ETSN15 (2024) Requirements Engineering

Lecture 6 part a:

Market-Driven Requirements Engineering [MDRE]
Requirements inter-dependencies [INTDEP]
Release Planning [RP]
Preparations for Lab 2

Part 6b in separate pdf:
RE for Open Source Software [OSSRE]

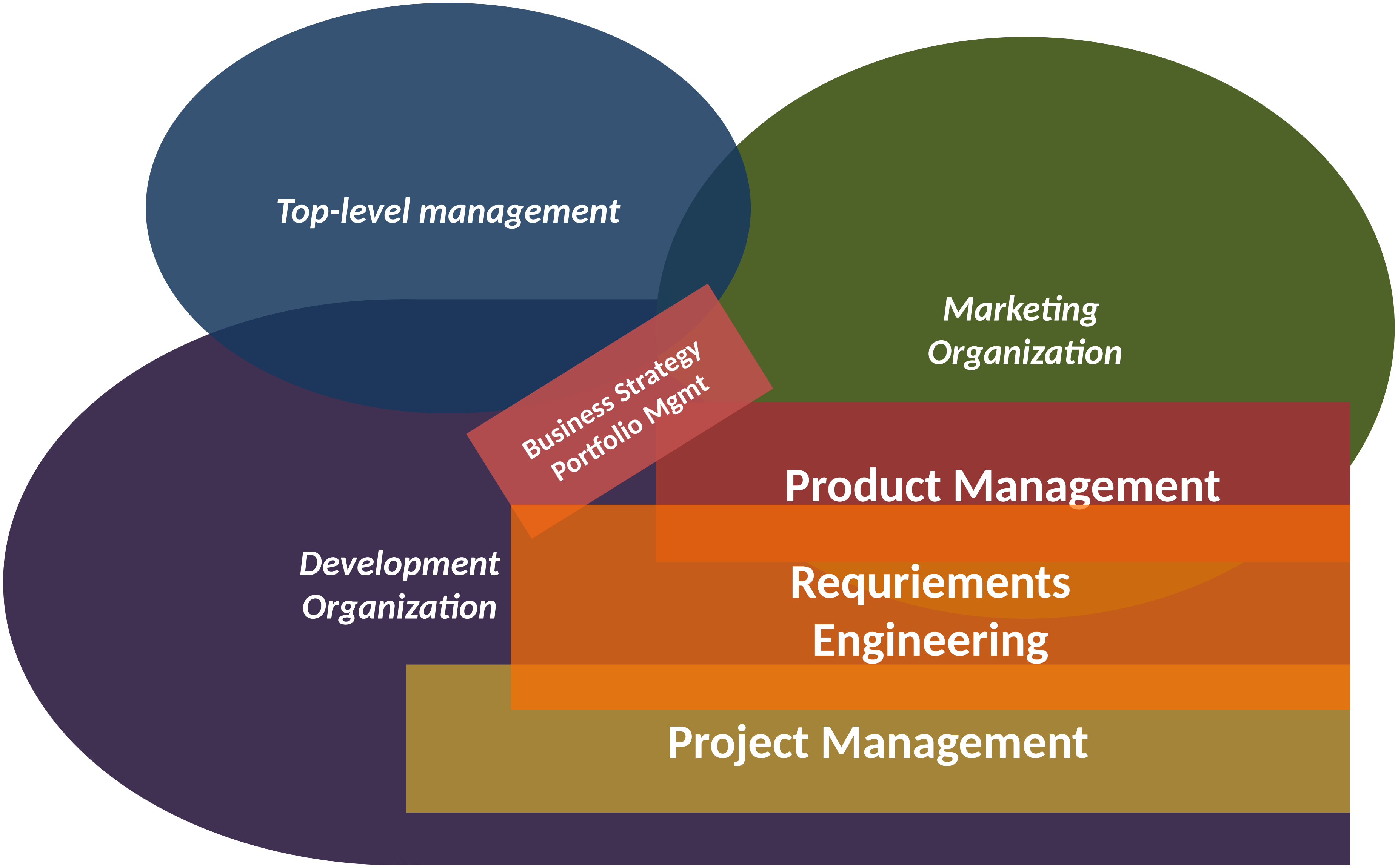
Björn Regnell
http://www.cs.lth.se/krav

Product Management and Market-Driven Requirements Engineering (MDRE)

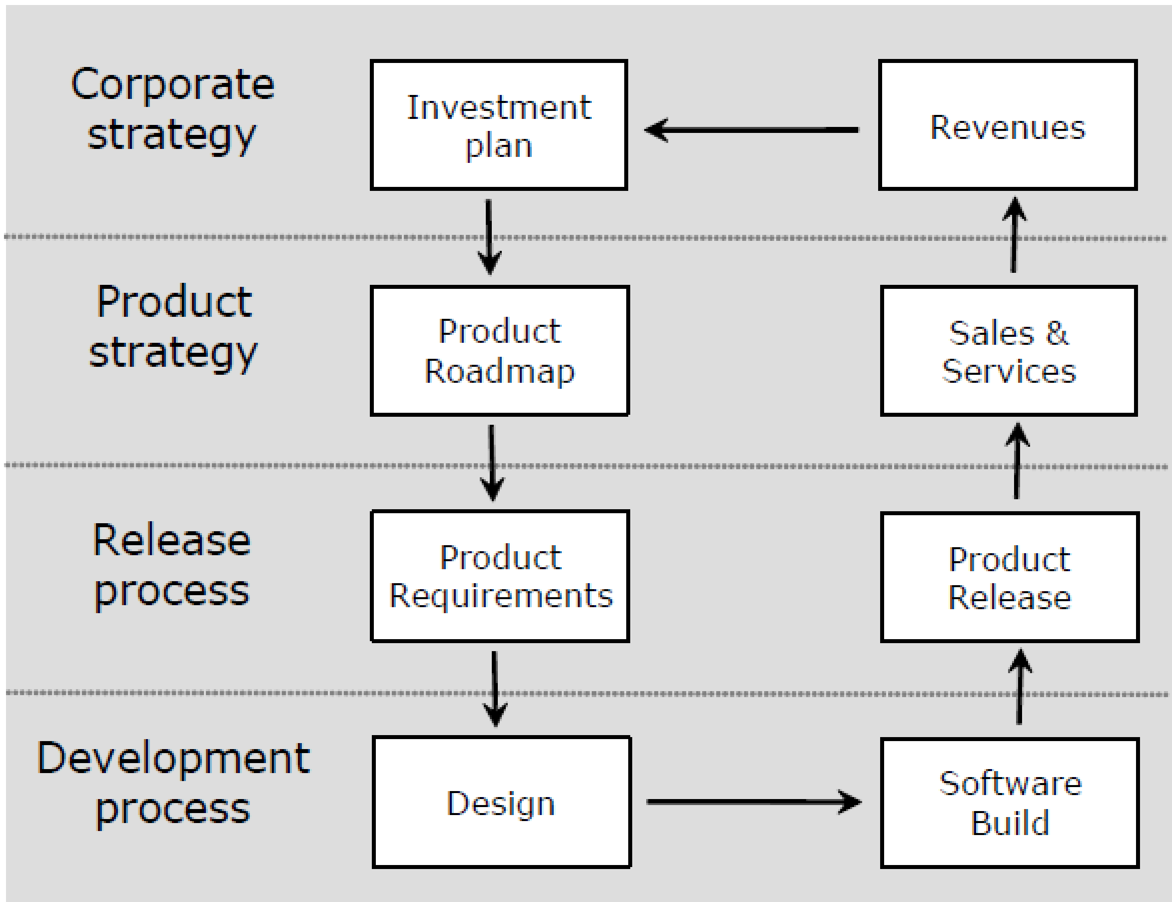
Book chapter [MDRE] in compendium

- Market-Driven Requirements Engineering for Software Products
- Regnell, B., & Brinkkemper, S.
- 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: $Q_p = A/(A+C)$

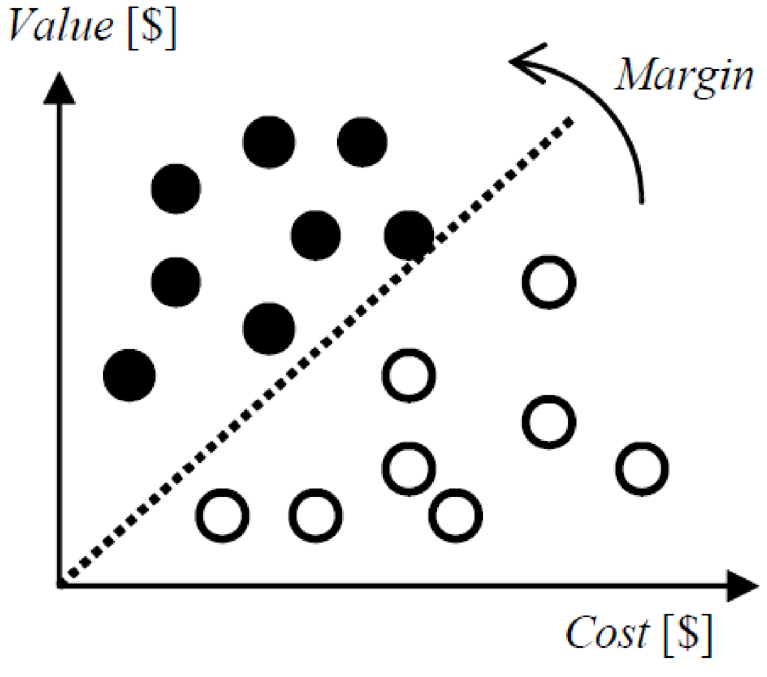
Decision Quality: $Q_d = (A+D)/(A+B+C+D)$

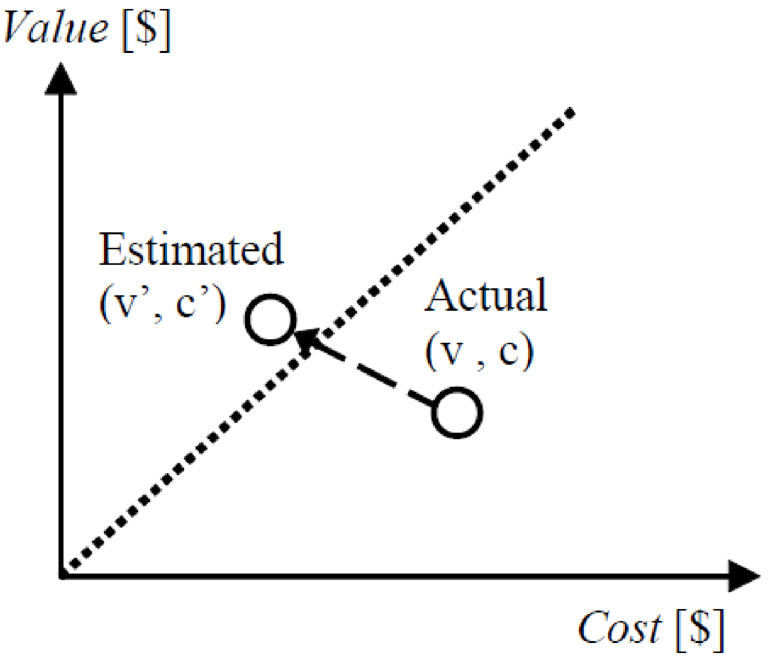
[MDRE]

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.





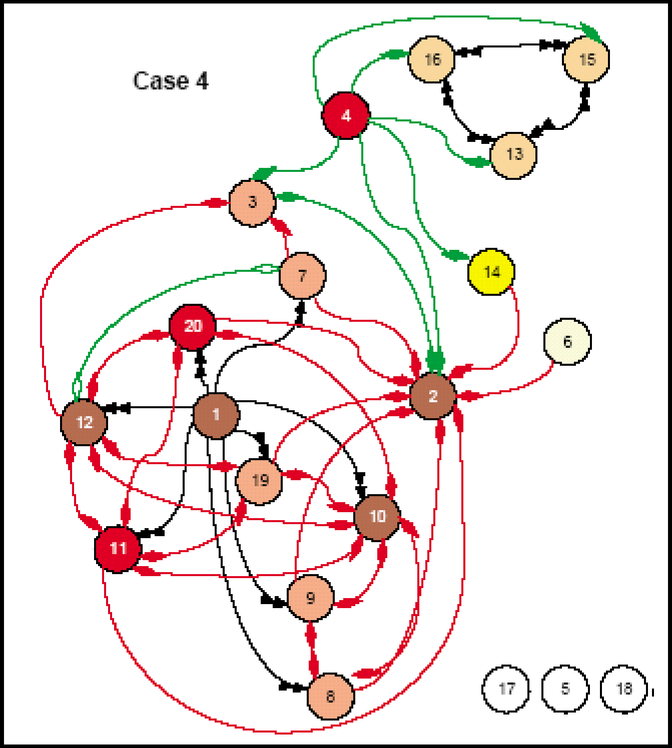
[MDRE]

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

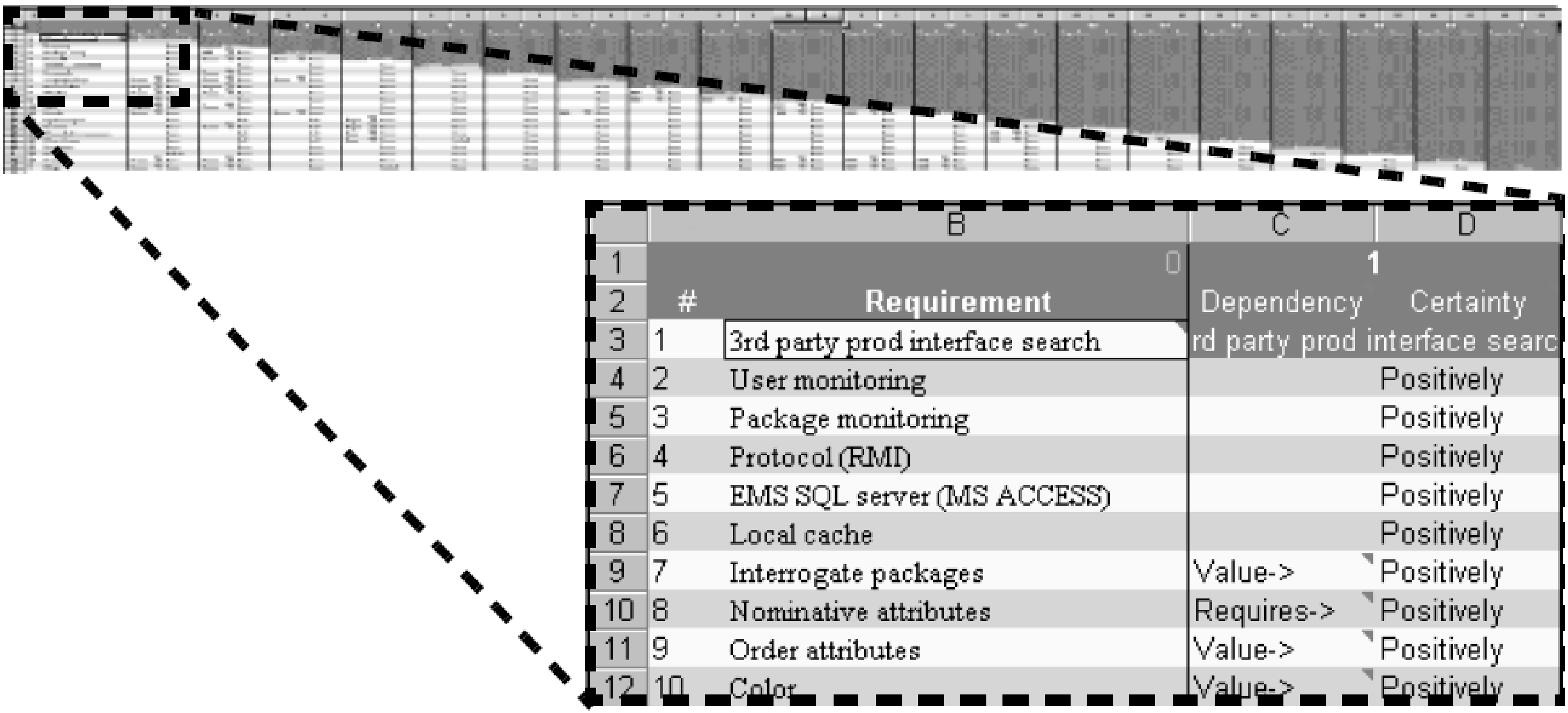
Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J.

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

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

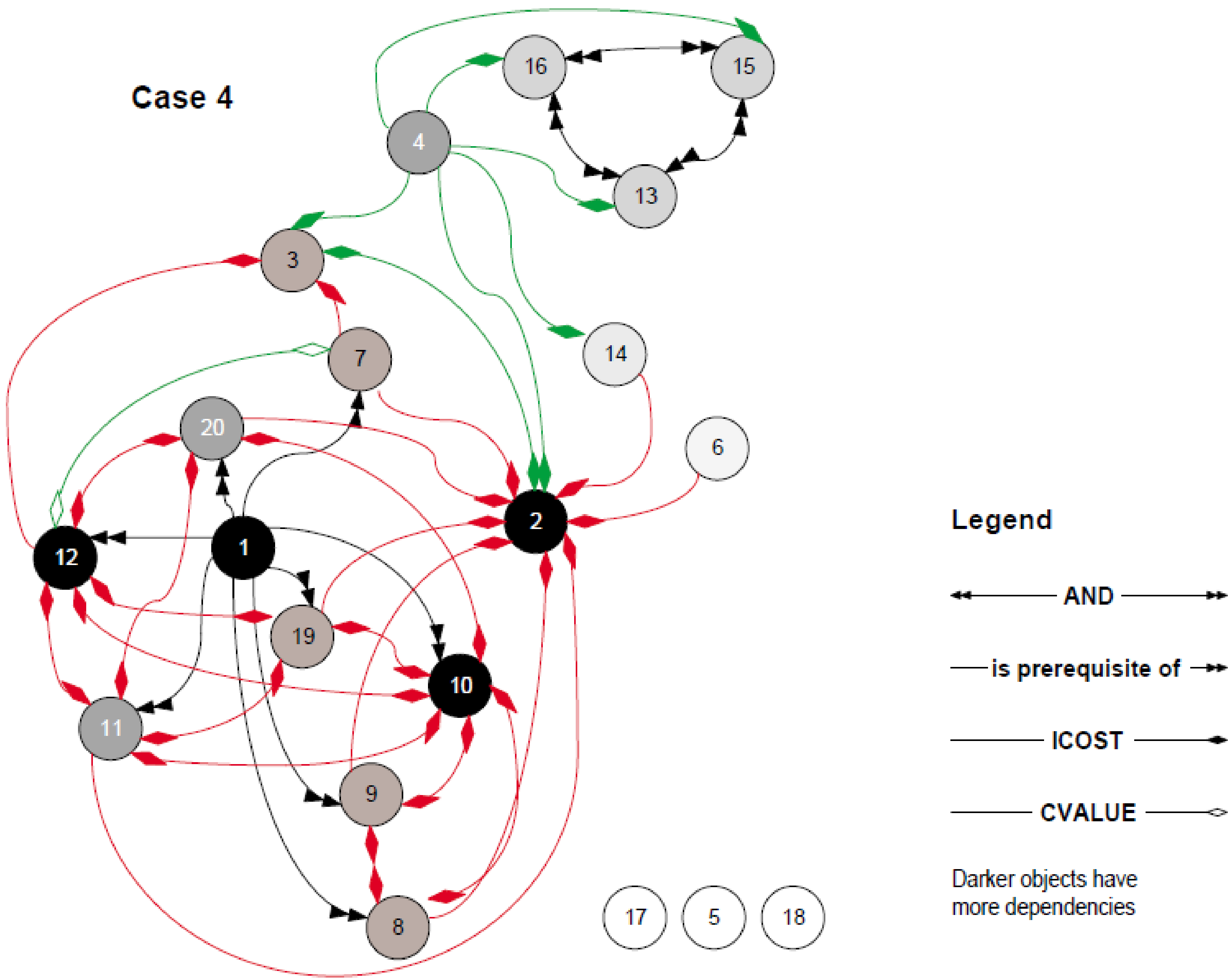


Figure 2. Visualization of requirements interdependecies for one of the five cases.

Coupling measures

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

I =#dependencies
R =#requriements

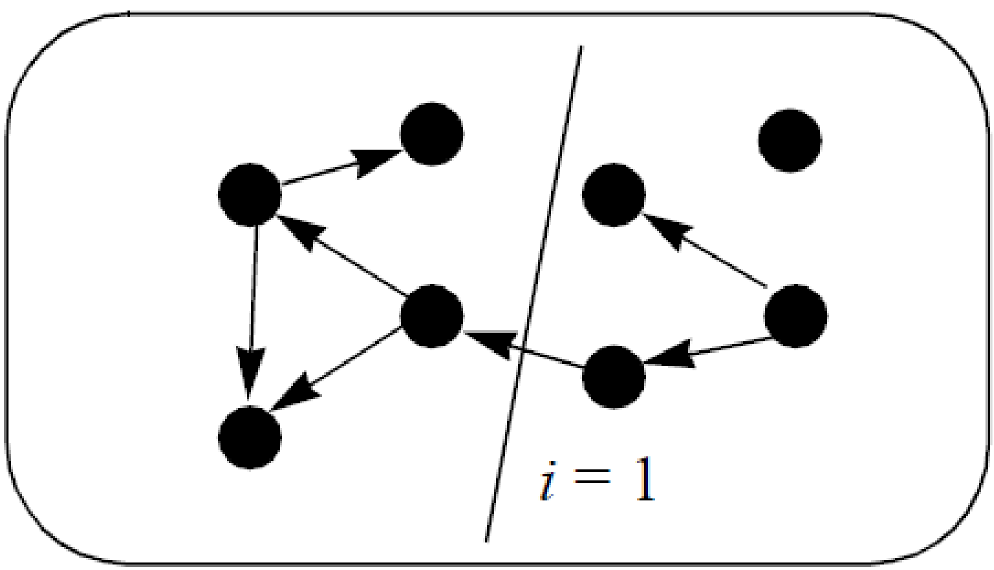
In survey:
10-22%

Release coupling:

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

i = #dep. betw. 2 partitions

Figure 3. Example illustrating the concepts of requirements 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]

Expressing CVALUE dependencies as Constraints in reqT

```
val m = Model(
          Req("x") has (Order(1), Benefit(100)),
          Req("y") has Order(1))  // Same release
val c = Constraints(
     Req("y")/Benefit :: {0 to 1000},
     Sum(Req("x")/Benefit, Req("y")/Benefit) === Var("SumXY"),
     Var("SumXY") :: {0 to 2000},
     IfThenElse(
       Req("x")/Order === Req("y")/Order, //If same release
       Var("SumXY") === 400,              //then more valuable
       Var("SumXY") === 200               //else less valuable
     ))

val m2 = (m + c).satisfy
```

```
m2: reqT.Model =
Model(
  Req("y") has (Benefit(300), Order(1)),
  Req("x") has (Order(1), Benefit(100)),
  Constraints(
    Var("SumXY") === 400))
```

Expressing CVALUE dependencies as Constraints in reqT

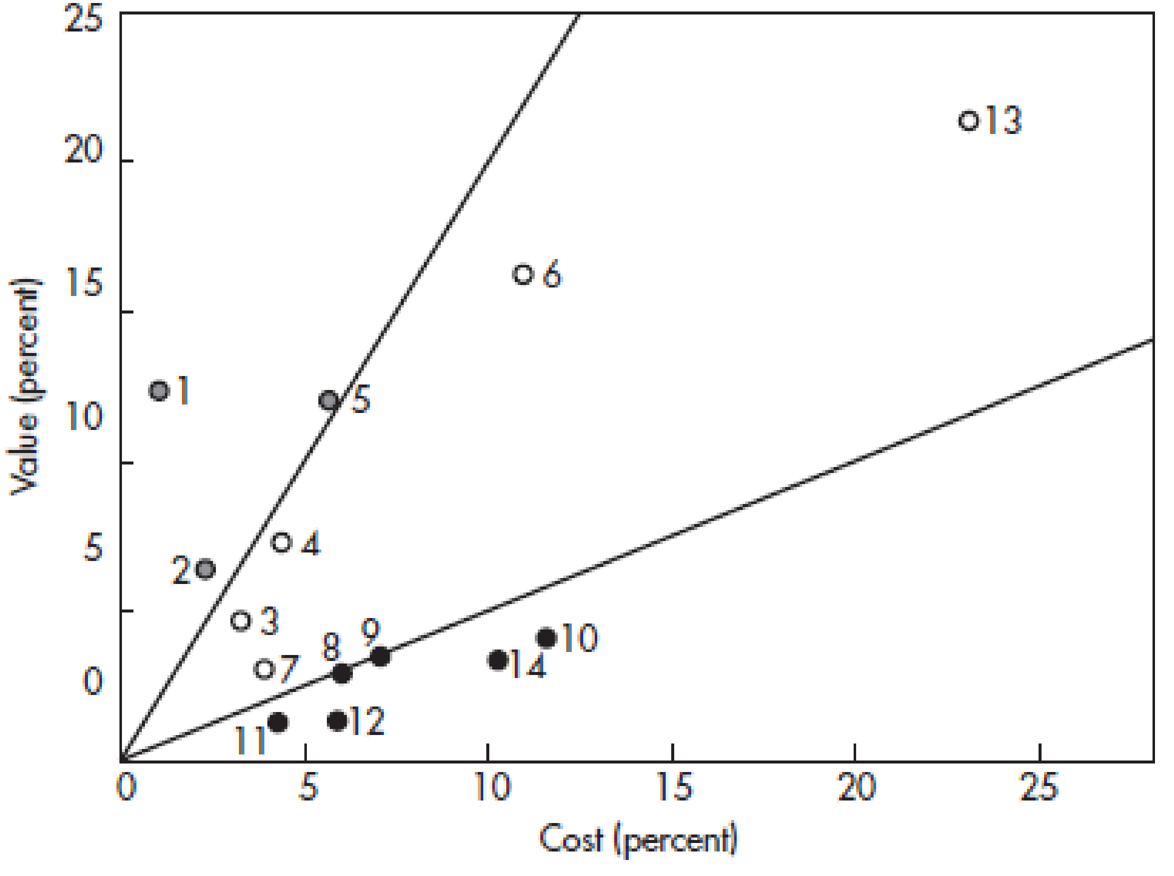
```
val m = Model(
          Req("x") has (Order(1), Benefit(100)),
          Req("y") has Order(2))  // Different releases
val c = Constraints(
     Req("y")/Benefit :: {0 to 1000},
     Sum(Req("x")/Benefit, Req("y")/Benefit) === Var("SumXY"),
     Var("SumXY") :: {0 to 2000},
     IfThenElse(
       Req("x")/Order === Req("y")/Order, //If same release
       Var("SumXY") === 400,              //then more valuable
       Var("SumXY") === 200               //else less valuable
     ))

val m2 = (m + c).satisfy
```

```
m2: reqT.Model =
Model(
  Req("y") has (Benefit(100), Order(2)),
  Req("x") has (Order(1), Benefit(100)),
  Constraints(
    Var("SumXY") === 200))
```

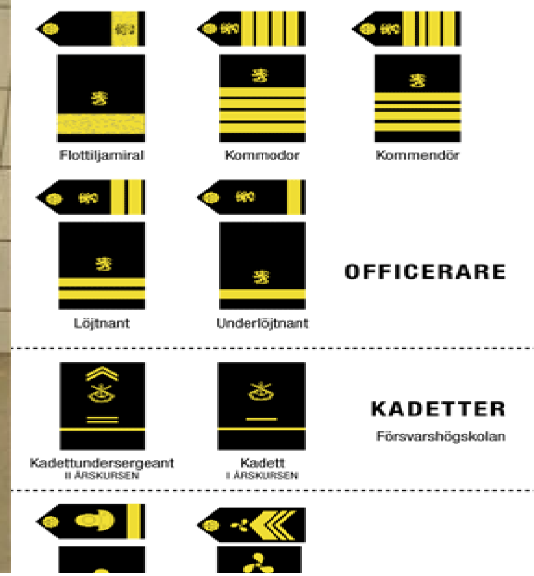
Requirements Prioritization (summary from week 1)

Estimating cost-benefit



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

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]

Prioritization techniques

- Grouping, numbering assignment (grading)
- **Ranking (sorting)**
- Top-ten (or Top-n)
- Analytical Hierarchy Process (AHP)
- **100$ test**
- Combination of techniques

On Lab 1 you used:

- **ordinal**-scale prio with **Ranking (sorting)** by pair-wise comparisons and
- **ratio**-scale prio with the **100$ test**

One (simplistic) approach to manage interdependencies:

- grouping

[PRIO]

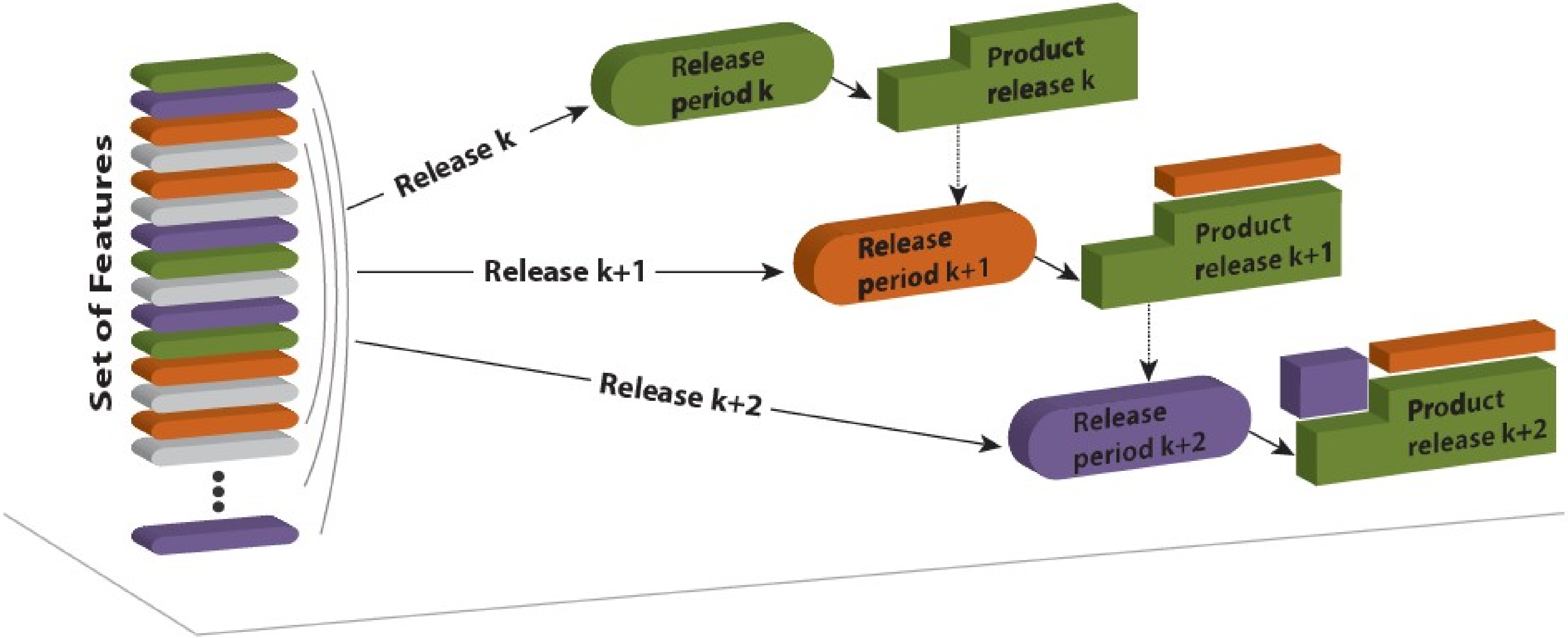
Release Planning

Paper [RP] in compendium

- The art and science of software release planning
- Ruhe, G., & Saliu, M. O.
- IEEE software, 22(6), 47-53. 2005

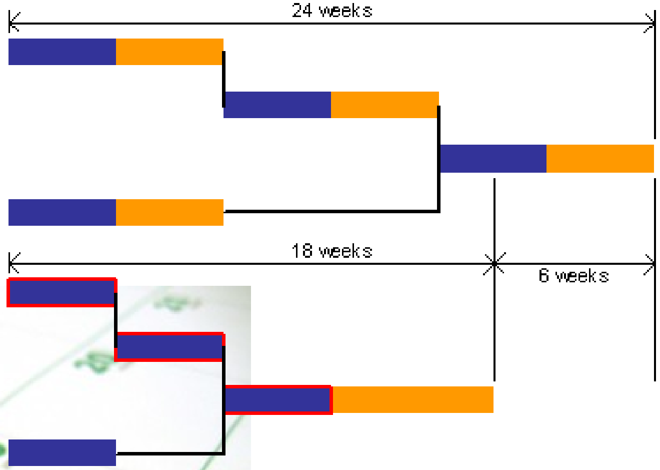
What is Release Planning?



[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:

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

[RP]


1.000.000.000.000
RELEASE PLANS
imgflip.com

Why greedy allocation is bad

https://gist.github.com/bjornregnell/80897de5b109f36c1b7ae29f43e4aa7b

```
val m = 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))
```

```
def plan(input: Model,
      pickNext: (Model,Release)=>Option[Feature]): Model = {
  var result = input
  releases(input).foreach { r =>
    var next = pickNext(result, r)
    while (next.isDefined) {
      result = allocate(result, next.get, r)
      next = pickNext(result, r)
    }
  }
  result
}

plan(m, random)
plan(m, greedy)
```

```
def features(m: Model): Vector[Feature] = m.tip.collect{case f: Feature => f}
def releases(m: Model): Vector[Release] = m.tip.collect{case r: Release => r}
def allocate(m: Model, f: Feature, r: Release): Model = m + (r has f)
def isAllocated(m: Model, f: Feature): Boolean = releases(m).exists(r => (m/r).contains(f))
def allocatedCost(m: Model, r: Release): Int = (m/r).entities.collect{case f => m/f/Cost}.sum
def isRoom(m: Model, f: Feature, r: Release): Boolean = m/r/Capacity >= allocatedCost(m,r) + m/f/Cost
def featuresInGreedyOrder(m: Model): Vector[Feature] = features(m).sortBy(f => m/f/Benefit).reverse

def random(m: Model, r: Release): Option[Feature] = scala.util.Random.shuffle(features(m)).
  filter(f => !isAllocated(m,f) && isRoom(m,f,r)).headOption

def greedy(m: Model, r: Release): Option[Feature] =
  featuresInGreedyOrder(m).find(f => !isAllocated(m,f) && isRoom(m,f,r))
```

Optimal vs. Greedy

```
val optimal = 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), Feature("c"), Feature("d"), Feature("e"), Feature("f"),
    Feature("g")),
  Release("r2") has (Capacity(90), Feature("h"), Feature("i"), Feature("j")))
```

```
def sumAllocatedBenefit(m: Model) =
  releases(m).map(r => (m/r).collect{case f: Feature => m/f/Benefit}.sum).sum

val beneftitOptimal = sumAllocatedBenefit(optimal)
val benefitGreedy   = sumAllocatedBenefit(plan(m,greedy))
val ratio = benefitGreedy.toDouble / beneftitOptimal
```

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

TODO!

- Skim read before exercise and lab next week:
  [AGRE, PROTO1 & 2] for exercise, [MDRE, INTDEP, RP, OSSRE] for lab 2
- Exercise this week on **prototyping** + **functional** requirements
  (Lau: 3-5 from last week)
- Hand in **Release R1**
- Book meeting with your supervisor

- **Next week**: note: only one lecture that week; topic: **Quality Requirements (QR)**:
  - Watch the QUPER-video (before or after the lecture)
    link to video on open course home page: https://cs.lth.se/krav/quality-requirements/
  - Come to the lecture on Tuesday in E:C as usual - any questions on QR are welcome
  - Do Exercise 4 where you work on QR in your project
  - Do Lab 2 (preferably in pairs) bring preparations
- Lab2 is next week but you need to start preparing **this week**...
  - Two parts: **Quality Requirements (QR)** and **Release Planning (RP)**
  - Preparations mean **a lot of reading + work** and take **significantly more time** compared to lab1