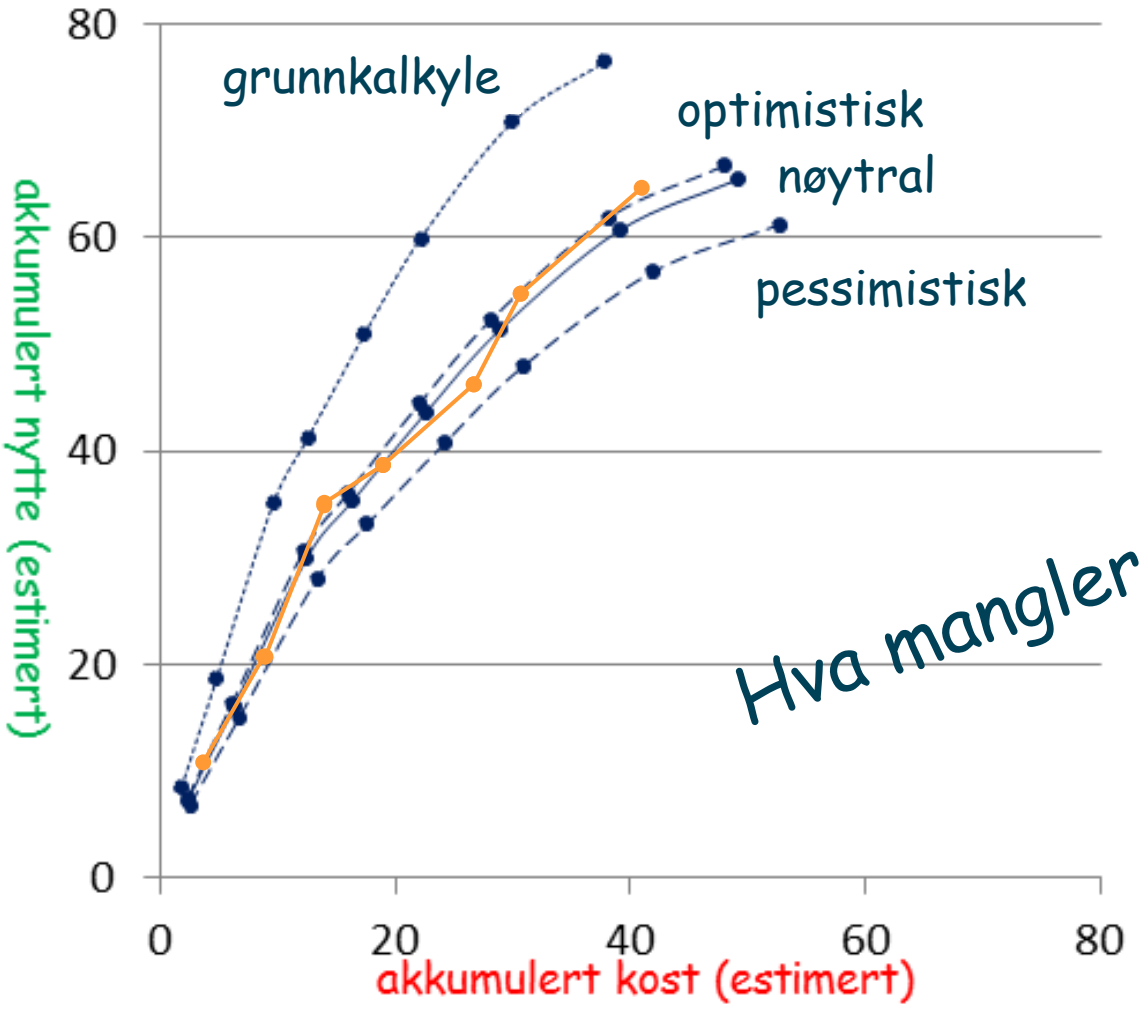
…og man følge med på hvordan man faktisk ligger an i forhold til planen, for både kost og nytte.

KS2 usikkerhetsregime for kost

Figur 1 Sammenhengen mellom kjernebegrepene
(for nærmere definisjoner: se kapittel 3)

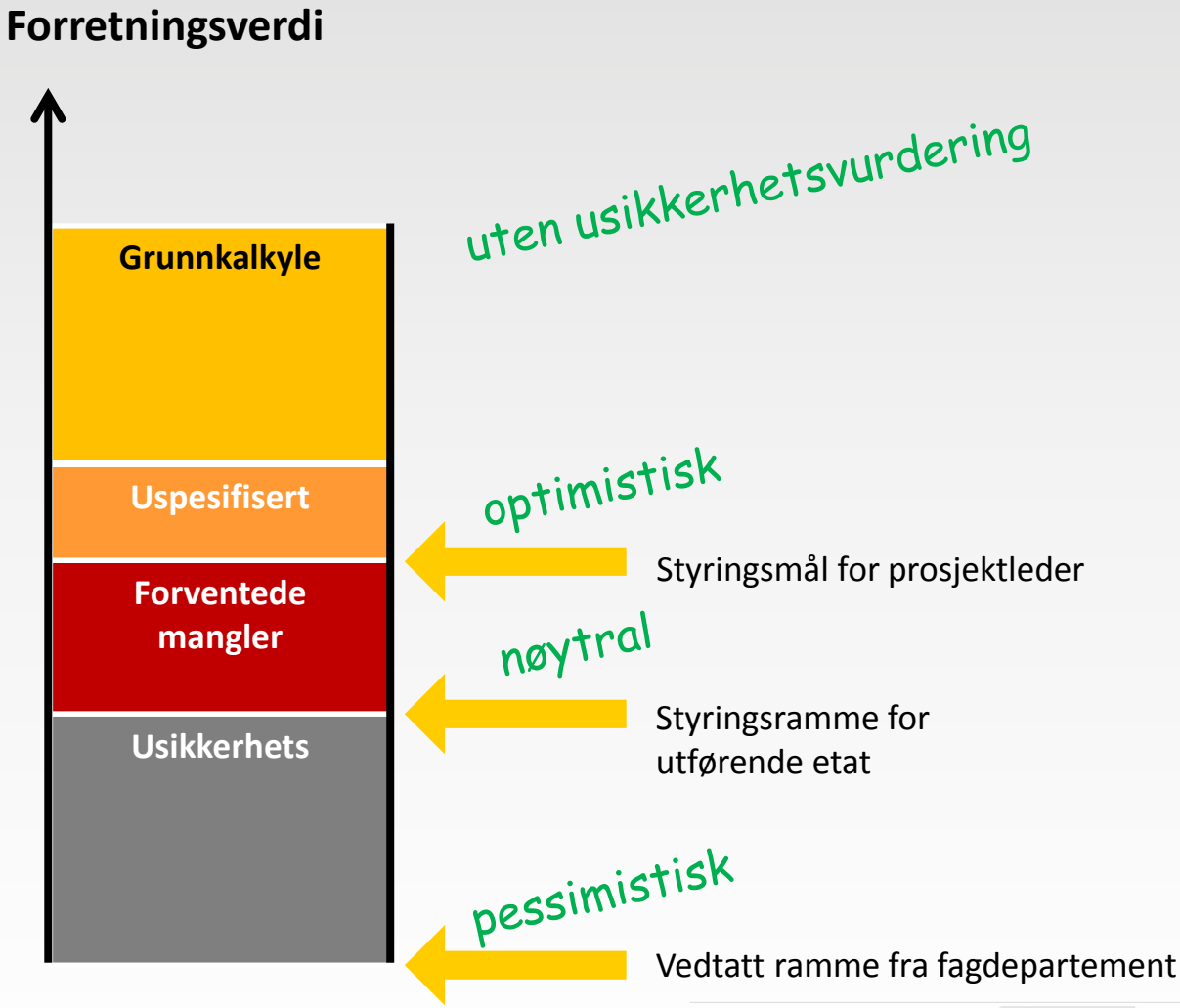


Inntjent forretningsverdi (Earned Business Value)



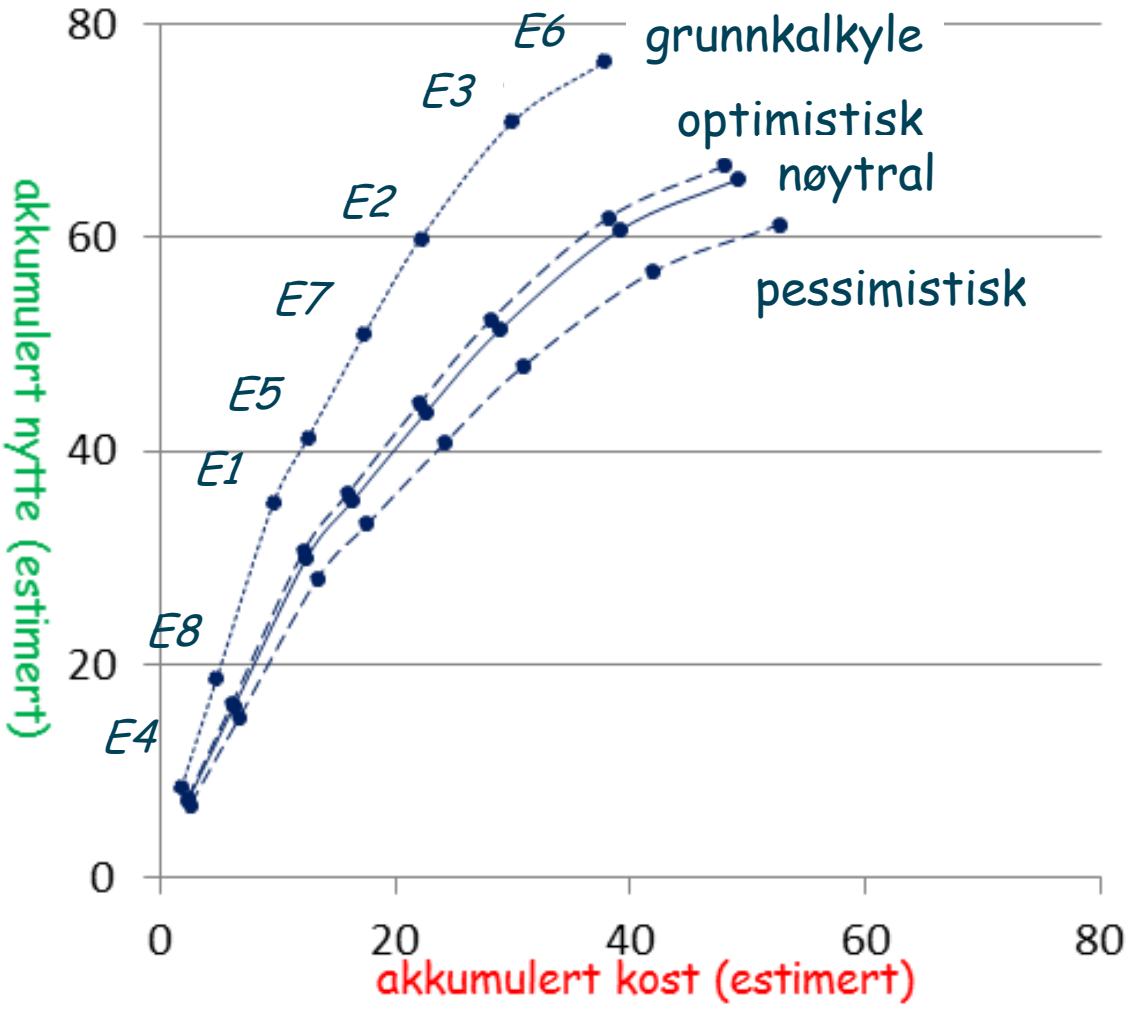
Hva mangler for å få til dette?

KS2 usikkerhetsregime for forretningsverdi



Inntjent forretningsverdi (Earned Business Value)

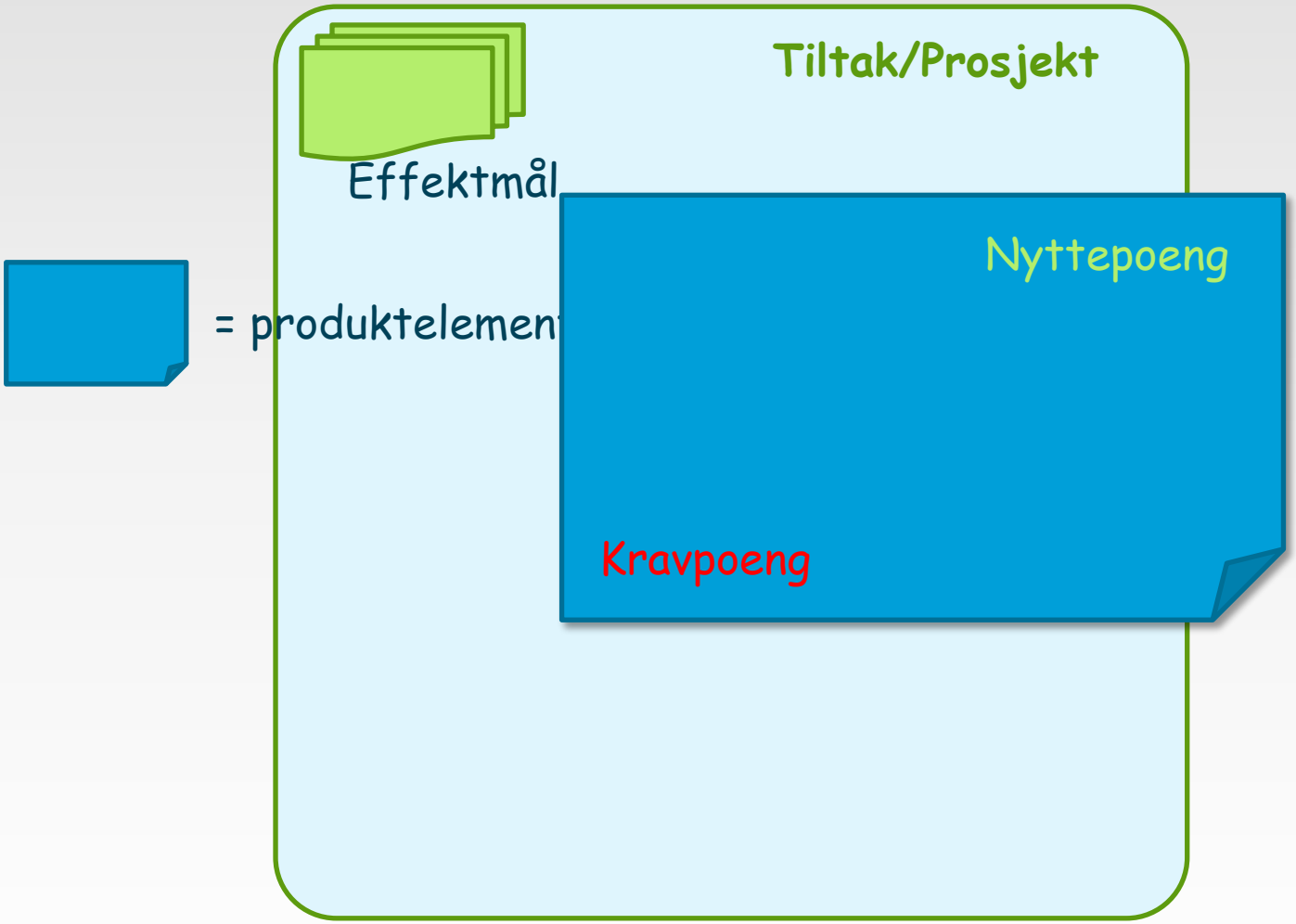
KS2 usikkerhetsregime



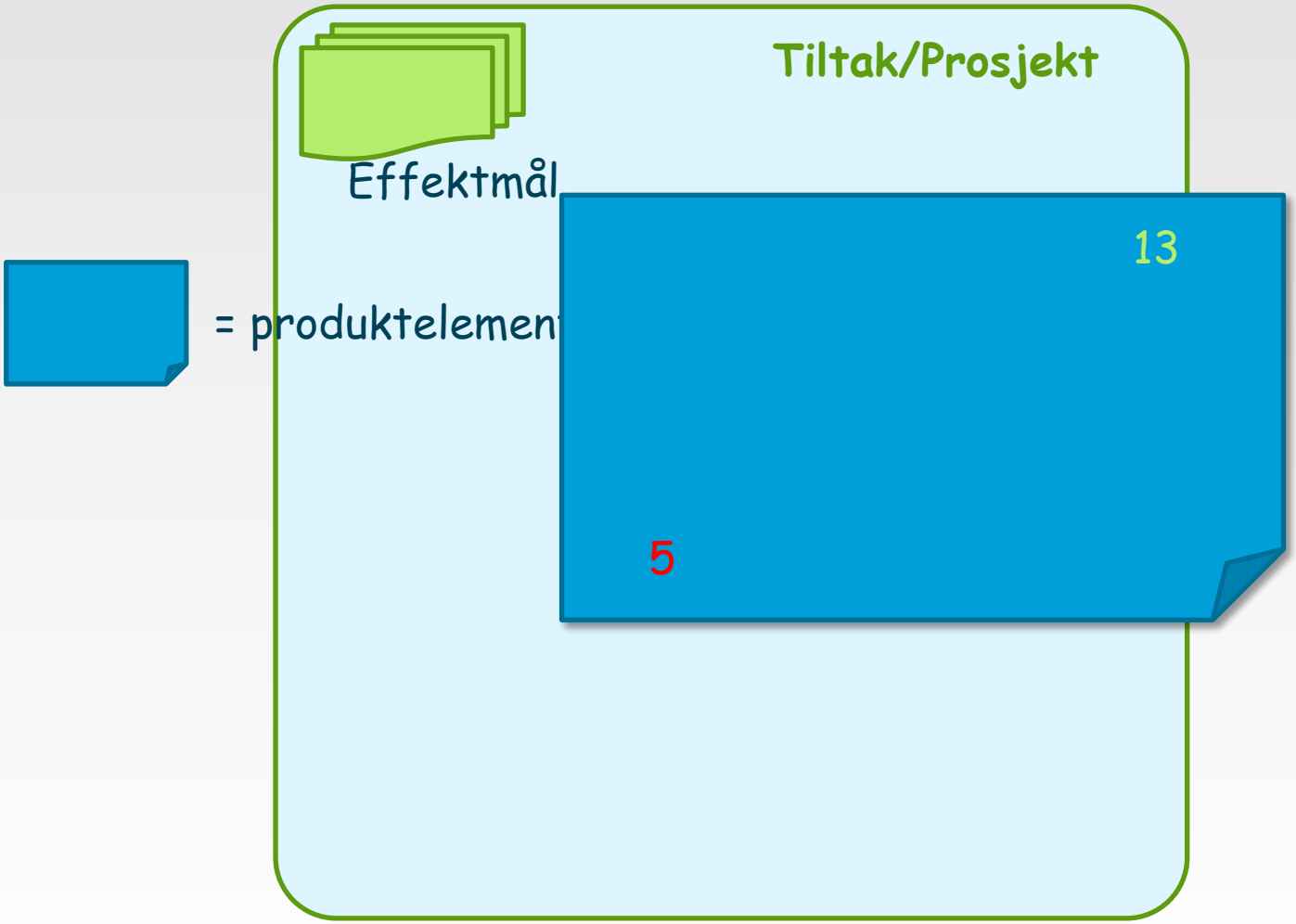
KS2 usikkerhetsregime

Vi får til dette takket være nytte- og kravpoeng

Nyttepoeng og kravpoeng



Nyttepoeng og kravpoeng

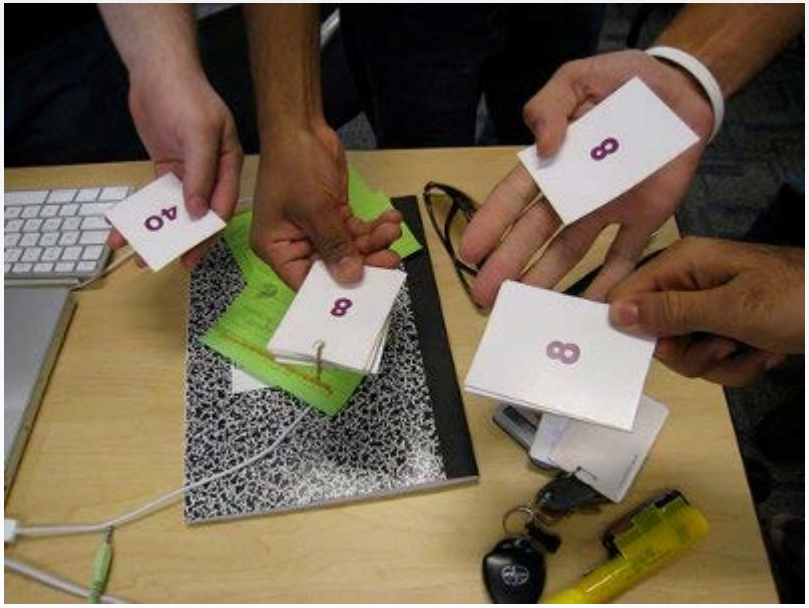


Uthenting av ekspertkunnskap

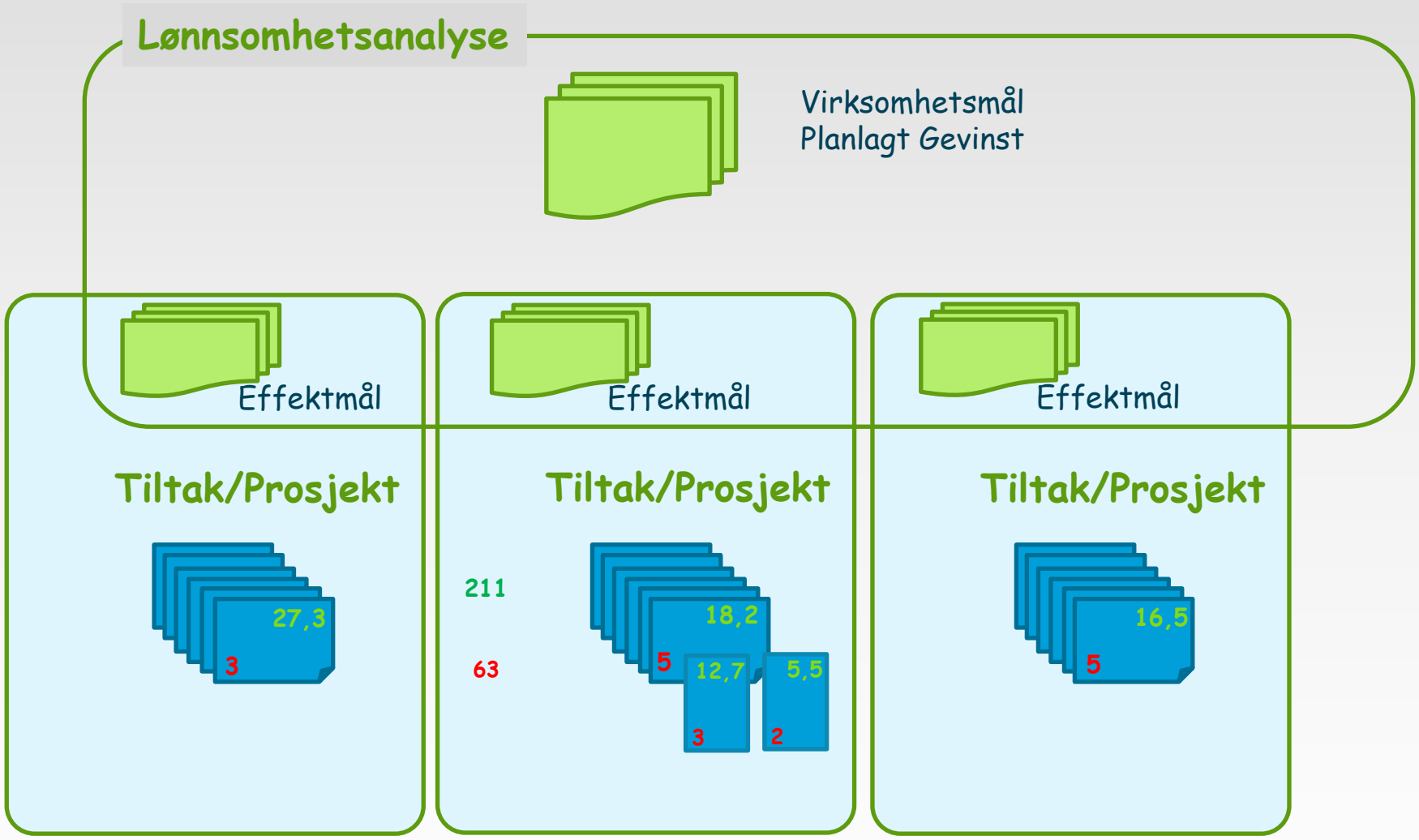
Relativ estimering

Nytte-Poker

Planning Poker



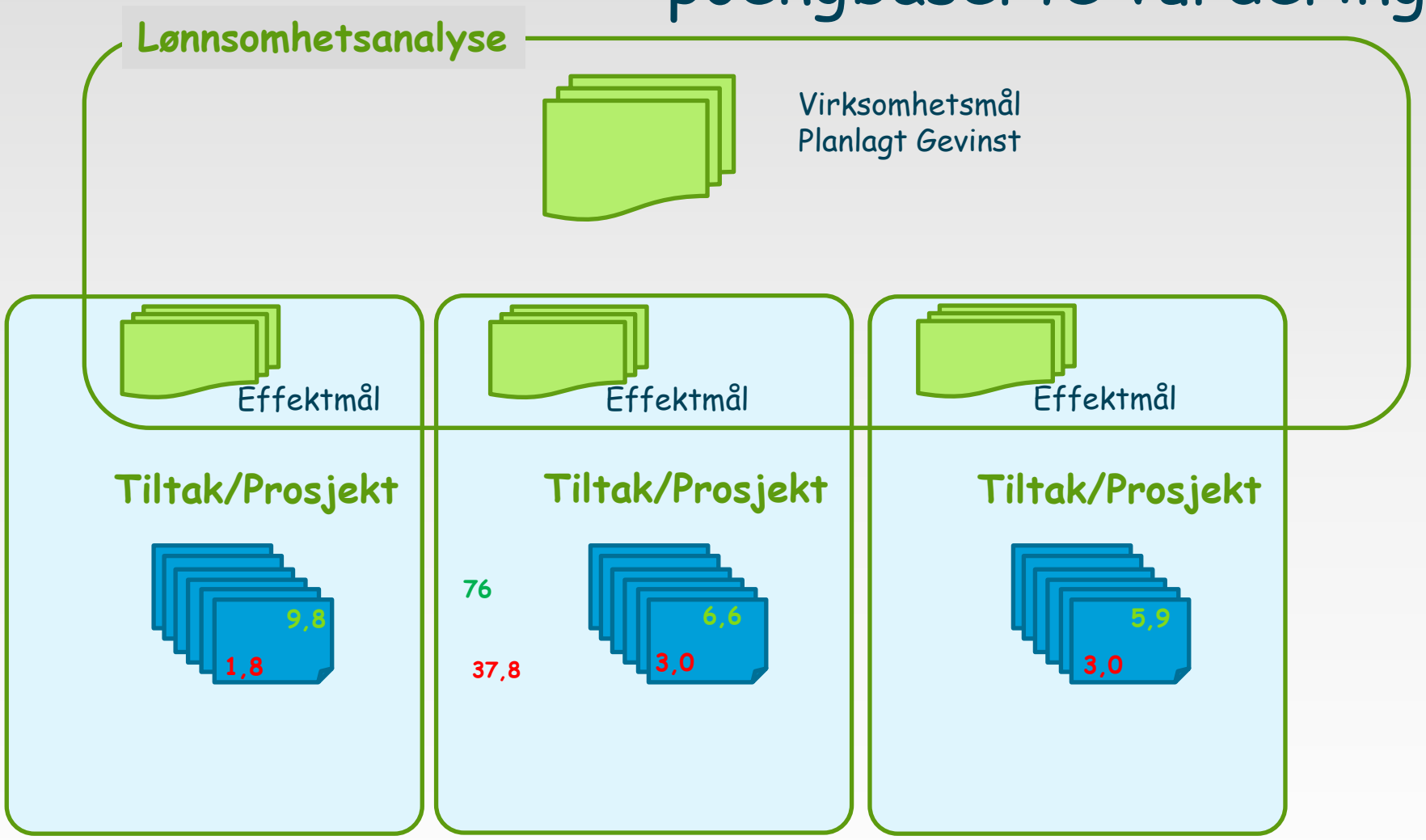
Poeng overalt



Grunnkalkyle (før usikkerhetsvurdering):

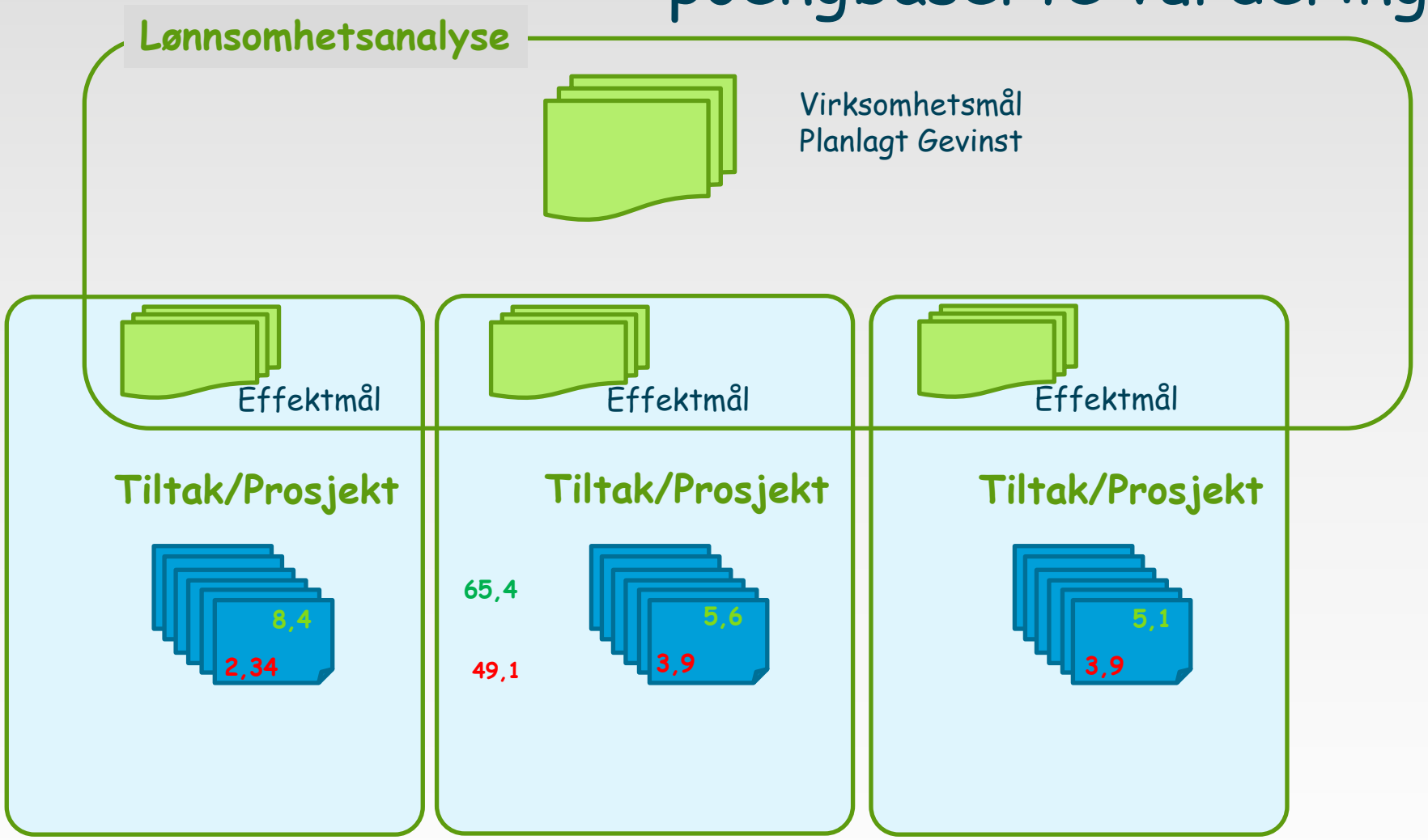
1 nyttepoeng = 0,36 mill
1 kostpoeng = 0,6 mill

Instansiere den relative poengbaserte vurderingen



Grunnkalkyle (før usikkerhetsvurdering):

Instansiere den relative poengbaserte vurderingen



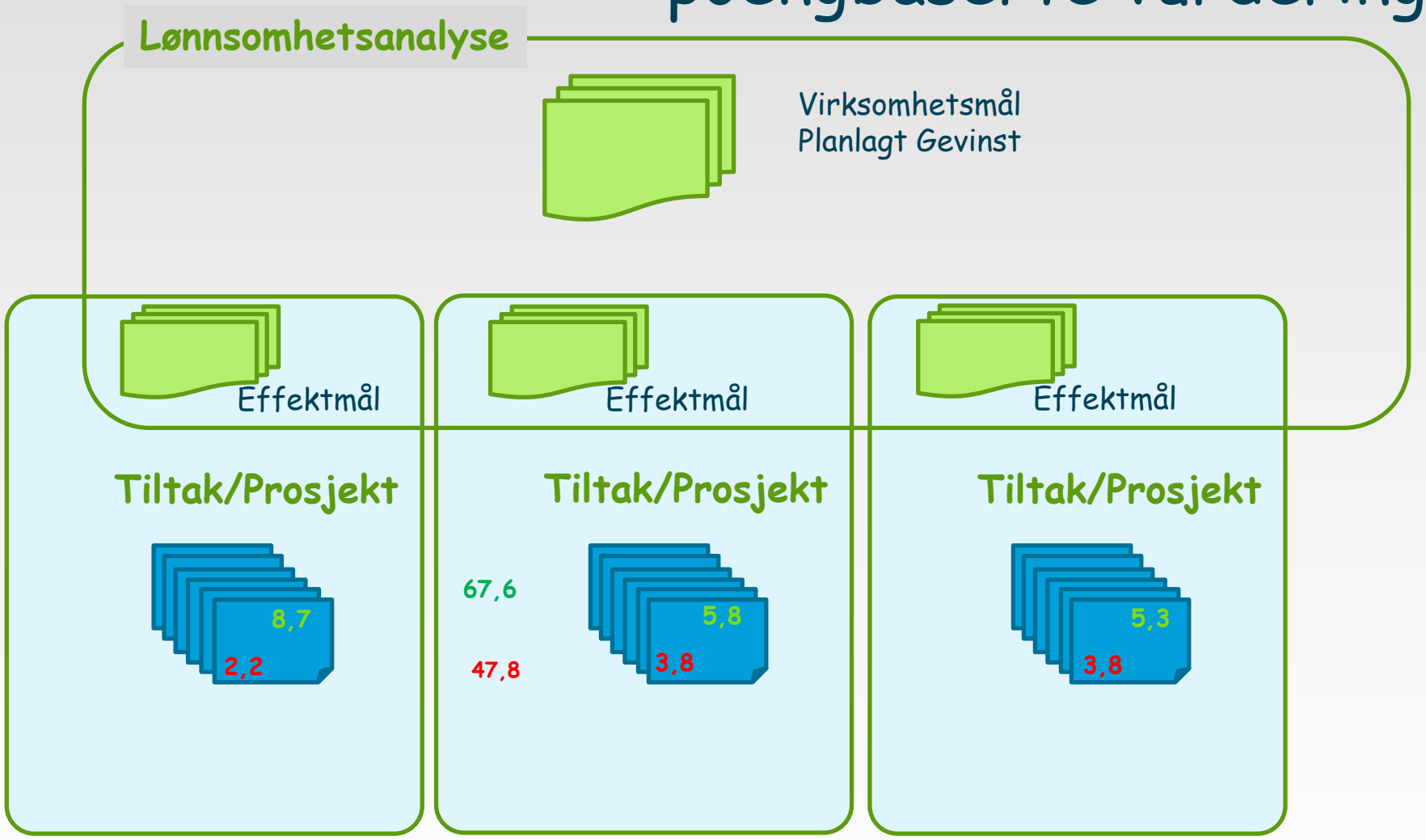
nøytral

P50 nyttepoeng = 0,31 mill

p50 kostpoeng = 0,78 mill

Grunnkalkyle (før usikkerhetsvurdering):

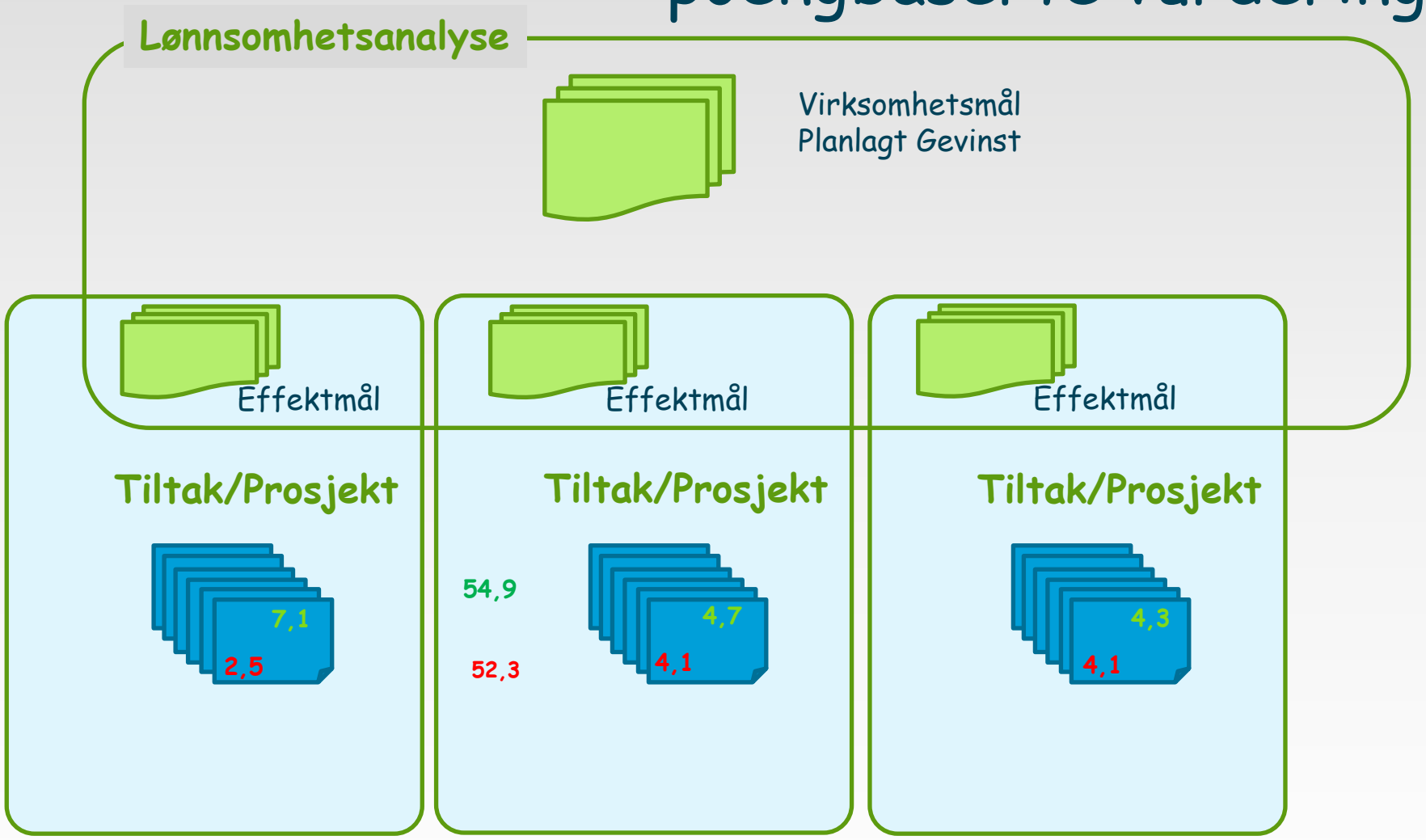
Instansiere den relative poengbaserte vurderingen



Optimistisk
p65 nyttepoeng = 0,32 mill
p35 kostpoeng = 0,76 mill

Grunnkalkyle (før usikkerhetsvurdering):

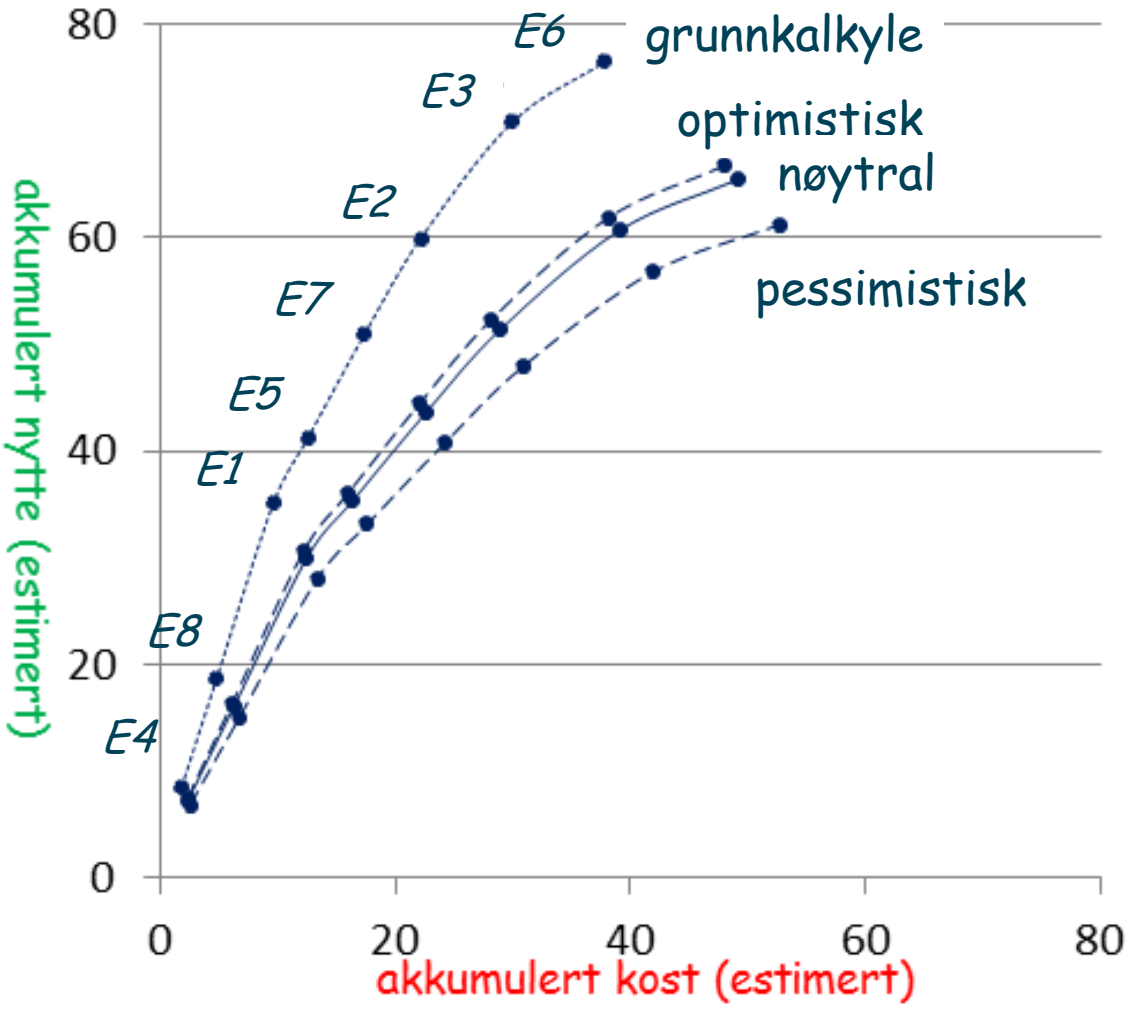
Instansiere den relative poengbaserte vurderingen



Pessimistisk
p15 nyttepoeng = 0,26 mill
p85 kostpoeng = 0,83 mill

Inntjent forretningsverdi (Earned Business Value)

KS2 usikkerhetsregime



KS2 usikkerhetsregime

Sensitivitetsanalyse

Tiltak/Prosjekt

Uten usikkerhetsvurdering
1 nyttepoeng = 0,36 mill
1 kostpoeng = 0,6 mill

Optimistisk
p65 nyttepoeng = 0,32 mill
p35 kostpoeng = 0,76 mill

Nøytral
P50 nyttepoeng = 0,31 mill
p50 kostpoeng = 0,78 mill

Pessimistisk
p15 nyttepoeng = 0,26 mill
p85 kostpoeng = 0,83 mill

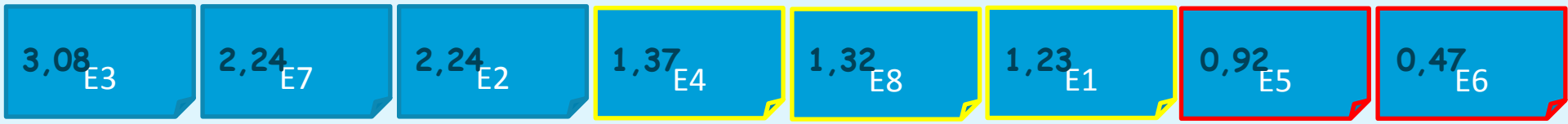
Sensitivitetsanalyse

Tiltak/Prosjekt

Uten usikkerhetsvurdering
1 nyttepoeng = 0,36 mill
1 kostpoeng = 0,6 mill

Optimistisk
p65 nyttepoeng = 0,32 mill
p35 kostpoeng = 0,76 mill

Nøytral
P50 nyttepoeng = 0,31 mill
p50 kostpoeng = 0,78 mill



Pessimistisk
p15 nyttepoeng = 0,26 mill
p85 kostpoeng = 0,83 mill

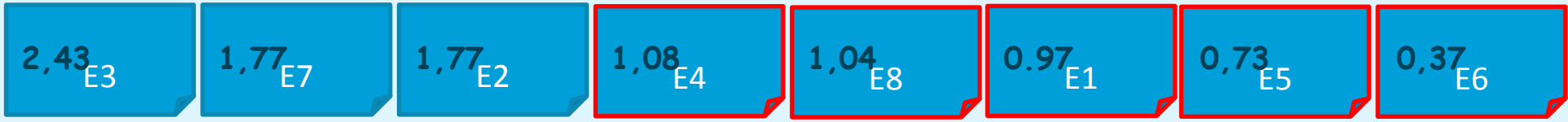
Sensitivitetsanalyse

Tiltak/Prosjekt

Uten usikkerhetsvurdering
1 nyttepoeng = 0,36 mill
1 kostpoeng = 0,6 mill

Optimistisk
p65 nyttepoeng = 0,32 mill
p35 kostpoeng = 0,76 mill

Nøytral
P50 nyttepoeng = 0,31 mill
p50 kostpoeng = 0,78 mill



Pessimistisk
p15 nyttepoeng = 0,26 mill
p85 kostpoeng = 0,83 mill

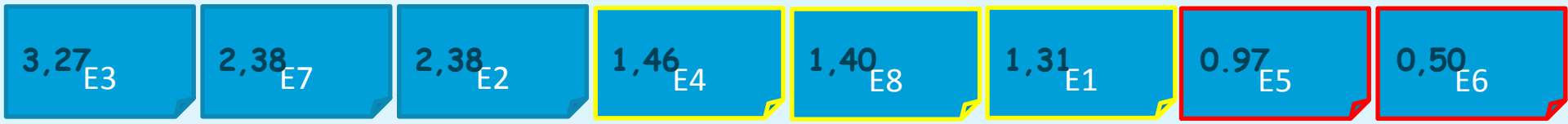
Sensitivitetsanalyse

Tiltak/Prosjekt

Uten usikkerhetsvurdering
1 nyttepoeng = 0,36 mill
1 kostpoeng = 0,6 mill

Optimistisk
p65 nyttepoeng = 0,32 mill
p35 kostpoeng = 0,76 mill

Nøytral
P50 nyttepoeng = 0,31 mill
p50 kostpoeng = 0,78 mill

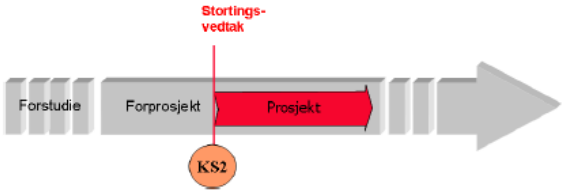


Pessimistisk
p15 nyttepoeng = 0,26 mill
p85 kostpoeng = 0,83 mill

Hvordan får vi fram de ulike realverdiene?

(pessimistisk, nøytral, optimistisk)

Kvalitetssikring av styringsunderlag og kostnadsoverslag (KS2)



Hensikt

Å kvalitetssikre styringsunderlag samt kostnadsoverslag for det valgte prosjektalternativ for prosjektet legges frem til investeringsbeslutning i Stortinget. Dels skal det være en kontroll av om prosjektet er veldefinert med realistiske rammer, dels skal analysen peke fremover ved å kartlegge de styringsmessige utfordringene i gjenstående faser av prosjektet.

Hva som utløser kvalitetssikringen

Forventet prosjektkostnad på over 750 mill. kroner.

Grunnlag for kvalitetssikringen

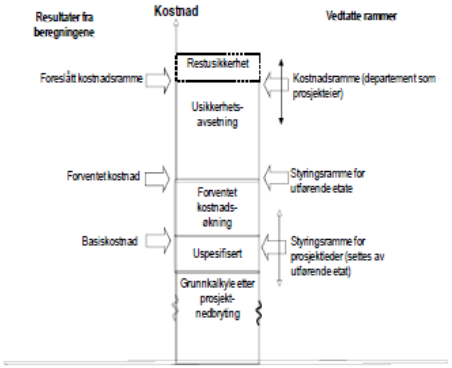
Prosjekter som meldes opp for KS2 skal være fort frem til fullført forprosjekt. Ved oppstart av kvalitetssikringen skal det foreligge:

- Sentralt styringsdokument for prosjektet.
- Et komplett basisestimat for kostnadene og for eventuelle inntekter
- Ferdig utredning av minst to prinsipielt ulike kontraktsstrategier.

Kvalitetssikringens innhold

Kvalitetssikrer skal gi tilråding om:

- Kostnadsramme for prosjektet inklusive nødvendig avsetning for usikkerhet, samt styringsramme for utførende etat.
- Hvordan prosjektet bør styres og organiseres for å sikre en kostnadseffektiv gjennomføring.,



Kilde: Finansdepartementet

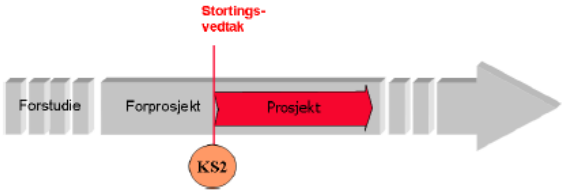
Nærmere om innholdet i kvalitetssikringen:

- Etterse at prosjektkonseptet er veldefinert og tydelig avgrenset, samt at prosjektet er viderefort i tråd med forutsetningene fra KS1. Vurdere om det har skjedd endringer i forutsetningene som kan påvirke anbefalingene fra KS1.
- Vurdere om elementene i det sentrale styringsdokumentet gir et tilstrekkelig grunnlag for styring av prosjektet, samt estimering og usikkerhetsvurdering.
- Kontrollere kostnadsestimatet (komplett, realistisk, transparent).
- Gjennomgå utredningen av kontraktstrategier.
- Kartlegge suksessfaktorer og fallgruber.
- Gjennomgå prosjektets usikkerhetsbilde, med hovedfokus på kostnadene. Herunder:
  - Estimatusikkerhet. Usikkerhet i kostnader for enkeltelementer i estimatet og estimatet som helhet.
  - Hendelsesusikkerheter. Eksterne hendelser som enten inntreffer eller ikke inntreffer.
  - Reduksjon av risiko. Vurdere hvilken mulighet prosjektet har til å påvirke usikkerheten, samt gevinster og kostnader ved risikoreduserende tiltak.
  - Potensialet for kostnadsreduserende forenklinger og reduksjoner i omfang vurderes særskilt ("kuttliste").
- Gi en samlet tilråding om kostnadsramme og styringsramme for utførende etat. Kostnads-rammen er det nivå Stortinget inviteres til å vedta, og settes normalt lik P85 minus kuttliste. Styringsrammen er det nivå utøvende etat forventes å levere for, normalt P50.
- Gi en tilråding om organisering og styring av prosjektet, herunder valg av kontraktsstrategi. Styring av usikkerhetsavsetningen er en særlig problemstilling, og kan tilsi behov for supplerende incitamenter.

Tidspunkt for kvalitetssikringen

Prosjektet skal ha gjennomgått KS2 forut for at det fremlegges for Stortinget for endelig investeringsbeslutning. Dette faller normalt sammen med avslutning av forprosjektfasen.

Kvalitetssikring av styringsunderlag og kostnadsoverslag (KS2)



Hensikt

Å kvalitetssikre styringsunderlag samt kostnadsoverslag for det valgte prosjektalternativ før prosjektet legges frem til investeringsbeslutning i Stortinget. Dels skal det være en kontroll av om prosjektet er veldefinert med realistiske rammer, dels skal analysen peke fremover ved å kartlegge de styringsmessige utfordringene i gjenstående faser av prosjektet.

Hva som utløser kvalitetssikringen

Forventet prosjektkostnad på over 750 mill. kroner.

Grunnlag for kvalitetssikringen

Prosjekter som meldes opp for KS2 skal vær… frem til fullført forprosjekt. Ved opp… kvalitetssikringen skal det foreligg…

- Sentralt styringsdokument for…
- Et komplett basisestimat fo… eventuelle inntekter
- Ferdig utredning av mi… kontraktsstrategier.

Kvalitetssikringe… …nh…

Kvalitetssikrer skal gi f… …g om…

- Kostnadsramme f… …jektet nødvendig avsetn… …usikk… styringsramme fo… …ende e…
- Hvordan prosjek… …styres å sikre en kostna… …ktiv gje…

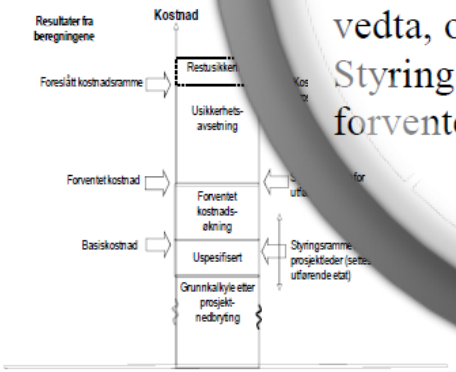
Nærmere om innholdet i kvalitetssikringen:

- Etterse at prosjektkonseptet er veldefinert og tydelig avgrenset, samt at prosjektet er viderefort i tråd med forutsetningene fra KS1. Vurdere om det har skjedd endringer i forutsetningene som kan påvirke anbefalingene fra KS1.
- Vurdere…
- …

usikkerheten, samt gevinster og kost… ved risikoreduserende tiltak.

- Potensialet for kostnadsreduserende forenklinger og reduksjoner i omfang vurderes særskilt (”kuttliste”).

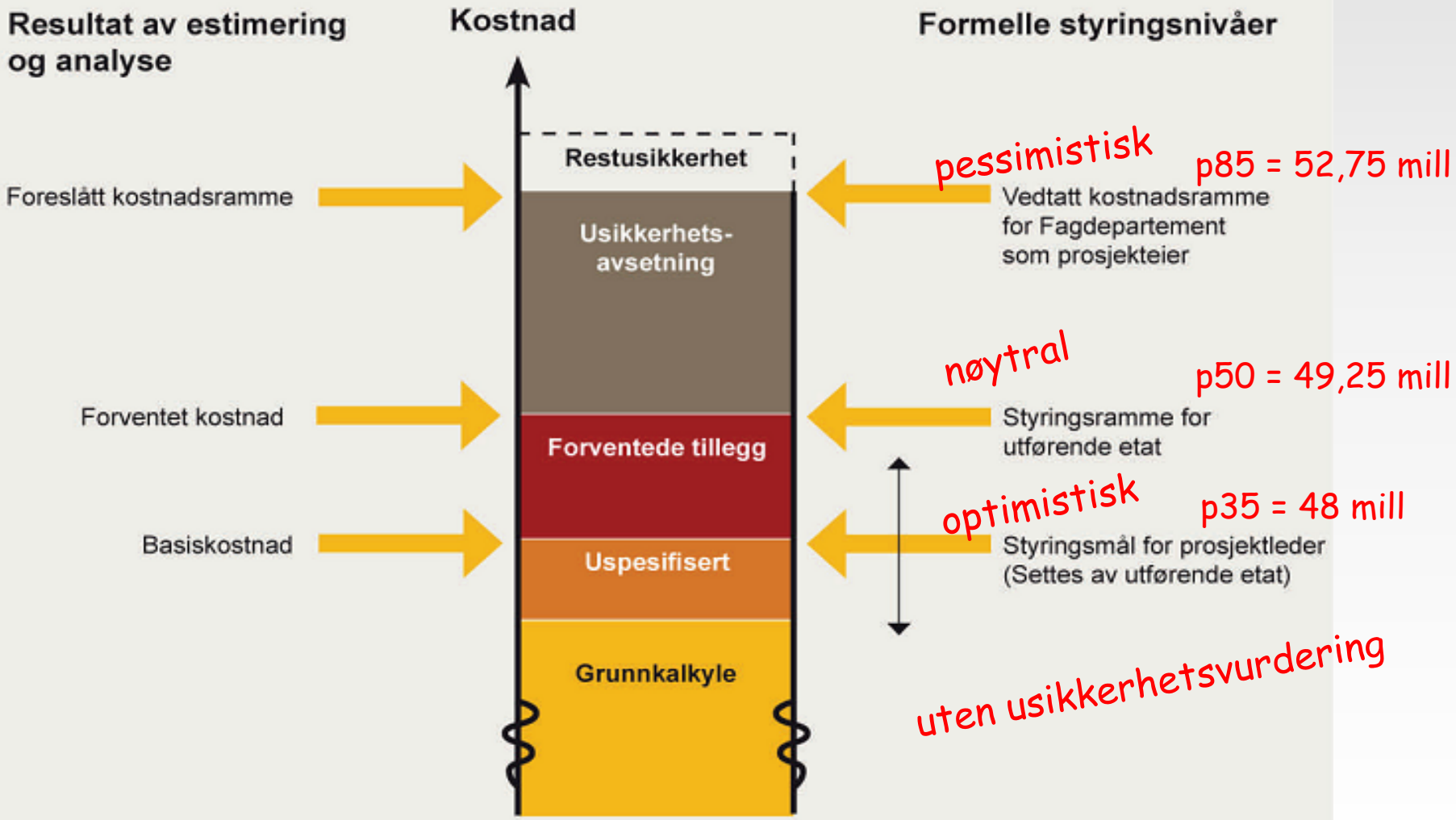
Gi en samlet tilråding om kostnadsramme og styringsramme for utførende etat. Kostnads-rammen er det nivå Stortinget inviteres til å vedta, og settes normalt lik P85 minus kuttliste. Styringsrammen er det nivå utøvende etat forventes å levere for, normalt P50.



Kilde: Finansdepartementet

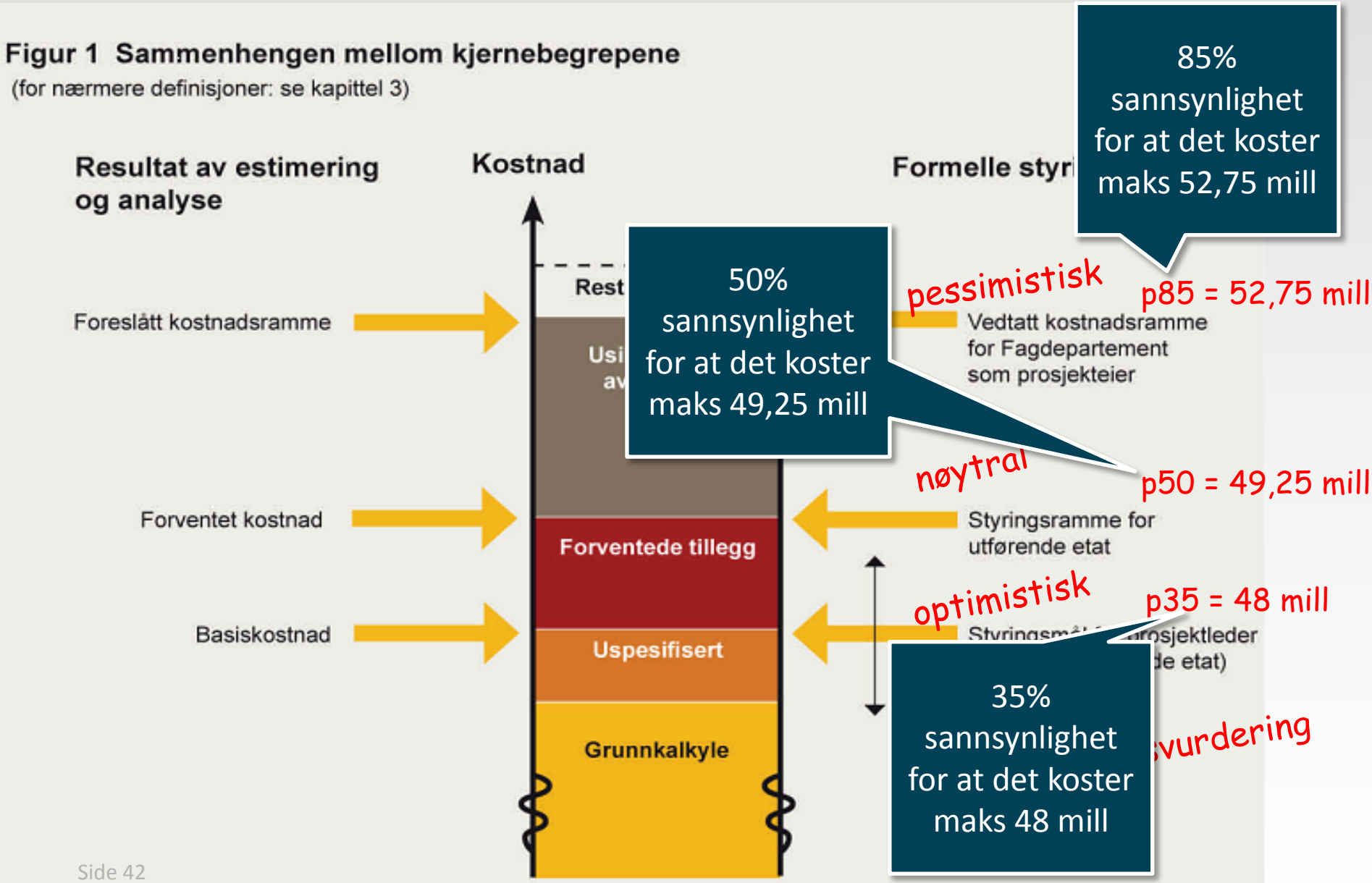
KS2 usikkerhetsregime for kost

Figur 1 Sammenhengen mellom kjernebegrepene
(for nærmere definisjoner: se kapittel 3)

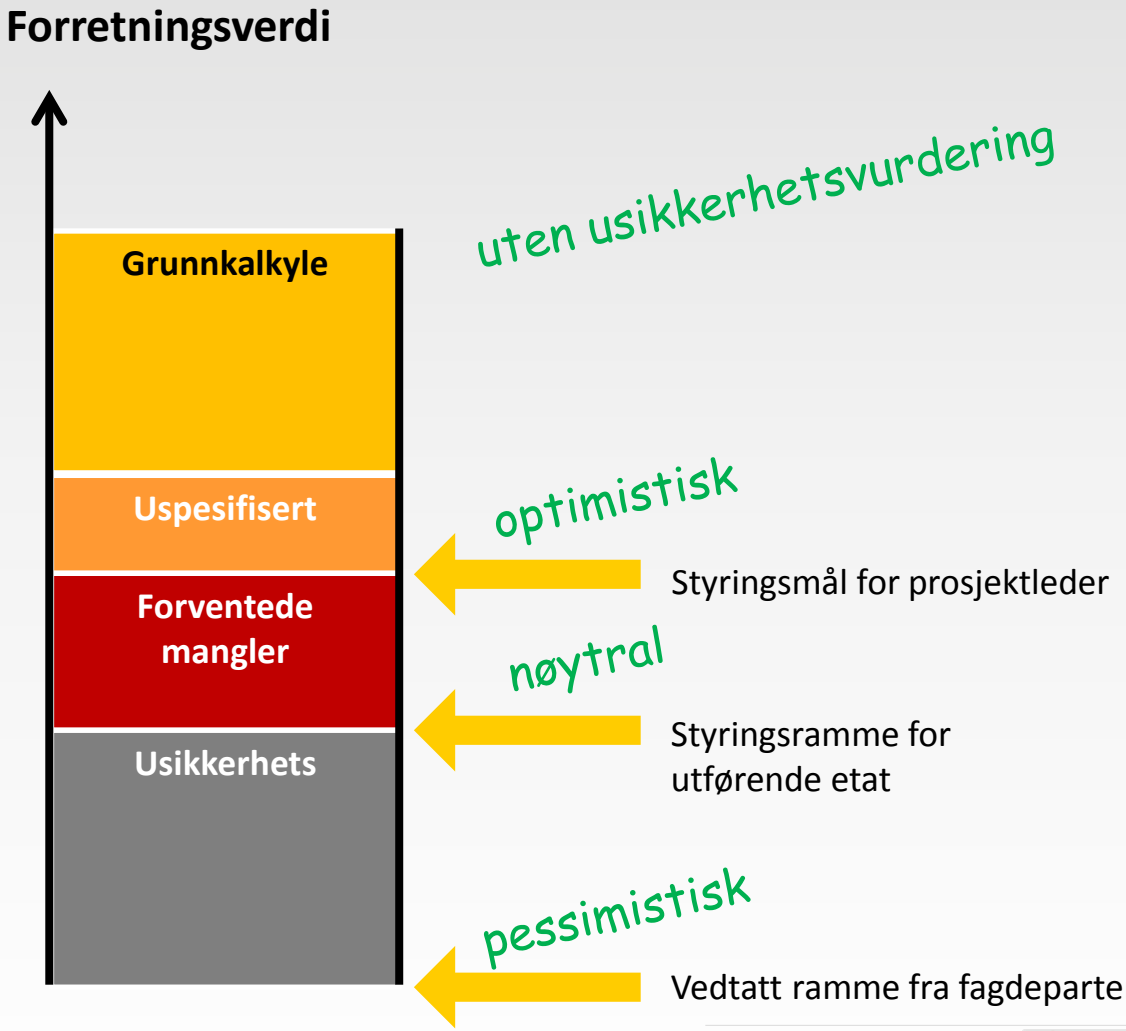


pX-tallene er generert ved Monte Carlo-simulering der kostestimatene er antatt uavhengige

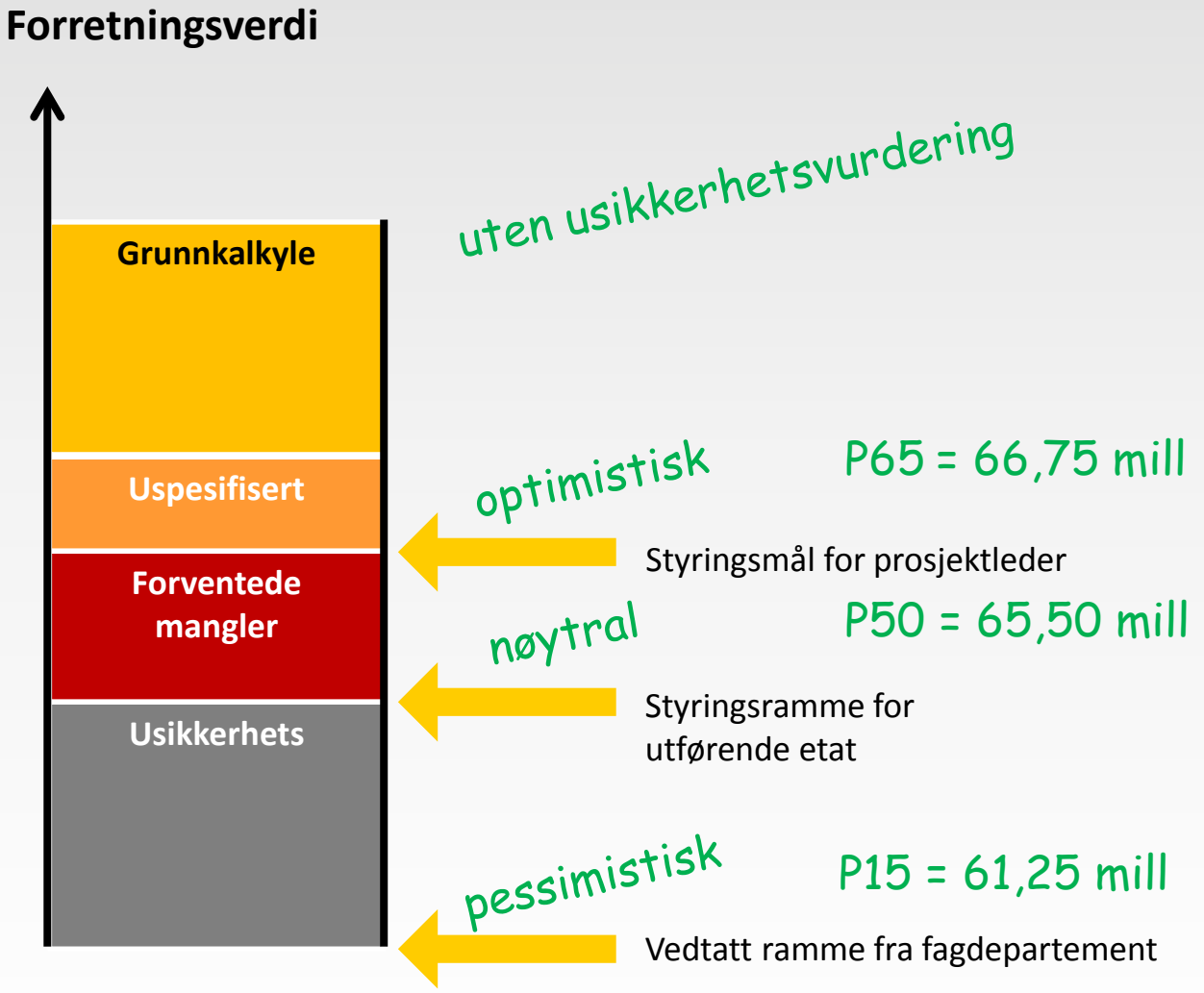
KS2 usikkerhetsregime for kost



KS2 usikkerhetsregime for forretningsverdi

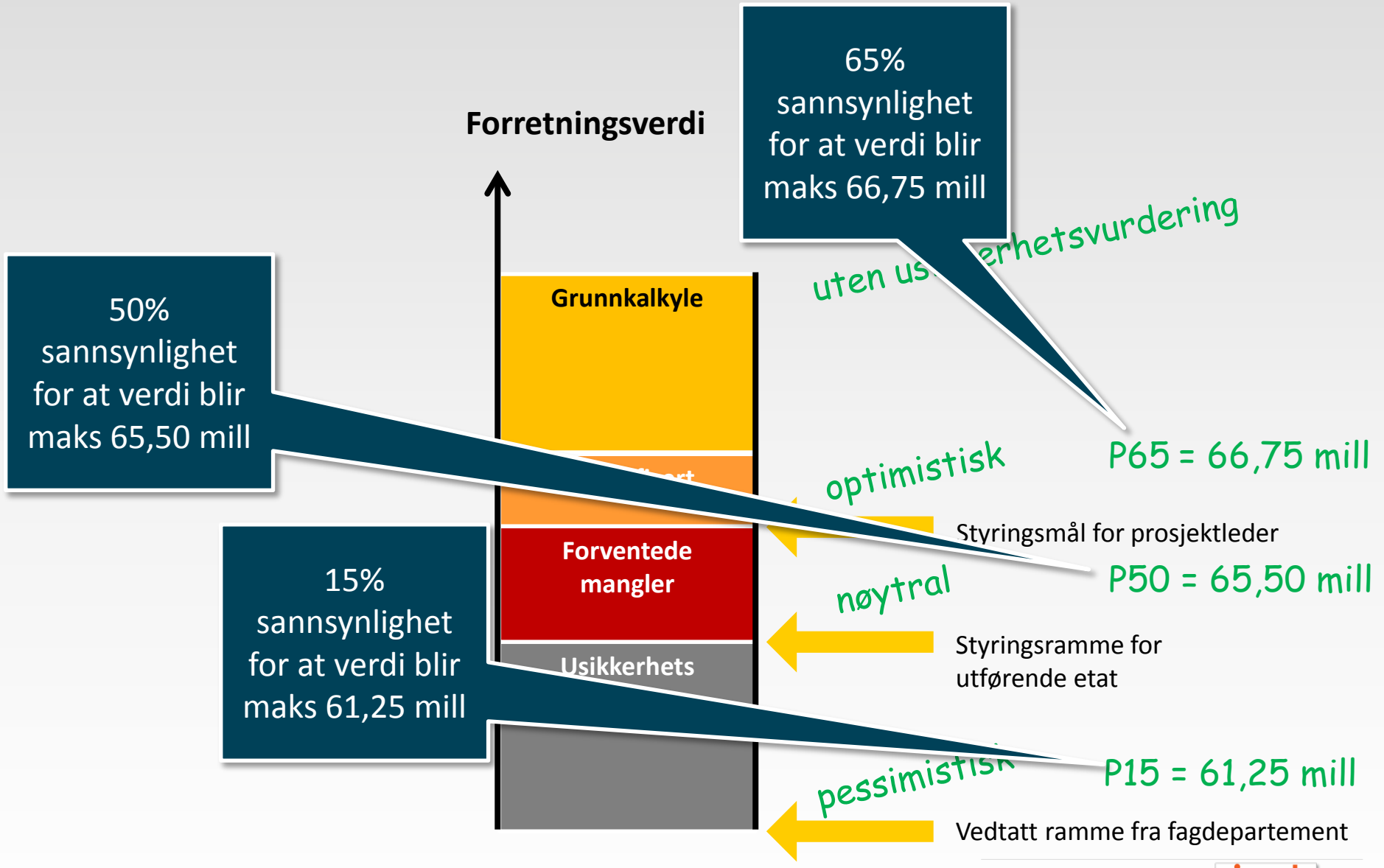


KS2 usikkerhetsregime for forretningsverdi



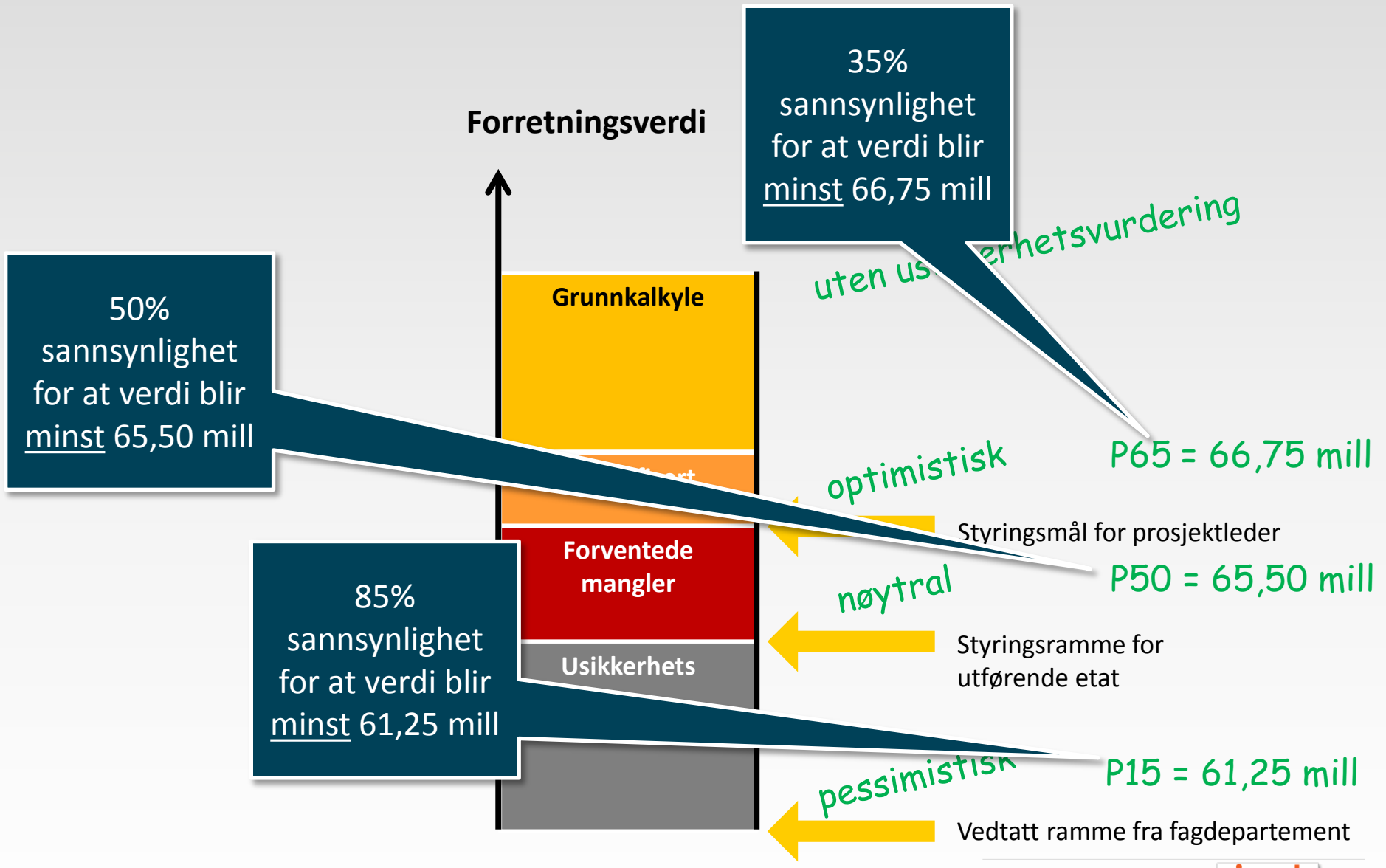
pX-tallene er generert ved Monte Carlo-simulering der nyttetimatene er antatt uavhengige

KS2 usikkerhetsregime for forretningsverdi



simula
Suksess med IT i offentlig sektor (SMIOS)

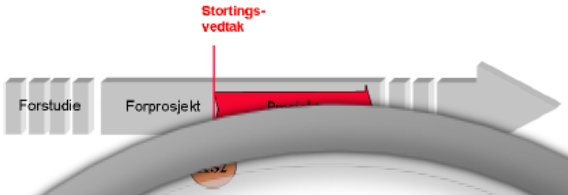
KS2 usikkerhetsregime for forretningsverdi



simula
Suksess med IT i offentlig sektor (SMIOS)

Hvordan får vi fram pX-verdier?

Kvalitetssikring av styringsunderlag og kostnadsoverslag (KS2)



Hensikt

Å kvalitetssikre styringsunderlag samt kostnadsoverslag for det valgte prosjekt
prosjektet legges frem til investering
Stortinget. Dels skal det være en k
prosjektet er veldefinert med rea
dels skal analysen peke fremov
styringsmessige utfordringen
prosjektet.

Hva som utløser k

Forventet prosjektkostn

Grunnlag for kv

Prosjekter som melde
frem til fullført forpr
kvalitetssikringen sk

- Sentralt styrings
- Et komplett basi
eventuelle inntek
- Ferdig utredning
kontraktsstrategie

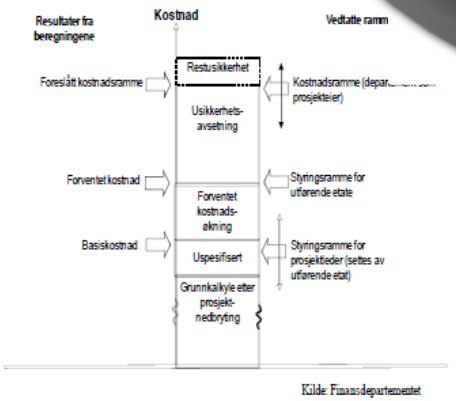
Kvalitetssikringe

Kvalitetssikrer skal gi tilr

- Kostnadsramme for pr
nødvendig avsetning fo
styringsramme for utfo
- Hvordan prosjektet bør styre
å sikre en kostnadseffektiv gje

Gjennomgå prosjektets usikkerhetsbilde, med hovedfokus på kostnadene. Herunder:

- Estimatusikkerhet. Usikkerhet i kostnader for enkeltelementer i estimatet og estimatet som helhet.
- Hendelsesusikkerheter. Eksterne hendelser som enten inntreffer eller ikke inntreffer.



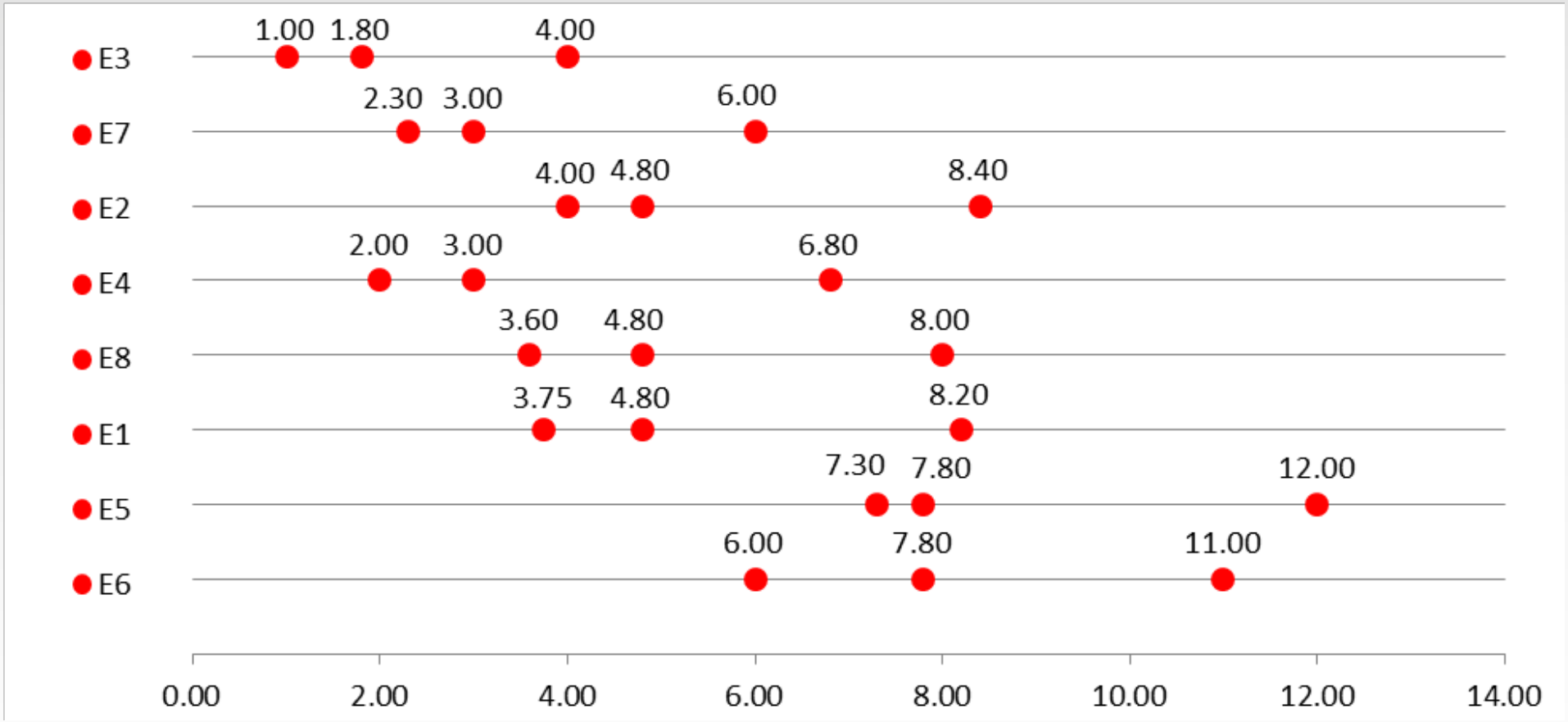
Kilde: Finansdepartementet

- Gi en tilråding om organisering og styring av prosjektet, herunder valg av kontraktsstrategi. Styring av usikkerhetsavsetningen er en særlig problemstilling, og kan tilsi behov for supplerende incitamenter.

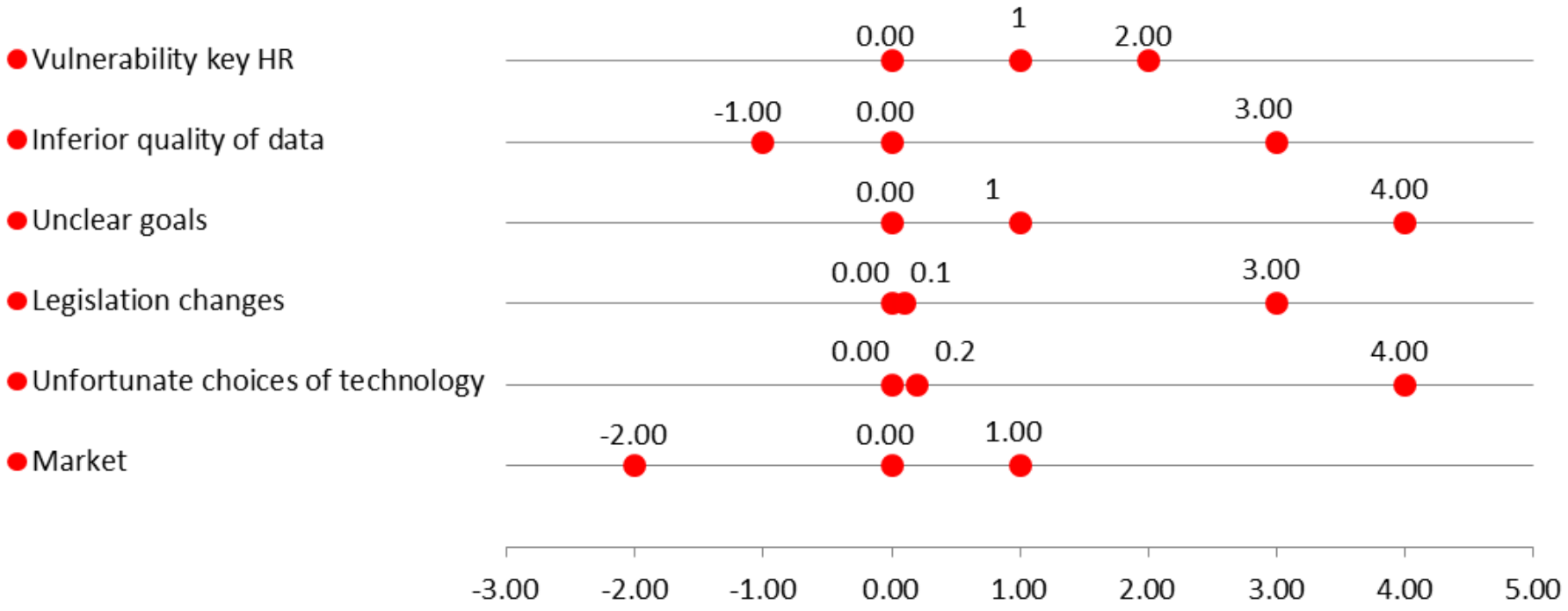
Tidspunkt for kvalitetssikringen

Prosjektet skal ha gjennomgått KS2 forut for at det fremlegges for Stortinget for endelig investeringsbeslutning. Dette faller normalt sammen med avslutning av forprosjektfasen.

Estimatusikkerhet kost trepunktestimater

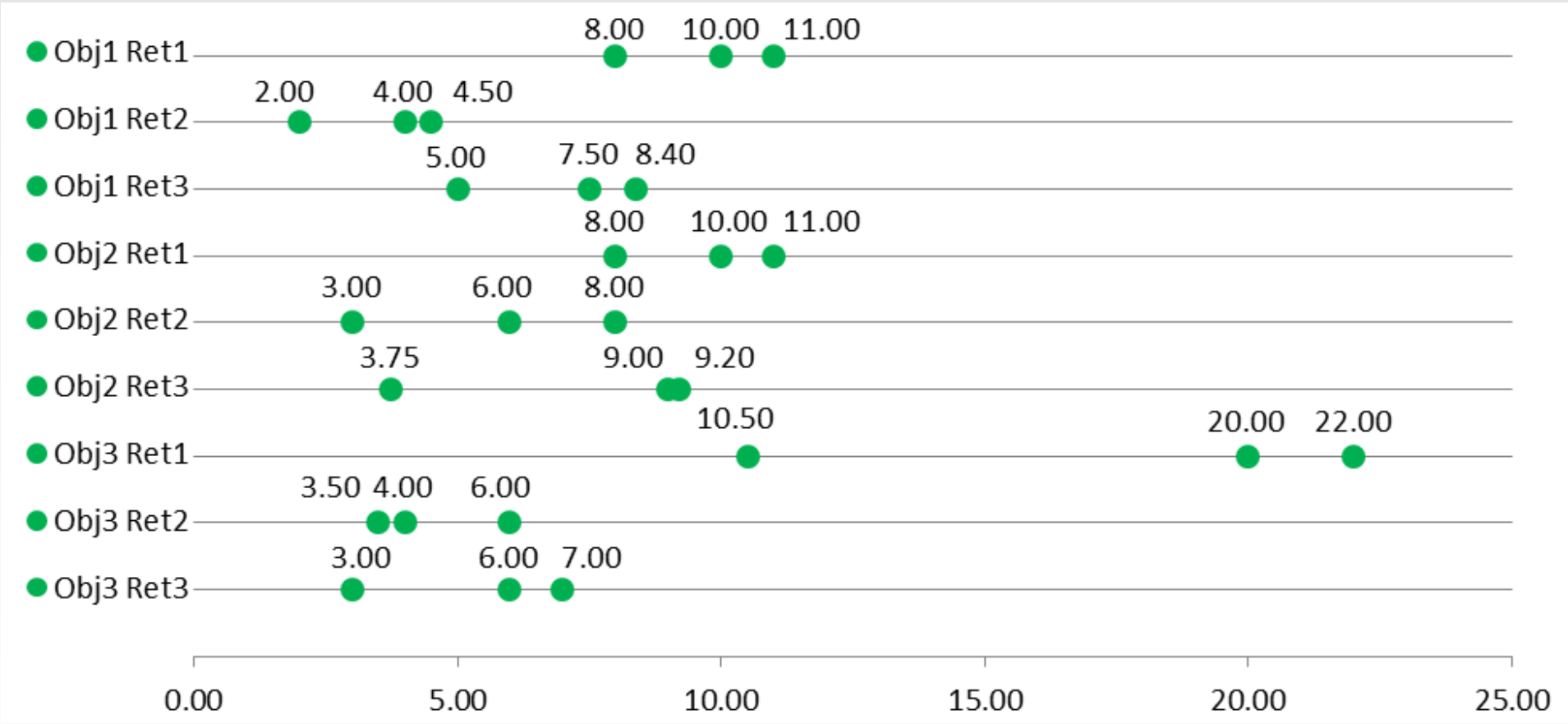


Hendelsesusikkerhet kost trepunktestimater

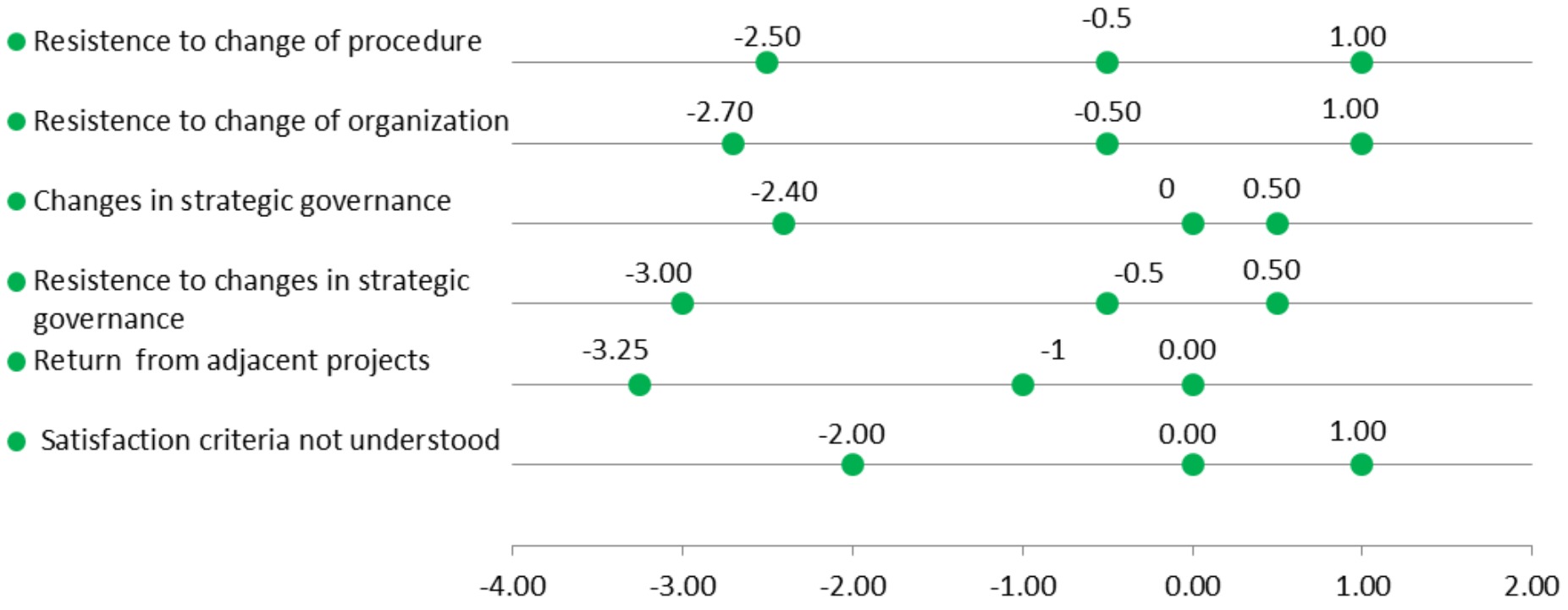


simula
Suksess med IT i offentlig sektor
(SMIOS)

Estimatusikkerhet nytte trepunktestimater

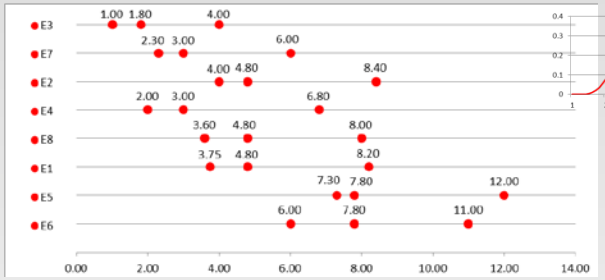


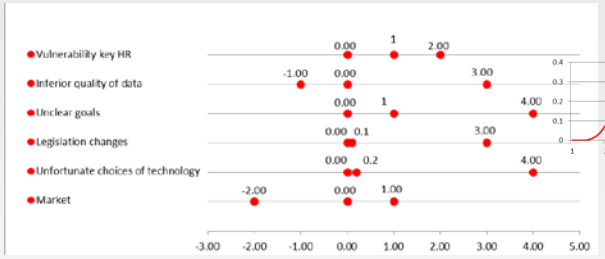
Hendelsesusikkerhet nytte trepunktestimater

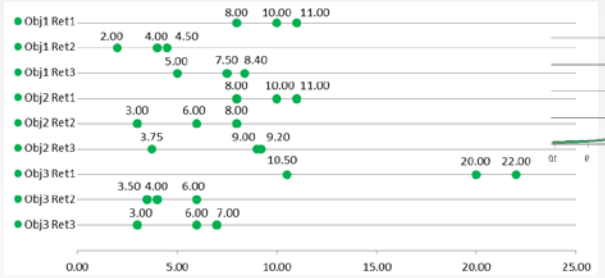


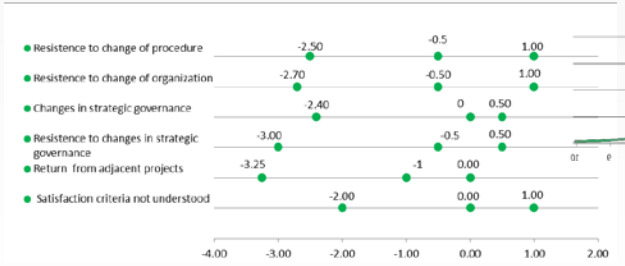
simula
Suksess med IT i offentlig sektor (SMIOS)

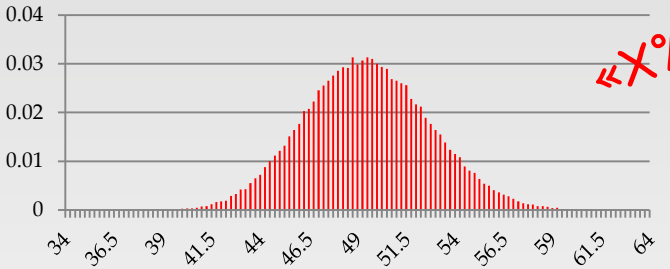
Simulere prosjektløp







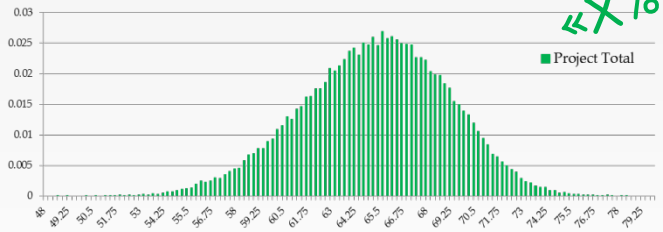




60 000 prosjektløp

«X% sjanse for ...»

pX



60 000 prosjektløp

«X% sjanse for ...»

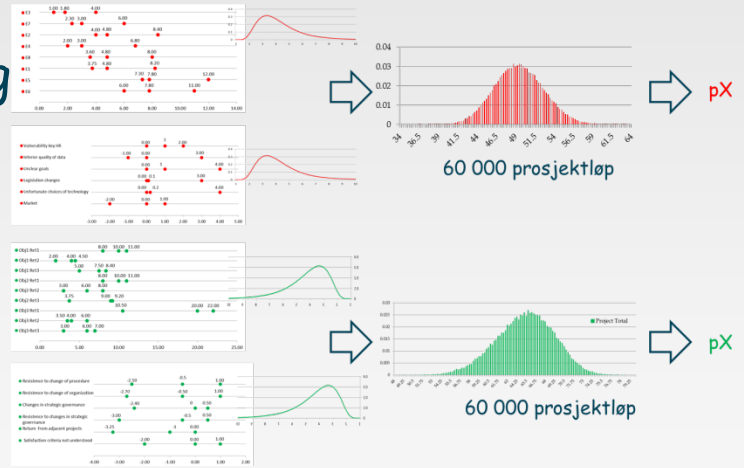
pX

simula
Suksess med IT i offentlig sektor (SMIOS)

Oppsummering

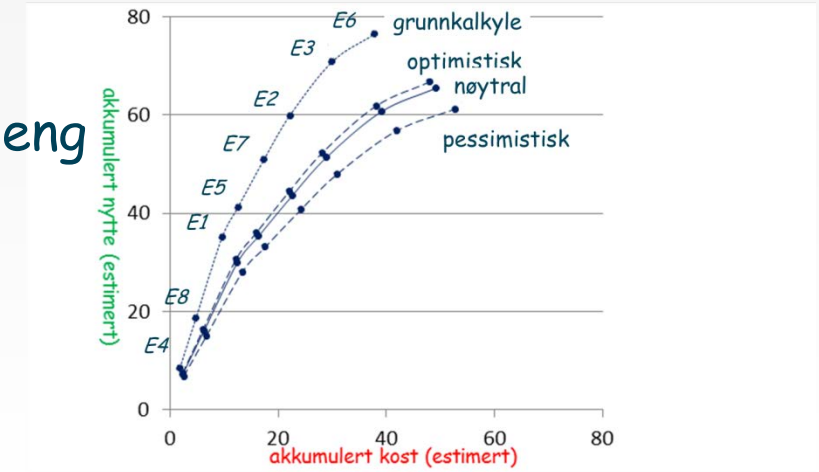
- Nyttepoeng i tillegg til kostpoeng!

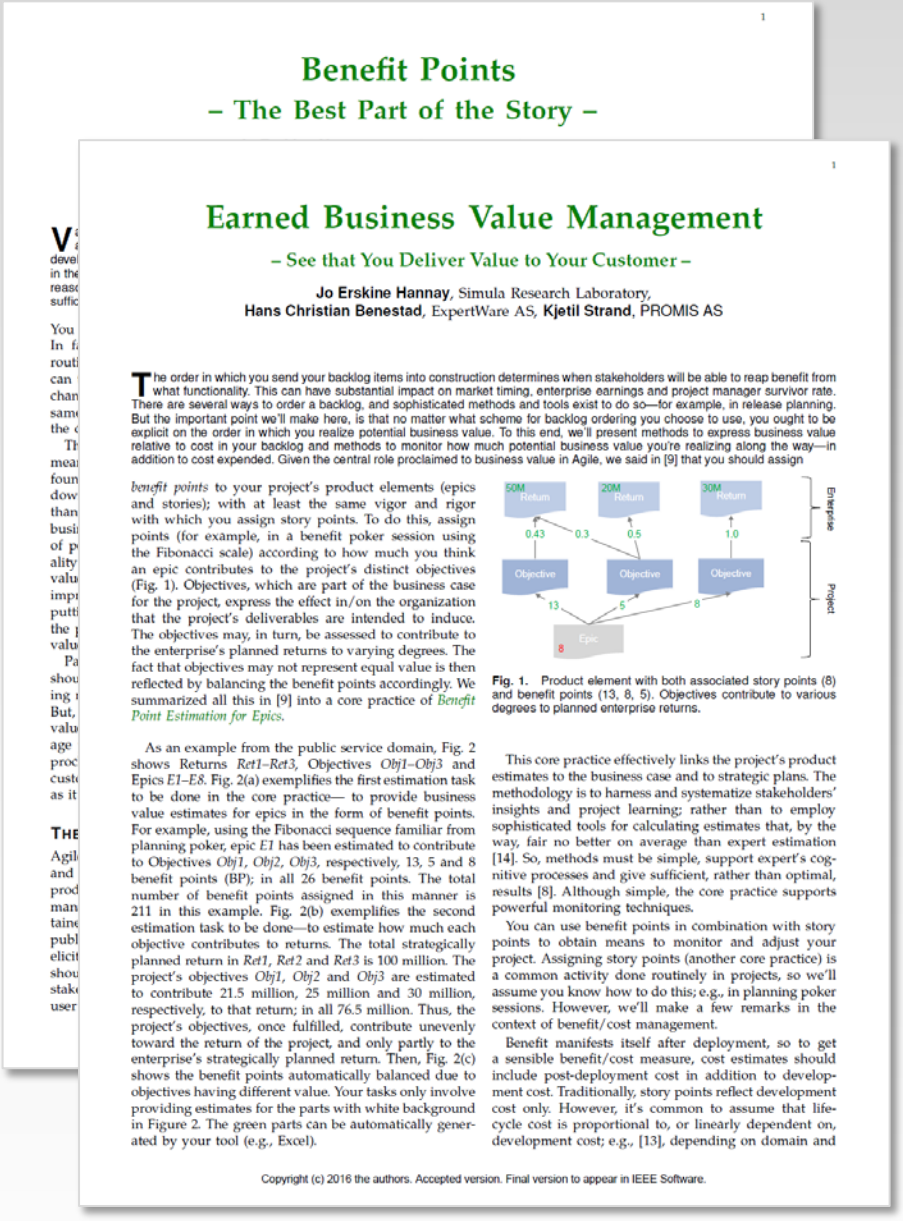
- Bruk metoder for usikkerhetsvurdering også på nytte (forretningsverdi)!



- Fra grunnkalkylen, generer alternative verdier som reflekterer usikkerhetsvurderingene…

- …og plugg dem inn i kost- og nyttepoeng for å få usikkerhetsbaserte styringsrammer for både kost og forretningsverdi!





Benefit Points

– The Best Part of the Story –

Earned Business Value Management

– See that You Deliver Value to Your Customer –

Jo Erskine Hannay, Simula Research Laboratory,
Hans Christian Benestad, ExpertWare AS, **Kjetil Strand**, PROMIS AS

The order in which you send your backlog items into construction determines when stakeholders will be able to reap benefit from what functionality. This can have substantial impact on market timing, enterprise earnings and project manager survivor rate. There are several ways to order a backlog, and sophisticated methods and tools exist to do so—for example, in release planning. But the important point we'll make here, is that no matter what scheme for backlog ordering you choose to use, you ought to be explicit on the order in which you realize potential business value. To this end, we'll present methods to express business value relative to cost in your backlog and methods to monitor how much potential business value you're realizing along the way—in addition to cost expended. Given the central role proclaimed to business value in Agile, we said in [9] that you should assign

benefit points to your project's product elements (epics and stories); with at least the same vigor and rigor with which you assign story points. To do this, assign points (for example, in a benefit poker session using the Fibonacci scale) according to how much you think an epic contributes to the project's distinct objectives (Fig. 1). Objectives, which are part of the business case for the project, express the effect in/on the organization that the project's deliverables are intended to induce. The objectives may, in turn, be assessed to contribute to the enterprise's planned returns to varying degrees. The fact that objectives may not represent equal value is then reflected by balancing the benefit points accordingly. We summarized all this in [9] into a core practice of *Benefit Point Estimation for Epics*.

Fig. 1. Product element with both associated story points (8) and benefit points (13, 8, 5). Objectives contribute to various degrees to planned enterprise returns.

As an example from the public service domain, Fig. 2 shows Returns *Ret1–Ret3*, Objectives *Obj1–Obj3* and Epics *E1–E8*. Fig. 2(a) exemplifies the first estimation task to be done in the core practice— to provide business value estimates for epics in the form of benefit points. For example, using the Fibonacci sequence familiar from planning poker, epic *E1* has been estimated to contribute to Objectives *Obj1*, *Obj2*, *Obj3*, respectively, 13, 5 and 8 benefit points (BP); in all 26 benefit points. The total number of benefit points assigned in this manner is 211 in this example. Fig. 2(b) exemplifies the second estimation task to be done—to estimate how much each objective contributes to returns. The total strategically planned return in *Ret1*, *Ret2* and *Ret3* is 100 million. The project's objectives *Obj1*, *Obj2* and *Obj3* are estimated to contribute 21.5 million, 25 million and 30 million, respectively, to that return; in all 76.5 million. Thus, the project's objectives, once fulfilled, contribute unevenly toward the return of the project, and only partly to the enterprise's strategically planned return. Then, Fig. 2(c) shows the benefit points automatically balanced due to objectives having different value. Your tasks only involve providing estimates for the parts with white background in Figure 2. The green parts can be automatically generated by your tool (e.g., Excel).

This core practice effectively links the project's product estimates to the business case and to strategic plans. The methodology is to harness and systematize stakeholders' insights and project learning; rather than to employ sophisticated tools for calculating estimates that, by the way, fair no better on average than expert estimation [14]. So, methods must be simple, support expert's cognitive processes and give sufficient, rather than optimal, results [8]. Although simple, the core practice supports powerful monitoring techniques.

You can use benefit points in combination with story points to obtain means to monitor and adjust your project. Assigning story points (another core practice) is a common activity done routinely in projects, so we'll assume you know how to do this; e.g., in planning poker sessions. However, we'll make a few remarks in the context of benefit/cost management.

Benefit manifests itself after deployment, so to get a sensible benefit/cost measure, cost estimates should include post-deployment cost in addition to development cost. Traditionally, story points reflect development cost only. However, it's common to assume that life-cycle cost is proportional to, or linearly dependent on, development cost; e.g., [13], depending on domain and

For å lære mer

- Kontakt oss!

johannay@simula.no
kjetil.strand@verdix.no
benestad@expertware.no

- Ta kurset IT Project Professional ITPP (Metier Academy)
- prosjektmetodikken PRINCE2®
- sammen med smidige teknikker og
- beste praksis for kontraktshåndtering.

46 PDU til PMP-sertifiserte fra Project Management Institute

http://www.smidigeprosjekter.no/itpp

http://simula.no - søk på «Hannay» -> publications
Http://hitledelse.no – Suksess med IKT i offentlig sektor -> Publikasjoner

simula
Suksess med IT i offentlig sektor (SMIOS)

Det var det

Neste gang: Periodisering