ETSN15 2024
Requirements Engineering

Lecture 5+6:

Help you to focus your reading of 6 papers:

Prototyping [PROTO]
Agile RE [AGRE]
Market-drive RE [MDRE]
Reqts interdependencies [INTDEP]
Release Planning [RP]
RE and Open Source [OSSRE]

Björn Regnell
http://www.cs.lth.se/krav

Requirements change...

- ...constantly! Sometimes very quickly!
- **Why?**
  - We learn (the whole point of RE...)
  - Changed needs and priorities
  - Disruptive new technology
  - Fierce competition
  - ...
- **What to do about it?**
  - *reduce uncertainty:* **do prototyping**
  - *live with it:* **try to be reasonably agile**

The Practice of Prototyping [PROTO]

Prototyping

use of a prototype to *explore, communicate, and evaluate potential solutions*

Prototype

early ***sample, model, or release****, which* ***simulates*** *aspects of the final product and enables cost effective* ***testing with real users***

- ❑ Communicate
- ❑ Validate
- ❑ Elicit

requirements, goals, ideas, priorities, …

PURPOSE, SCOPE, USE, STRATEGY

Prototyping: major risk

- Prototype code forced into production code
  - “We need it now – just release it!”
  - “...but it was developed for another purpose :(“
- Prototype code is not production grade quality => will lead to technical debt

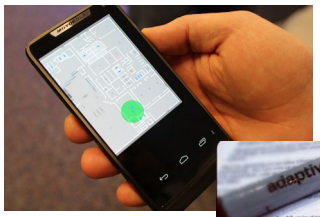
PURPOSE of Prototyping [PROTO]

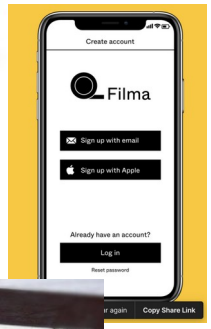
PURPOSE
Exploration & learning
Communication: sales, alignment
Incremental development
Quality improvement
Validation & Testing - Fit: Problem-solution, Product-market - Technical feasibility - Usability testing

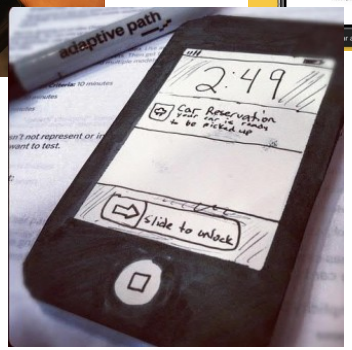
Advice:

- Consider your **purposes** with prototyping and for each prototyping instance
- Select **scope, media,** and **use** of prototype to match purpose

SCOPE & MEDIA of Prototype [PROTO]







SCOPE
Breadth of functionality
Functional refinement
Visual appearance
Interactive & haptic behaviour
Data realism

MEDIA
Sketch: paper or computer-based
Wireframe: paper or computer-based
Mock-up: paper or computer-based
Source-code software
Other: video, interview

Advice:

- Consider which **functional breadth** and **refinement**, **visual appearance** and **interactive behaviour** that is needed for your **purposes**
- Balance the **costs** of prototype building (affected by Scope and Media) against possible **benefits**

USE of Prototype [PROTO]

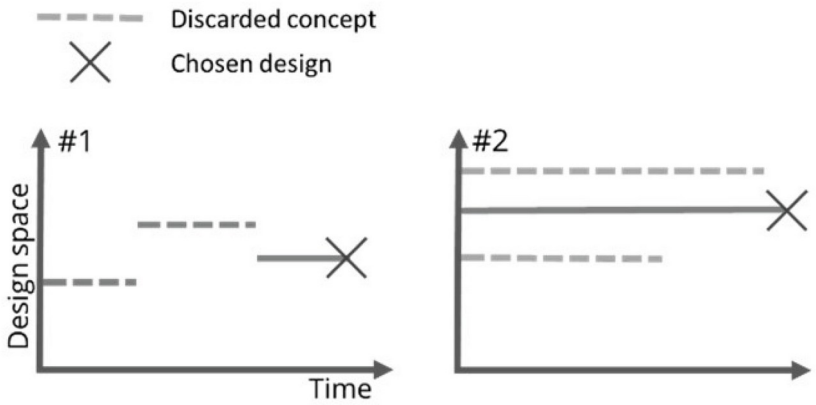
USE of prototype
Reviewers: internal, FFF (family-foes-friends), external
Prototype interaction: yes, no (demo)
Review approach: scenario-based, free
Usage environment

Advice:

- Select **reviewers** that represent **stakeholders** and **user categories** that can provide feedback needed for chosen **purposes**
- Design **review approach** and **interaction** to align with **purpose** and focus of prototyping
- Select **usage environment** to match **purpose**

Exploration STRATEGY [PROTO]

Iteration focus: Business, product, feature, optimization
Decide on lagom iteration size



[Tronvoll et al. 2017]

Single vs **Parallel exploration**

Advice

- Consider the size of potential solution space and select suitable type of **exploration** and **iteration size**
- In early stages, consider **parallel exploration**, when more certain, switch to **single exploration**
- Match **prototype scope**, **media** and **use** to the iteration focus, and align with purpose

Summary of Prototyping Aspects [PROTO]

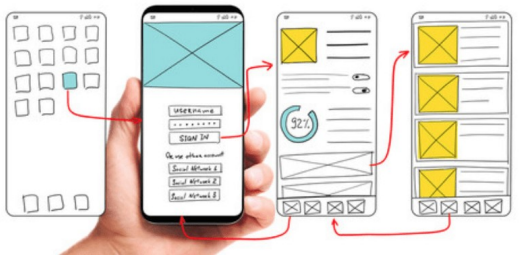
PURPOSE of Prototyping
- Exploration & learning
- Communication: sales, alignment
- Incremental development
- Quality improvement
  - Validation & Testing
    problem-solution / product-market fit, technical feasibility, usability testing

SCOPE of Prototype
- Breadth of functionality
- Functional refinement
  - Visual appearance
  - Interactive & haptic behaviour
  - Data realism

Prototype MEDIA
- Sketch: paper or computer-based
- Wireframe: paper or computer-based
- Mock-up: paper of computer-based
- Source-code software
- Other: video, interview

USE of prototype
- Reviewers: internal, FFF, external
- Prototype interaction: yes, no (demo)
- Review approach: scenario-based free
- Usage environment

Exploration STRATEGY
- Single vs parallel exploration
- Iteration focus: Business, product, feature, optimisation
- Iteration size



Optional Video on Prototyping in practice from 2022:

Available in Canvas.

Hampus Jakobsson

Angel investor in > 100 companies

Now: Pale Blue Dot, $100m climate-tech fund

Previous: LTH, built & scaled 2 startups

”We don’t do requirements. We are agile.”

Wrong! Exactly all projects need & have requirements ==

ideas/decisions of what the product should do

In Agile projects, ***some*** requirements ***are*** documented

- as traditional requirements
- as user stories & acceptance criteria
- as backlog entries
- as test cases
- combo of “requirements” and other artifacts

Many requirements are ***NOT*** documented (can be risky)

Underlying assumptions → agile RE

The Agile Manifesto, http://agilemanifesto.org/, 2001

Requirements change...

- ...because of evolution in technology, business, customer needs, ...
- **Therefore**, do *not spend much time* on ***details in initial requirements*** elicitation. Instead, requirements emerge during the development process.

Extensive specification is costly & time consuming

- Developing extensive documentation and models may be counterproductive.
- **Therefore**, do not document requirements in detail upfront.

The Customer (representative) can tell us

- The customer is available for frequent interaction with the developers.
- **Therefore,** rely on customer interactions to elicit and validate requirements and don't spend time on extensive specifications.

Cheaper (overall) to manage change gradually

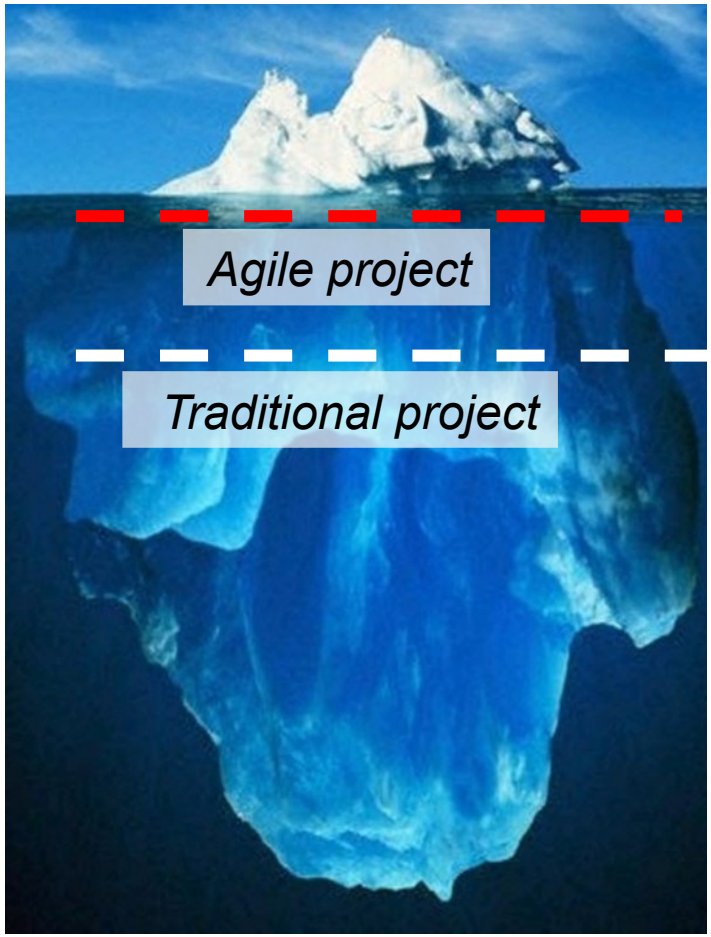
- The cost of making changes to big systems increases dramatically over time.
- **Therefore**, do RE by frequent iterations in small increments, and refactor continuously to reduce cost of change.

RE in Agile Projects [AGRE]

Practices

- *Iterative RE*: Gradual detailing
- Work order
  - *Extreme prioritization*: Just-in-time
  - *Constant planning*
- Integrated RE:
  Dev roles more involved in RE
  - *Face-to-face communication*
  - *Reviews & tests*
  - *Prototyping*
  - *Test-driven development*

Level of detail at dev start



[AGRE]

Agile Requirements Engineering Practices: An Empirical Study

Balasubramaniam Ramesh and Lan Cao

IEEE Software, pp. 60-67, January/February 2008

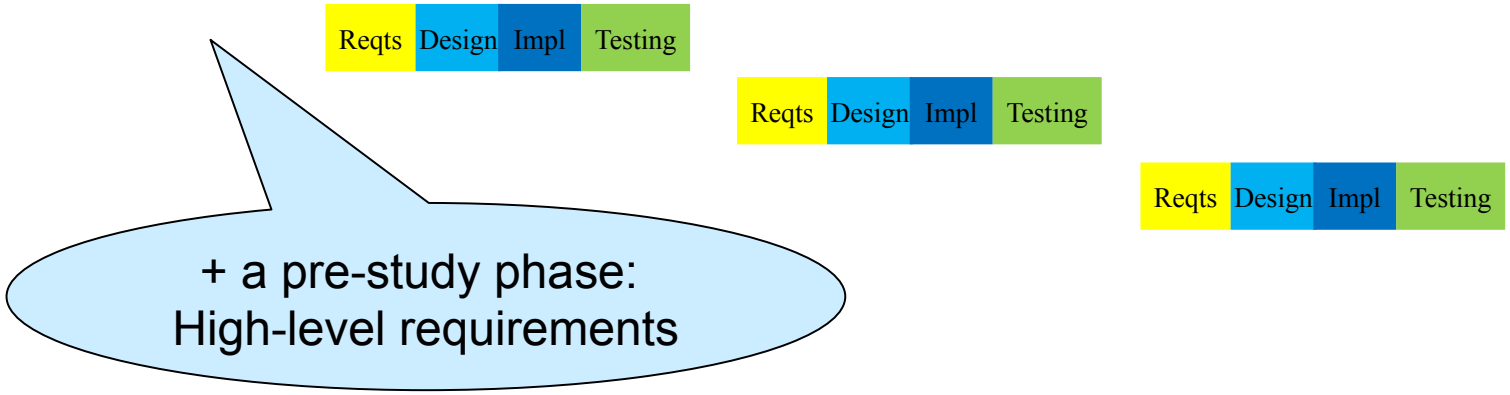
Agile RE practices in 16 companies

Adoption level	Practice						
	Face-to-face communication	Iterative RE	Extreme prioritization	Constant planning	Prototyping	Test-driven development	Reviews & tests
High	8	9	10	8	8	5	11
Medium	8	5	6	6	3	1	4
Low	0	2	0	2	0	0	1
None	0	0	0	0	5	10	0

Organization pseudonym	Industry and products
Enco	Energy and communications. Offers forecasting tools.
HealthCo	Healthcare and utilities. Offers an online service to help customers select health insurance and utility services.
Venture	Across industries. Helps brick-and-mortar companies develop a Web presence.
Entertain	Film and television industry. Offers high-tech indexing and search tools online.
HuCap	Administration. Carries out human-resource administration for other companies online.
TravelAssist	Transport and tourist industry. Offers online services.
ManageRisk	Across several industries. Offers insurance online.
Transport	Transportation and logistics industry. Offers services online.
ServeIT	Consulting and services. We studied the part of the firm that offers consulting services for business-to-business communication.
HealthInfo	Healthcare information systems. Offers information systems solutions to hospitals, physicians' offices, and home healthcare providers.
SecurityInfo	Security software. Offers software for Internet security.
AgileConsult	Software consulting. Offers consulting services on agile software development.
EbizCo	Packaged software development. Offers e-business connections and transactions.
FinCo	Online financial-transaction support. Offers online payments.
NetCo	Network software consulting. Offers services on developing network systems and architectures.
BankSoft	Banking information systems. Offers software that handles financial transactions.

Traditional Development Process

Agile Development Process – Integrated RE



- Same activities, different sizing and timing
  - → Different principles and management approach
  - → Different people detailing requirements
  - → Different documentation formats

User story & Acceptance Criteria

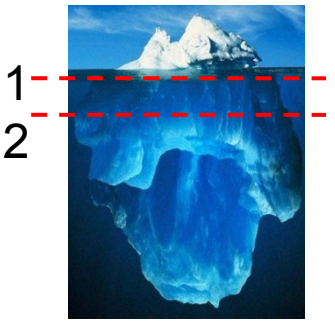
User story:

As a passenger, I can cancel a flight reservation

Acceptance criteria / test cases

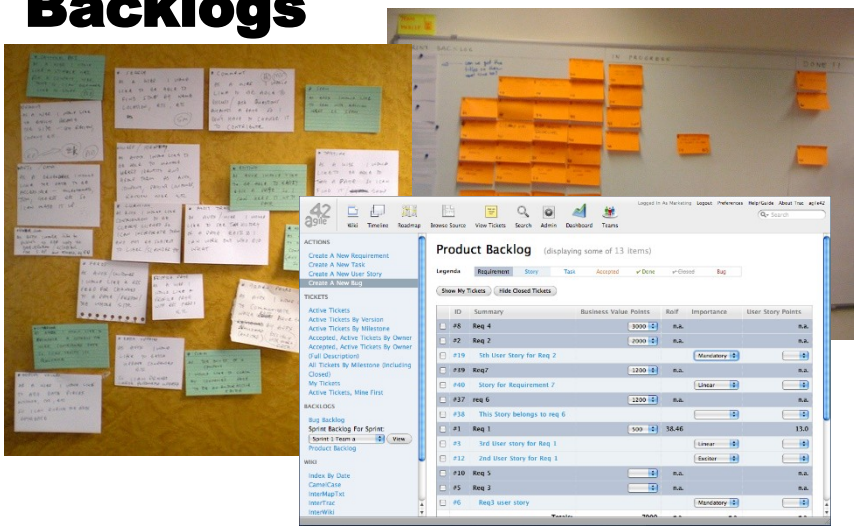
- Verify that a premium member can cancel the same day without a fee
- Verify that a non-premium member is charged 10% for a same-day cancellation
- Verify that an email confirmation is sent
- Verify that the hotel is notified of any cancellation

Specification with user stories

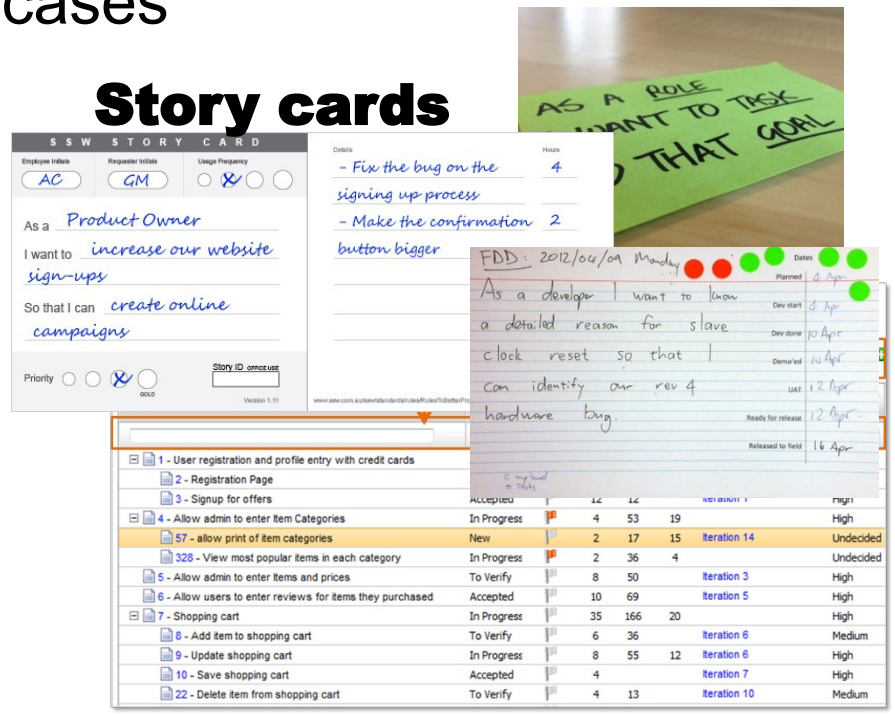


1. Product Owner/Customer **defines** & **prioritizes** Epics/User stories in **product backlog**
2. Team **defines** details for each user story in **sprint backlog**
    1. Tasks
    2. Acceptance criteria & test cases

Backlogs



Story cards



Face-to-face communication

Direct communication between customer and development

- Techniques
  - User Stories == high-level requirements spec
  - Complemented by other artifacts, e.g. "backlog"
- Prerequisites
  - Active involvement of (knowledgeable) customers

Customers can steer project

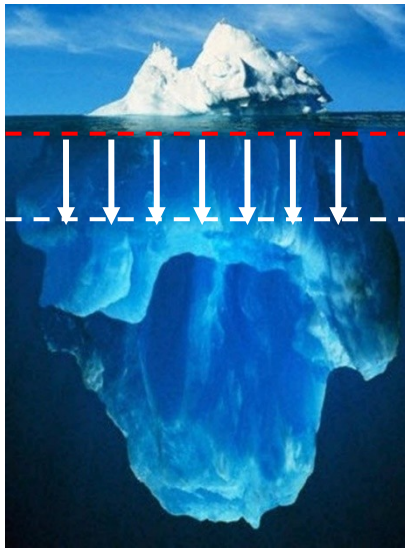
Avoids time-consuming documentation

Risk of inadequate requirements

On-site customer rep is challenging

Handling more than one customer

Relies on trust rather than agreed requirements



Iterative RE

Requirements **emerge during development** based on **initial high-level requirements**

- Techniques
  - Requirements analysis and detailing for each development cycle
  - Requirements intertwined with design

Good customer relationship

Clearer and understandable requirements due to direct customer interaction

Accurate cost and scheduling of project

Neglect of quality requirements

Lack of documentation beyond dev team

Extreme Prioritization & Constant Planning

Aim to deliver **most valuable features first**

Responsive to changes in customer demands

- Techniques
  - Work on most valuable features first
  - Continuously revise prioritisation & planning (for each iteration)
  - Constant feedback from customer

Customer provides business prio

Re-prioritization supported by dev process

Early validation minimizes need & cost for major changes

Other criteria suffer, e.g. quality

Instability in dev work

Inadequate architecture and increased costs

Refactoring requires time and experience

Prototyping & Reviews & Acc Test

Communicate through prototypes and frequent review meetings

Involves customers, developers and testers

Requirements **validation** and **refinement** through feedback

- Techniques

  End-of-sprint sign-off meeting

Efficient validation

Assess project status

Trust: Customer, Mgmt

Early problem identification

Risks with evolving prototypes in production

Unrealistic expections regarding leadtime

Weak formal validation, consistency checks

Dev of acc tests require access to customers

Test-Driven Development

Developers **create test before writing new code**

Tests specify expected behaviour of code

Tests capture complete requirements

Traces to production code facility reqts changes

Requires competence in testing, requirements understanding and customer collaboration

Most organizations fail to implement this practice

Summary of Benefits & Challenges of Agile RE

Practices	Benefits	Challenges
Face-to-face communication	• Customers can steer the project • No time-consuming documentation	• If no intensive interaction, then bad reqts. • On-site customer representation is difficult
Iterative RE	• Better relationship with the customer • More understandable reqts	• Cost & Schedule Estimation • Lack of documentation • Neglect of non-functional requirements
Extreme prioritization	• Customers provide business reasons • Opportunities for reprioritization.	• Business value not enough • May lead to instability
Constant planning	• Minimizes the need for major changes • Cost of addressing a change decreases	• Early architecture becomes inadequate • Refactoring isn't always obvious
Prototyping	• Help communicate with customers to validate and refine requirements	• Risky to deploy prototypes into production • Create unrealistic expectations
Test-driven development	• Gives traceability that make changes easier	• Developers unused to test before coding • Requires a thorough understanding of reqts and extensive collaboration between the developer and the customer
Reviews & acceptance tests	• Help to know if project is on target • Increase customer trust and confidence • Identify problems early • Obtain management support	• No formal model or verification of reqts • Consistency checking or formal inspections seldom occur. • Difficult if lacking customer access

Pros & Cons of Agile Development

Strengths

- quickly delivers working increments
- avoids unnecessary overhead
- short communication paths
- feedback from early stages used in developing latter stages

Weaknesses

- weak long-term and overall perspective
- weak / missing documentation
- weaker specialist competence
- less structure/guidance for weaker engineers

Product Management and Market-Driven Requirements Engineering (MDRE)

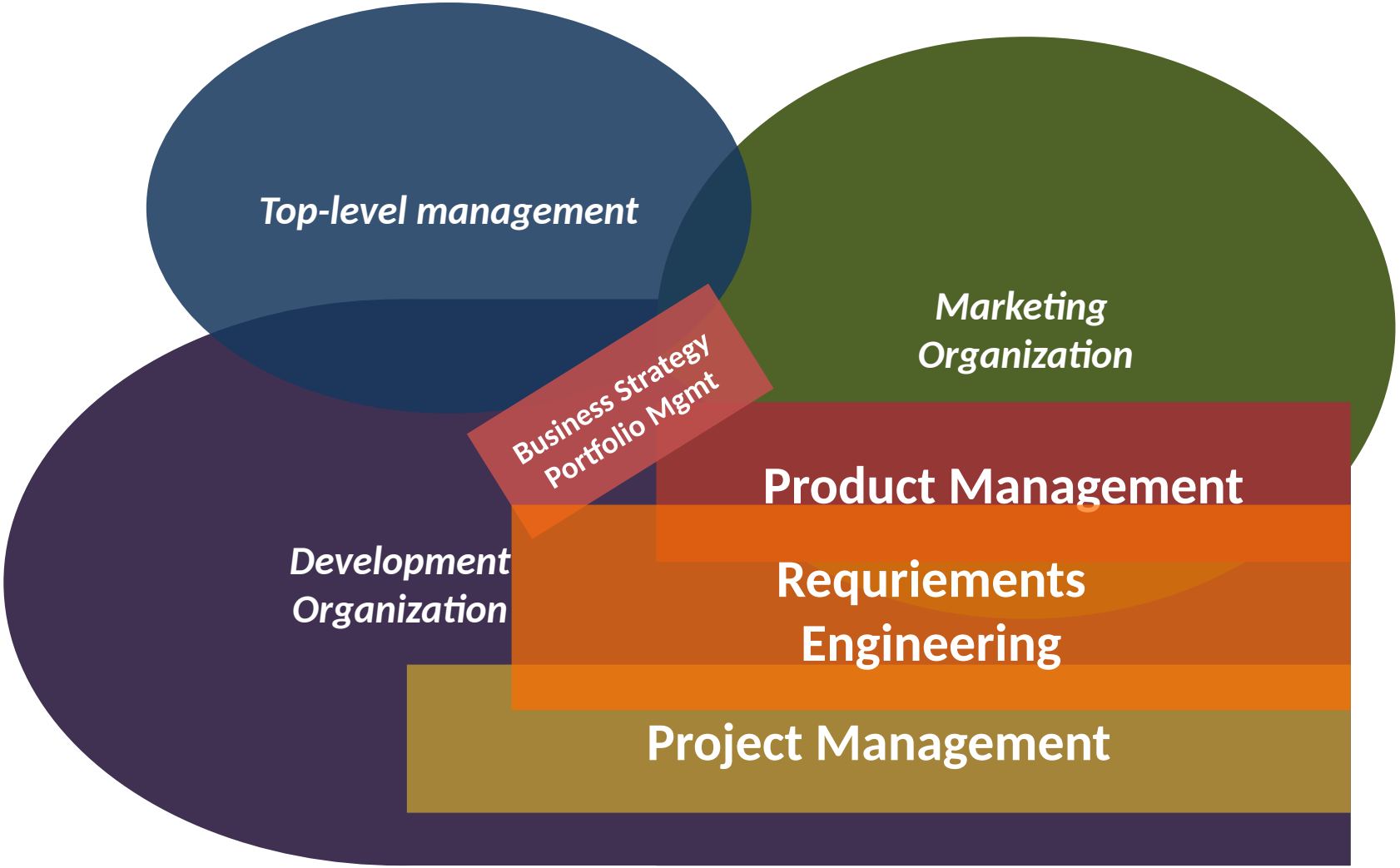
[MDRE]

Market-Driven Requirements Engineering for Software Products

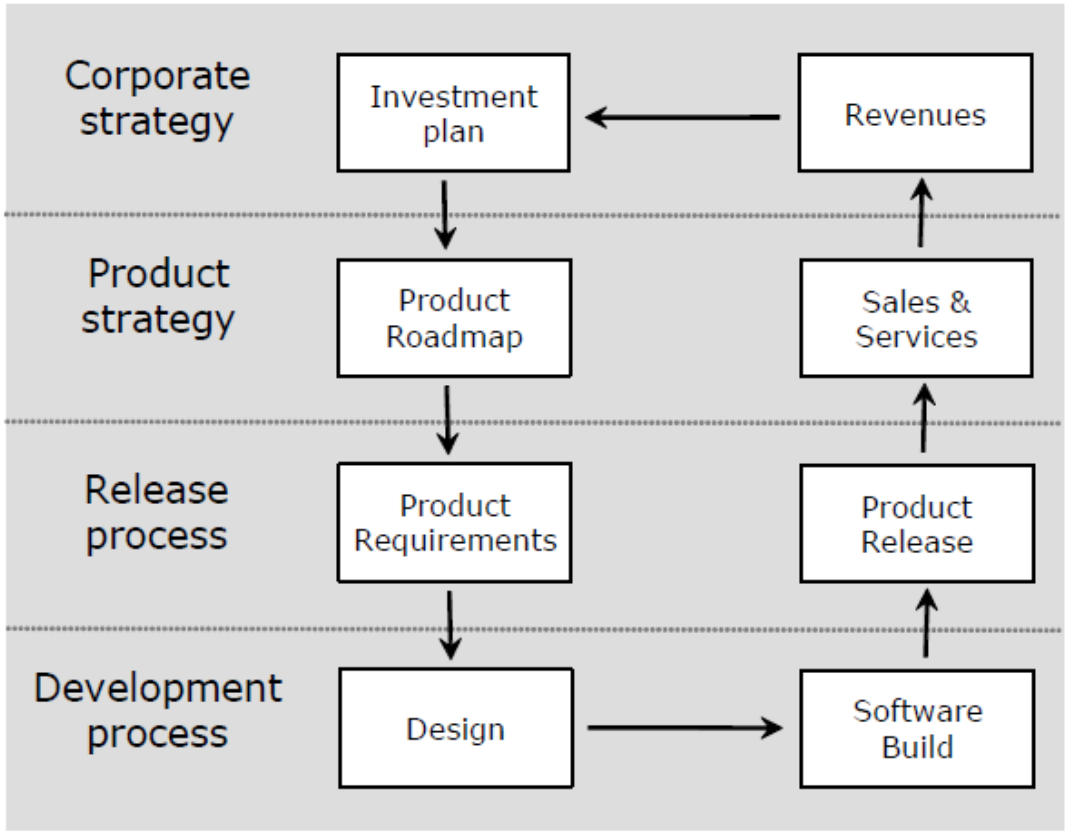
Regnell, B., & Brinkkemper, S.

Book chapter in *Engineering and Managing Software Requirements*, Eds. A. Aurum and C. Wohlin, Springer, ISBN 3-540-25043-3, 2005

RE vs. Product & Project Mgmt



The investment cycle



[MDRE]

Different types of products

1. Generic product on the open market
2. Customer-specific product developed based on contract

- The distinction is often blurred:
  the same organization combines several types
  - e.g., generic + customized
- Sometimes products evolve from customer specific to generic

Table 13.1. Examples of variants of hardware and software products.

	Pure Hardware	*Embedded Systems (HW+SW)*	*Pure Software*
Generic	Note sticks	Mobile phone	Firewall
Customized	Office furniture	Customized car	Enterprise resource planning systems
Customer-Specific	Portrait painting	Military vehicle	Web Site

[MDRE]

Characteristics of MDRE

- Success through sales and market share
  - (not just customer satisfaction)
- Release Planning focus on
  - Time-to-market
  - Multiple release
- Continuous evolution
  - (not just maintenance)
- Inventing requirements + market analysis
  - (not just collecting 1-on-1)
- Stakeholders
  - Market segments with potential customers
  - Competitors (confidentiality often needed)
- Continuous inflow of requirements

[MDRE]

Some challenges in MDRE

- Balancing market pull and technology push
- Chasm between marketing and development
- Requirements dependencies
- Cost-value-estimation and release planning
  - Over- and under-estimation
- Overloaded requirements management
  - Stage gates and triage

[MDRE]

Decisions outcomes in MDRE

		Decision	
		Selected	*Rejected*
Requirements Quality	*alfa*	*A* Correct selection ratio	*B* Incorrect selection ratio
	beta	*C* Incorrect selection ratio	*D* Correct selection ratio

Product Quality: $Q_p = A/(A+C)$

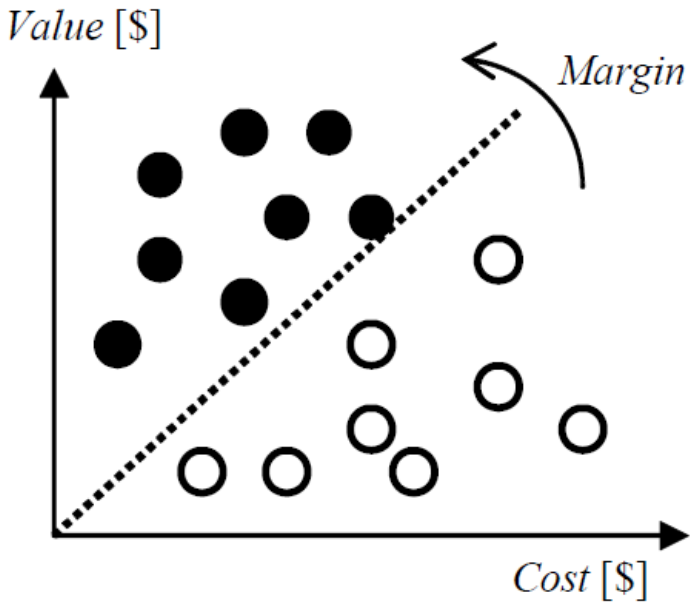
Decision Quality: $Q_d = (A+D)/(A+B+C+D)$

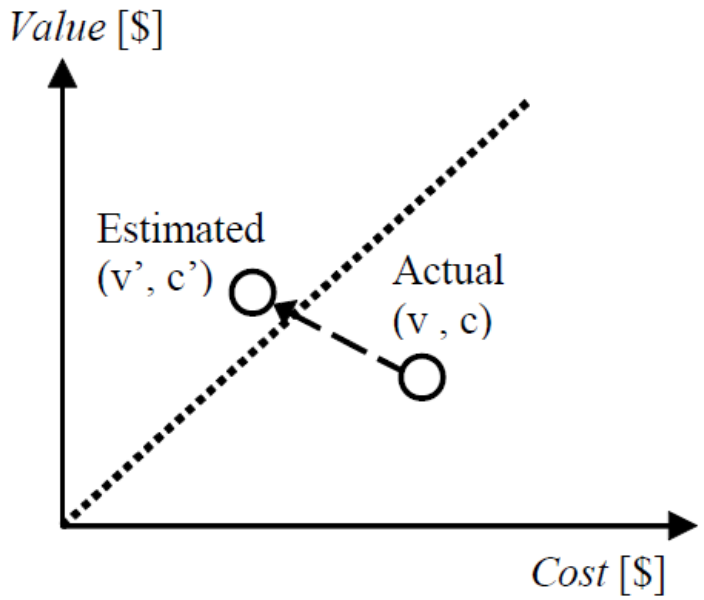
[MDRE]

Finding the golden grains despite uncertain cost-value estimates

Figure 13.1 (a) Cost-Value Diagram with alfa-requirements (filled) and beta-requirements (empty).

Figure 13.1 (b) Estimated values are differing from actual values causing wrong selection decision.





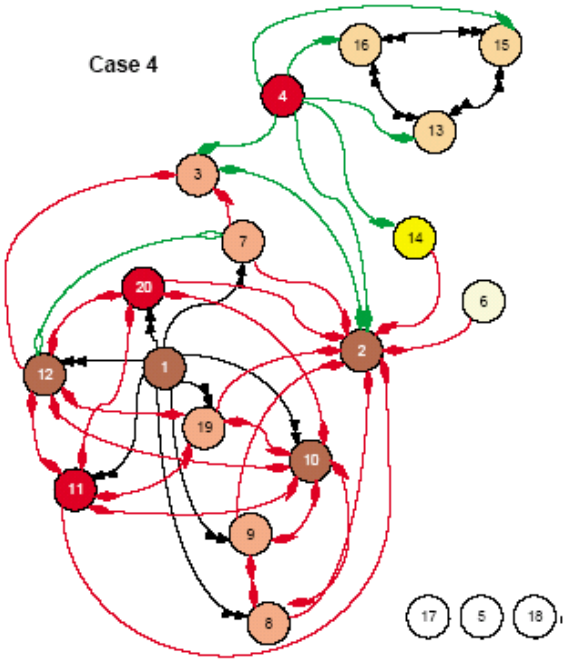
[MDRE]

Some inter-related challenges in MDRE

- Requirements **dependency** management
- Requirements **prioritization**
- **Release planning**

  - Balancing market pull and technology push
  - Chasm between marketing and development
  - Cost-value-estimation (over- & under-est.)
  - Overloaded requirements management

[INTDEP]



An industrial survey of requirements interdependencies in software product release planning

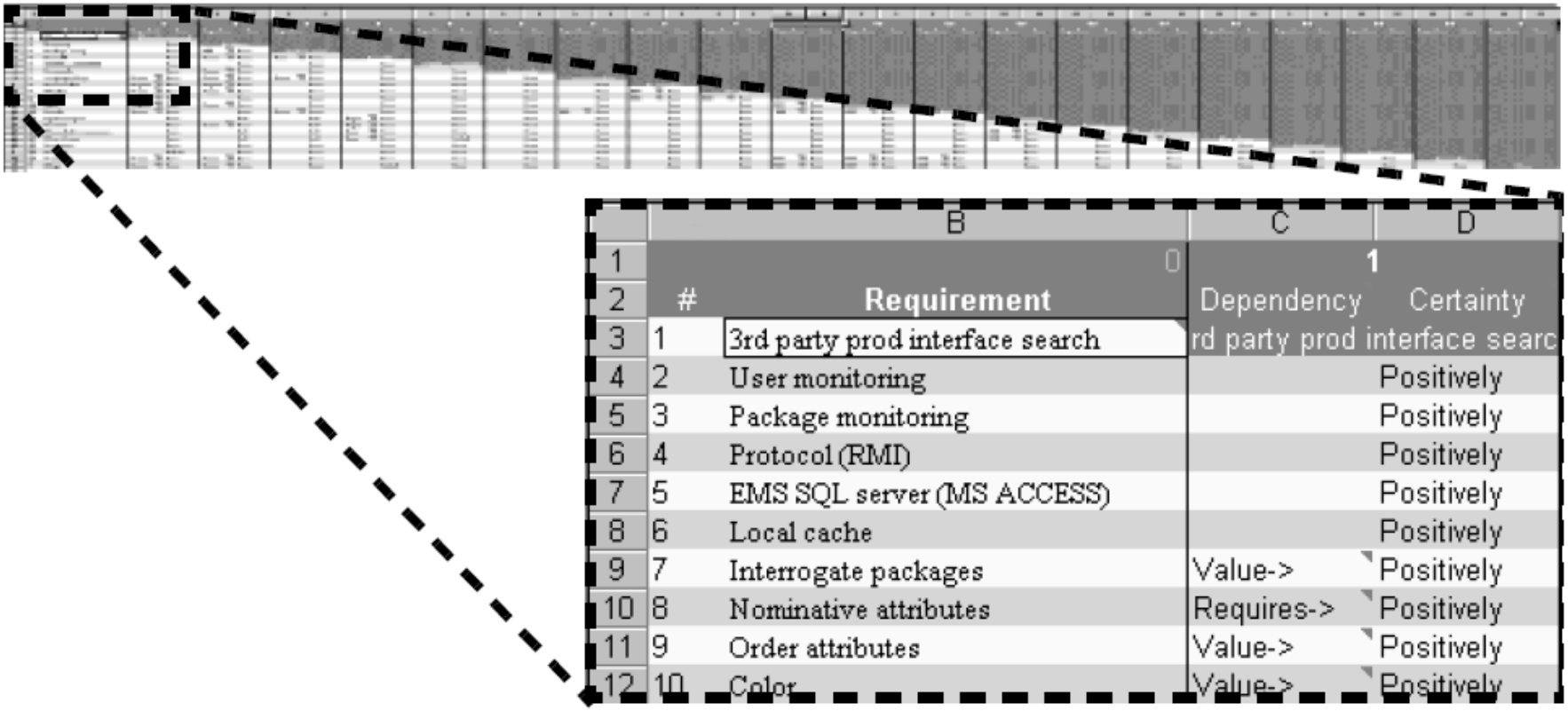
Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J.

IEEE Int. Conf. on Requirements Engineering (RE01), Toronto, Canada, pp. 84–91 (2001)

Research Method

- survey of five different companies
- a manager of a product/project was asked to identify and classify interdependencies among 20 high priority requirements.

Data collection



	B	C	D
1	0	1	
2	# Requirement	Dependency	Certainty
3	1 3rd party prod interface search	rd party prod interface searc	
4	2 User monitoring		Positively
5	3 Package monitoring		Positively
6	4 Protocol (RMI)		Positively
7	5 EMS SQL server (MS ACCESS)		Positively
8	6 Local cache		Positively
9	7 Interrogate packages	Value->	Positively
10	8 Nominative attributes	Requires->	Positively
11	9 Order attributes	Value->	Positively
12	10 Color	Value->	Positively

Figure 1. The spreadsheet designed for pairwise assessment of 20 requirements.

Different types of interdependencies

Table 2. Preliminary set of interdependencies.

Priority	Type	Meaning
1	R_1 AND R_2	R_1 requires R_2 to function, and R_2 requires R_1 to function.
2	R_1 REQUIRES R_2	R_1 requires R_2 to function, but not vice versa.
3	R_1 TEMPORAL R_2	Either R_1 has to be implemented before R_2 or vice versa.
4	R_1 CVALUE R_2	R_1 affects the value of R_2 for a customer. Value can be either positive or negative.
4	R_1 ICOST R_2	R_1 affects the cost of implementing R_2. Value can be either positive or negative.
5	R_1 OR R_2	Only one of $\{R_1, R_2\}$ needs to be implemented.

Examples:

AND. A printer requires a driver to function, and the driver requires a printer to function.

REQUIRES. Sending an e-mail requires a network connection, but not the opposite.

TEMPORAL. The function *Add object* should be implemented before *Delete object*. (This type is doubtful, which is discussed in section 3.1)

CVALUE. A detailed on-line manual may decrease the customer value of a printed manual.

ICOST. A requirement stating that “no response time should be longer than 1 second” will typically increase the cost of implementing many other requirements.

OR. In a word processor, the capability to create pictures in a document can either be provided as an integrated drawing module or by means of a link to an external drawing application.

Not always straight forward …

- "*if R2 is completely worthless to the customer without R1, and we would thus never do R2 without R1, do we classify the relationship as REQUIRED or just CVALUE?*"
- REQUIRES sometimes arises from the opposite reasoning: "If we do R2, then we can do R1 too!", which implies that the direction of the relationship could be the opposite;
  could e.g. be called "ENABLES" or "**HELPS**"

Summary of identified interdependencies [INTDEP]

Table 2. Summary of identified interdependencies.

	# dependencies	most common type	# singular req's	10% of the req's are responsible for	20% of the req's are responsible for	coupling (cf. section 3.5)
Case 1 (prod.)	19	ICOST 79%	4	47% of distinct interdep's	79% of distinct interdep's	10%
Case 2 (prod.)	29	CVALUE 45%	3	55% of distinct interdep's	76% of distinct interdep's	15%
Case 3 (prod.)	42	ICOST 86%	3	50% of distinct interdep's	74% of distinct interdep's	22%
Case 4 (besp.)	41	AND 41%	3	44% of distinct interdep's	71% of distinct interdep's	22%
Case 5 (besp.)	24	REQUIRES 79%	4	42% of distinct interdep's	67% of distinct interdep's	13%

1. 10% of the requirements are responsible for roughly 50% of the interdependencies
2. 20% of the requirements are responsible for roughly 75% of all interdependencies
3. About 20% of the requirements are singular
4. Customer-specific: more functionality-related ; Market-driven: more value-related dependencies

Example of dependency structures

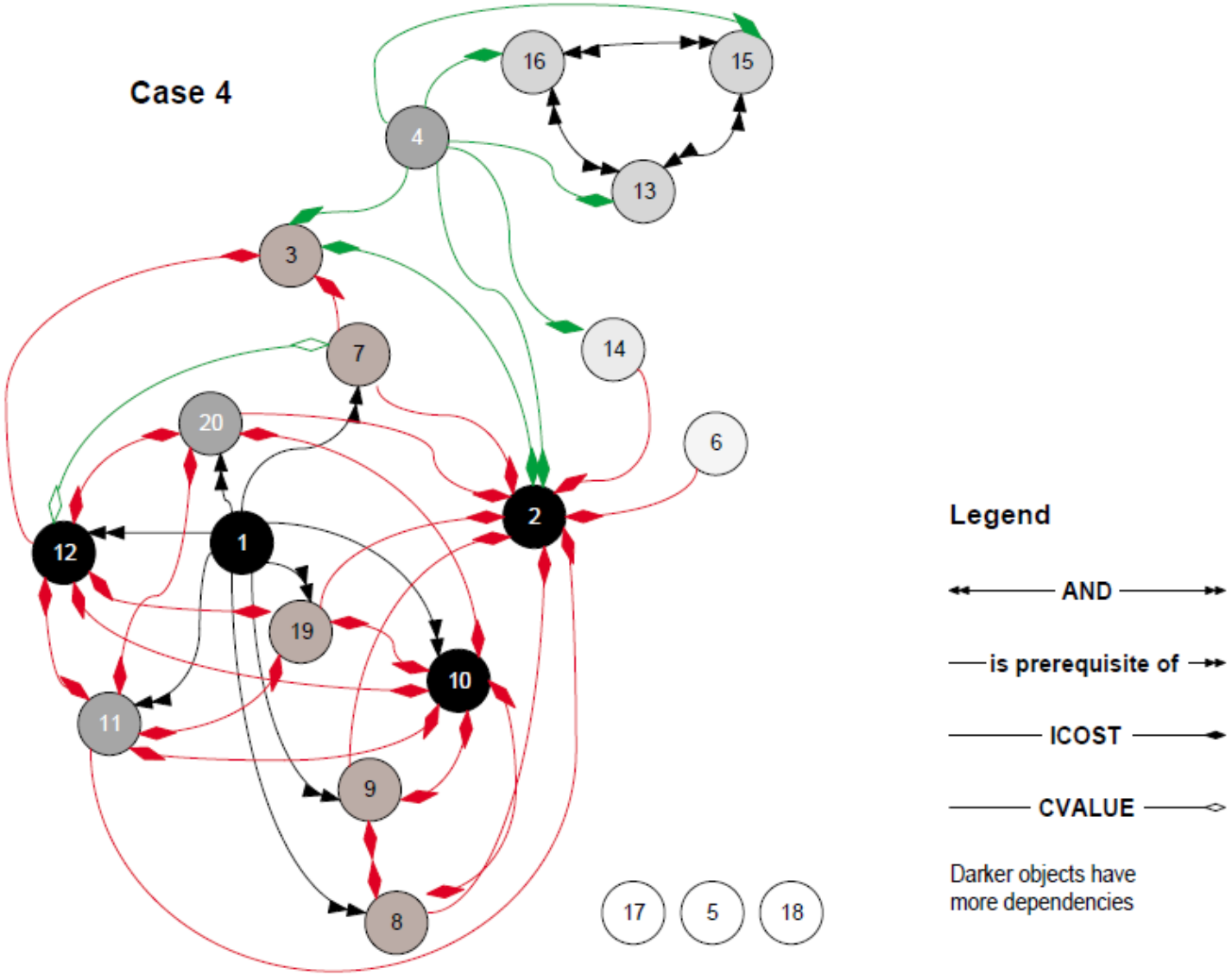


Figure 2. Visualization of requirements interdependecies for one of the five cases.

Coupling measures

$$Creq = \frac{I}{(R(R-1))/2}$$

I =#dependencies
R =#requriements

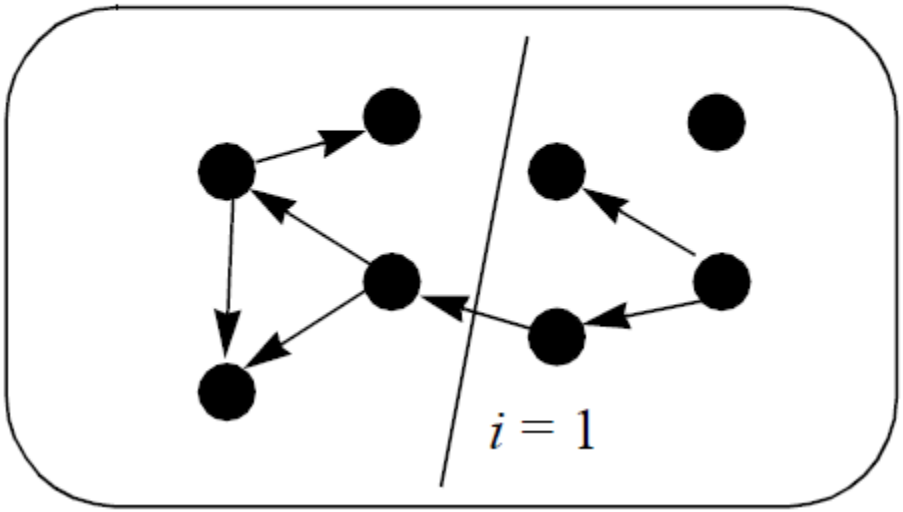
In survey:
10-22%

Release coupling:

$$Crel = \frac{i}{I}$$

i = #dep. betw. 2 partitions

Figure 3. Example illustrating the concepts of require-ments and release coupling.



$R = 8$
$I = 7$

$$Creq = \frac{7}{28}$$

$$Crel = \frac{1}{7}$$

Expressing dependencies in reqT

- An **AND** relation is equivalent to two **mutual requires**-relations:

```
Feature("printerX1") requires Feature("driverX")
Feature("driverX") requires Feature("printerX1")
```

- A requires relation can be **non-mutual** :

```
Feature("sendEmail") requires Feature("networkAccess")
```

- **Temporal relations** regarding a preferred implementation order can be expressed using **precedes**:

```
Function("add") precedes Function("delete")
```

- Exclusion (xor) can be expressed by an **excludes** relation (only one is needed as exclusion is mutual):

```
Design("centralized") excludes Design("distributed")
Design("distributed") excludes Design("centralized")
```

- Entities that support or hinder each other can be modeled using **hurts** and **helps** relations :

```
Goal("secure") helps Goal("safe")
Goal("secure") hurts Goal("simple")
```

[Some examples modified from Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag: "An industrial survey of requirements interdependencies in software product release planning", J.: *Int. Conf. on Requirements Engineering (RE01)*, Toronto, Canada, pp. 84–91, 2001]

Requirements Prioritization
(summary from week 1)

Prioritization techniques [PRIO]

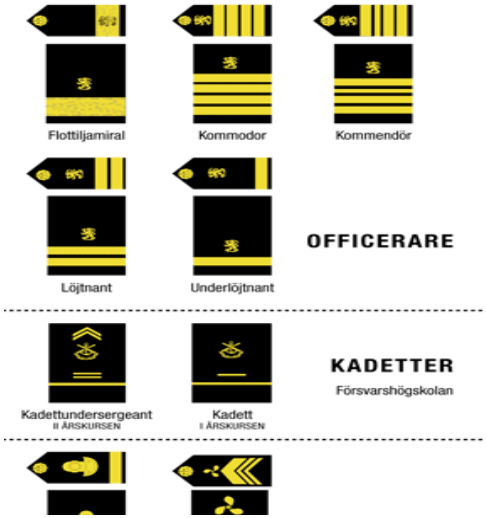
- **Direct numerical assignment (grading)** [Lau 7.4]
  - Can be done using any scale (categorical, ordinal, ratio) depending on what the number actually means.
  - Quick & easy; **but**
    - a risk is that all reqs are deemed highly important as they are not challenged against each other
    - may be misinterpreted as ratio scale (even if ”4” not necessarily is "twice as much" as ”2” when using an ordinal scale).
- **Ratio scale 100$-test**
  - Ratio scale, quick and easy, risk of shrewd tactics (listigt taktikspel)
- **Ordinal scale Ranking**
  - sorting (easy, quick) or pairwise comparison (show consistency)
- **Top-ten (or Top-n)**
  - *Ordinal scale* if the top list is ranked or *Categorical scale* if grouping is not ranked;
  - very quick and simple, gives a rough estimate on a limited set of req

Prioritization scales

Categorization

e.g.: must, ambiguous, volatile

Partition in groups without greater-less relations



Ordinal scale

e.g.: more expensive, higher risk, higher value

Ranked list
A>B

Ratio scale

ex: $, h,

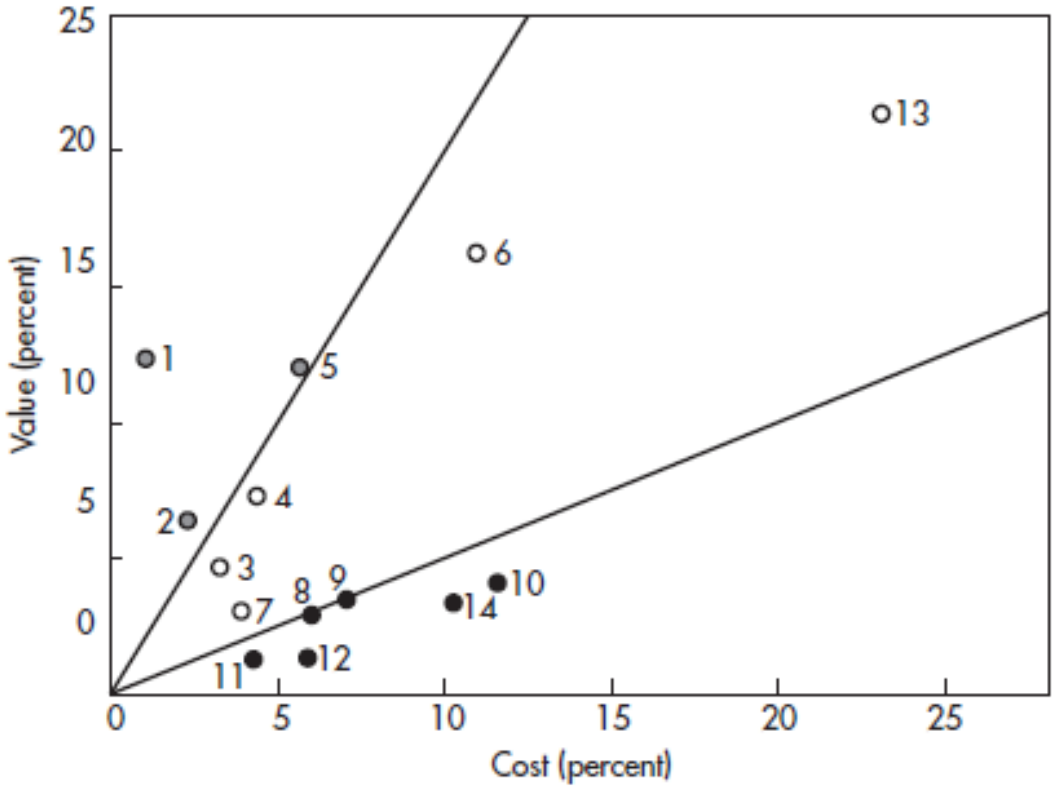
% (relative)

Numeric relations:
A=2*B

[PRIO]

Combine and visualize two criteria

Example: Cost-Benefit diagram



Karlsson, Joachim, and Kevin Ryan. "A cost-value approach for prioritizing requirements." *IEEE software* 14.5 (1997): 67-74.

Release Planning

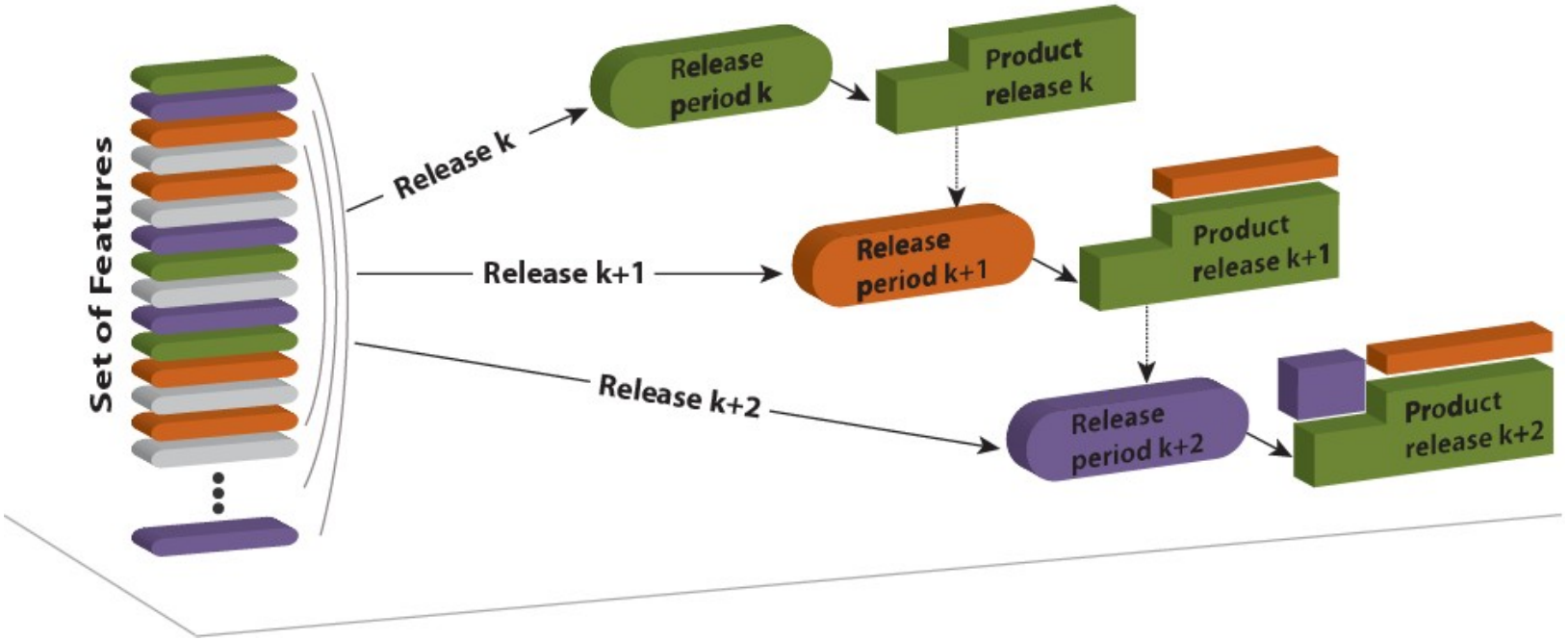
[RP]

The art and science of software release planning

Ruhe, G., & Saliu, M. O.

IEEE software, 22(6), 47-53. 2005

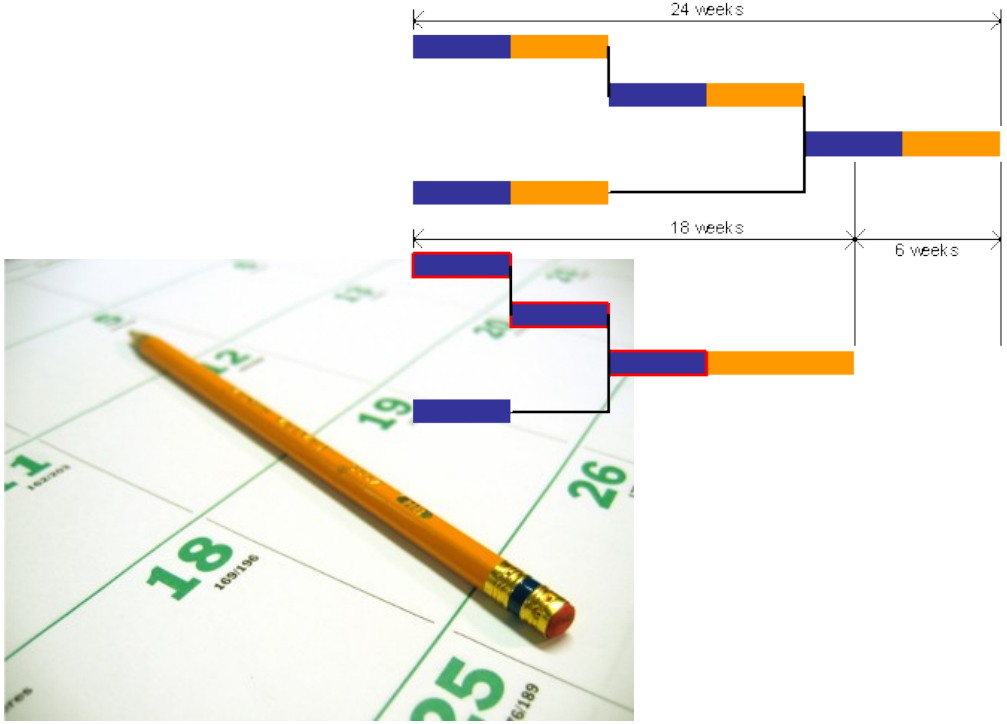
What is Release Planning?



[RP]

Release Planning involves...

- ...prioritization + scheduling under various constraints, e.g., resource and precedence constraints



[RP]

Example planning parameters

- Requirements priorities (from prioritization)
- Available resources
- Delivery time
- Requirements dependencies
  - Precedence, Coupling, Excludes
- System architecture
- Dependencies to the code base

[RP]

What is a good release plan?

- A good release plan should
  - Provide maximum business value by
    - offering the best possible blend of features
    - in the right sequence of releases
  - satisfy the most important stakeholders involved
  - be feasible with available resources, and
  - take dependencies among features into account

[RP]

Simplistic Release Planning

- Informal process
- Unclear rationale behind decisions
- No systematic management of dependencies
- Simplistic greedy allocation is no good
- A zillion possibilities already with 20 features and 3 releases:

$$4^{20} > 1.000.000.000.000 = 10^{12} \text{ possibilities}$$

[RP]


1.000.000.000.000
RELEASE PLANS

Why greedy allocation may be really bad...

```
val input = Model(
  Feature("a") has (Benefit(90), Cost(100)),
  Feature("b") has (Benefit(85), Cost(90)),
  Feature("c") has (Benefit(80), Cost(25)),
  Feature("d") has (Benefit(75), Cost(23)),
  Feature("e") has (Benefit(70), Cost(22)),
  Feature("f") has (Benefit(65), Cost(20)),
  Feature("g") has (Benefit(60), Cost(10)),
  Feature("h") has (Benefit(55), Cost(30)),
  Feature("i") has (Benefit(50), Cost(30)),
  Feature("j") has (Benefit(45), Cost(30)),
  Release("r1") has Capacity(100),
  Release("r2") has Capacity(90),
)
```

Run code in below gist to compare random and greedy release planning:
https://gist.github.com/bjornregnell/780b86285d8aff9830b7749bf7688ae1

Example from [RP]

WAS: weighted average satisfaction of stakeholder priorities

Table 2

Two qualified release plan alternatives, listing the release to which each feature is assigned and each weighted average satisfaction

	Release Plan x1		Release Plan x2	
Feature $f(i)$	$x1(i)$	WAS(i,k)	$x2(i)$	WAS(i,k)
1. Cost reduction of transceiver	1	84.0	1	84.0
2. Expand memory on BTS controller	1	287.0	1	287.0
3. FCC out-of-band emissions	1	252.0	3	0.0
4. Software quality initiative	3	0.0	1	233.8
5. USEast, feature 1	1	134.4	3	0.0
6. USEast, feature 2	2	516.6	3	0.0
7. China feature 1	2	277.2	1	88.2
8. China feature 2	2	43.2	1	19.6
9. 12-carrier BTS for China	3	0.0	2	72.0
10. Pole-mount packaging	3	0.0	3	0.0
11. Next-generation BTS	3	0.0	3	0.0
12. India BTS variant	3	0.0	2	75.6
13. Common feature 01	1	37.8	1	516.6
14. Common feature 02	1	8.4	1	277.2
15. Common feature 03	2	54.0	2	54.0
Objective function value $F(x)$		1,694.6		1,708.0

The release planning part of Lab 2

- Paper [RP] use mathematical optimization based on integer *linear programming*
- During Lab 2 you will use reqT to instead do release planning based on integer **constraint satisfaction**
- reqT includes a DSL for constraint satisfaction problems that can be solved using the JaCoP solver
- Before lab2: create a small RP problem for your project:
  - 3 features and 2 stakeholders
  - estimate relative benefit for each feature from the viewpoint of each stakeholder
  - estimates of relative cost for each feature from development and test perspectives
  - use fictitious estimates if necessary but aim to be realistic if possible
- During lab 2 task 2 you will use reqT to solve your RP problem.

To do …

- **Read papers: [PROTO], [AGRE], [MDRE], [INTDEP], [RP] [OSSRE]**
  - Many pages to read: **make a plan** to read some pages every day...
  - **Focus** your reading based on lecture slides

- **Guest lecture L6b on open source: Dr. Johan Linåker** (Tue 4pm in E:C)
- **Attend Exercise 3** on Functional requirements [Lau:3-5]
- **Hand in Release R1 in Canvas**
- **Book meeting** with supervisor in Canvas: reply to announcement by Matthias
- **Start preparing for Lab2** (week 5)
  - Quality requirements (QR) & Release Planning (RP)
  - Preparations include reading + working**,**
    prep. will take significantly **more time** compared to lab1

- *Next week...*
  - Attend **Lecture L7** on Quality Requirements [QUPER, Lau:6-7]:
    - Watch video on Quper before or after the lecture:
      https://cs.lth.se/krav/quality-requirements/
    - Attend **Exercise 4** where you work on QR in your project
  - ***Extra Seminar*** Feb. 11th at 3pm in E:C (not part of exam):
    AI4RE and RE4AI, Guest: Matthias Wagner, prepared questions welcome!