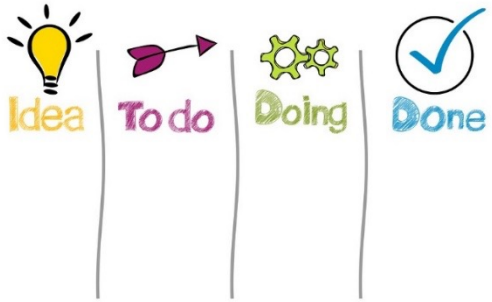


VO Softwareentwicklungsprozess

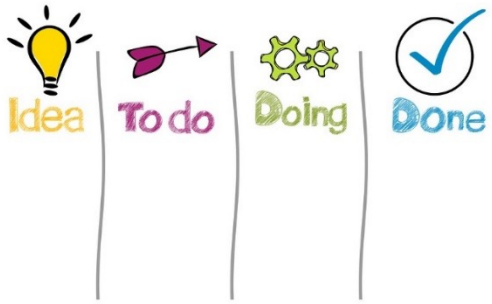
https://youtu.be/7rjBqmD88TQ

Alexander Felfernig und Trang Tran

Institut für Softwaretechnologie, Inffeldgasse 16b/2

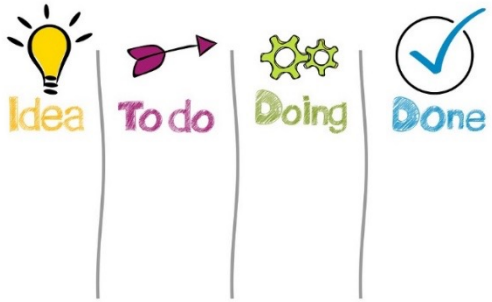
- Software Processes -

Why Do Software Projects Fail?

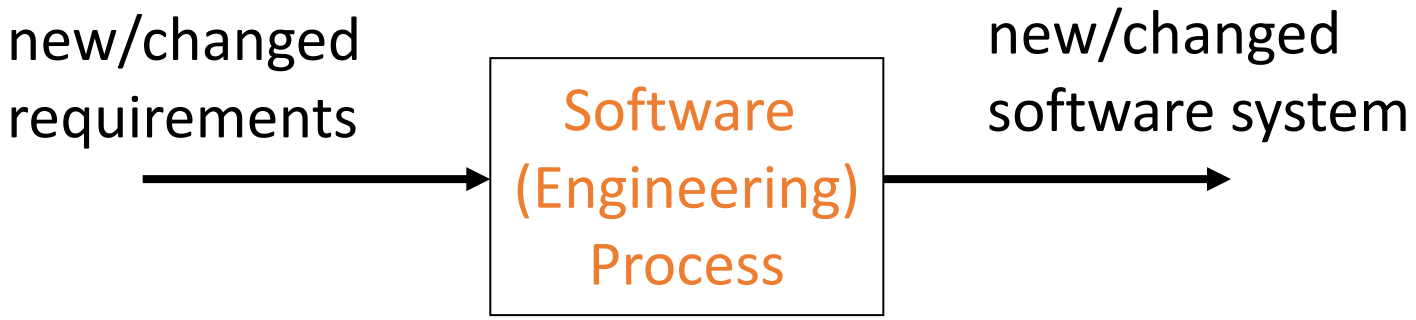


- Missing business case: no need to develop a software if the business case is not known!
- Requirements are regarded as a „wish-list“: wrong prioritization → opportunity costs, unsuccessful startup!
- No estimates, no focus!
- Late inclusion of personnel: binding of existing personnel!
- Unforeseen requirements: code rot / software erosion!
- Wrong development process!

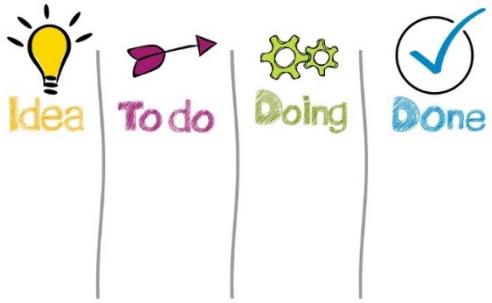
Software Processes



- Each software engineering book explains "software process" in different ways …
- A process exactly defines who is doing what, when, and how (in order to reach an objective).

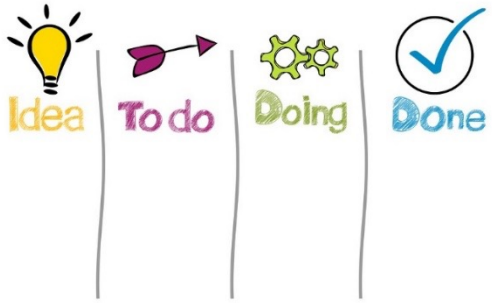


Software Processes: Basic Tasks



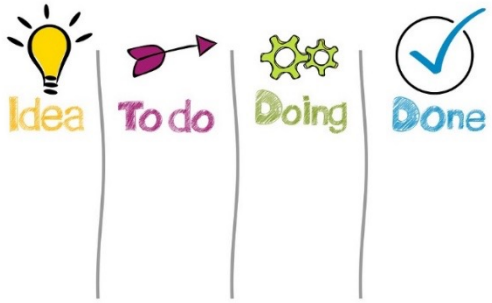
- Requirements Engineering: analyze and prioritize the requirements!
- Design and Implementation: designing the architecture and implement the prioritized requirements!
- Verification and Validation: is the software correct? Are the requirements fulfilled?
- Evolution: installation, maintenance, and extension to fulfill new requirements.

Software Processes: Insights



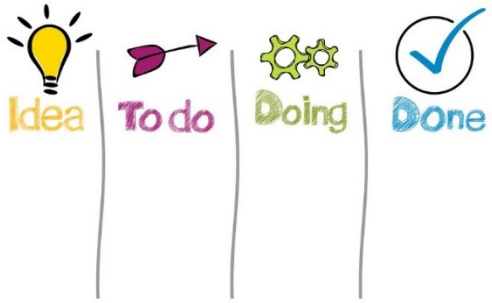
- Independent of the chosen process: stakeholders do not necessarily follow all procedures in detail, the most relevant ones are followed (ako “cherry-picking”)
- Prior to process models: „writing the code and fixing the problems“
- Two broad process types: sequential („complete one phase, then go to the next“) and evolutionary („follow an iterative approach“)

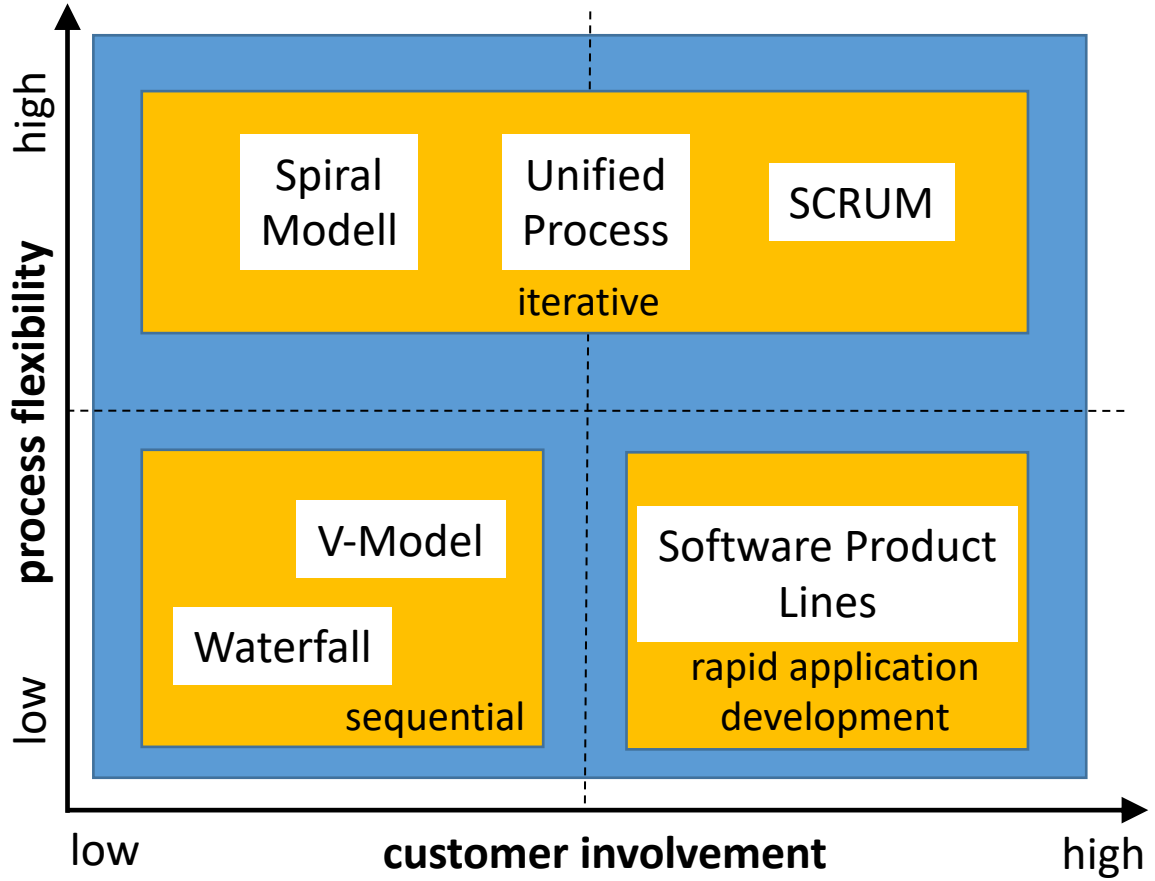
3 Generic Processes



- Waterfall/Sequential: classical lifecycle model (requirements → … → installation and maintenance)
- Evolutionary/Iterative: development & implementation of small increments on the basis of user feedback
- Rapid Application Development: reuse-driven component-oriented software development, software product lines

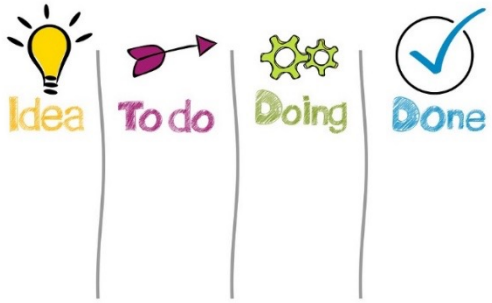
Example Processes: Overview

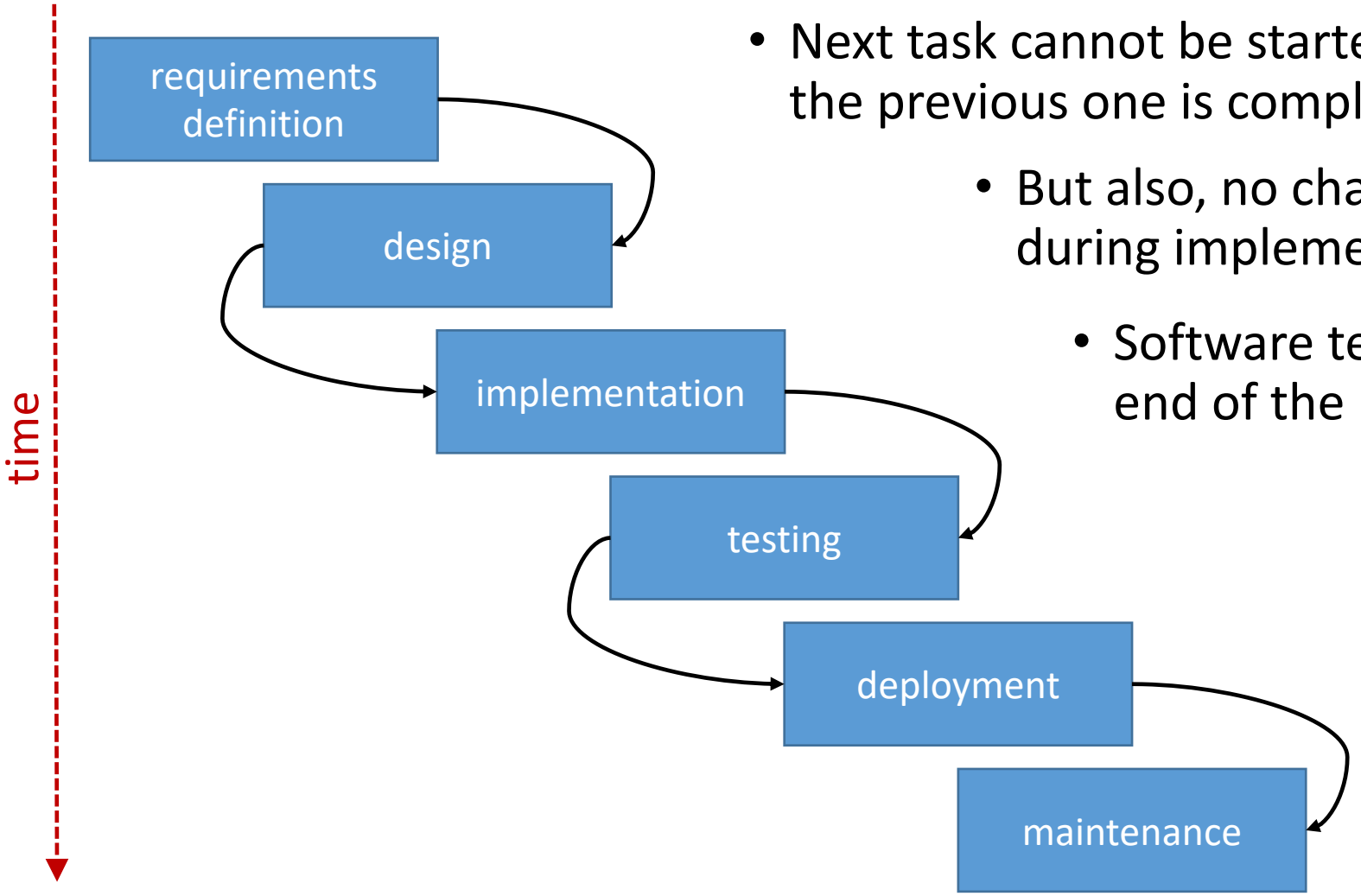




- Low process flexibility
  - linear character, easier to implement & manage
- High process flexibility
  - iterative character, management more complex
- Low customer involvement
  - requirements clear
- High customer involvement
  - requirements or features unclear
  - customers & communities are integrated more intensively

Example: Waterfall

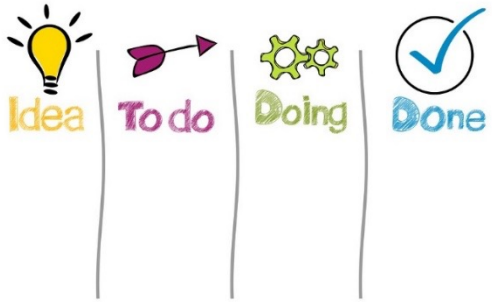


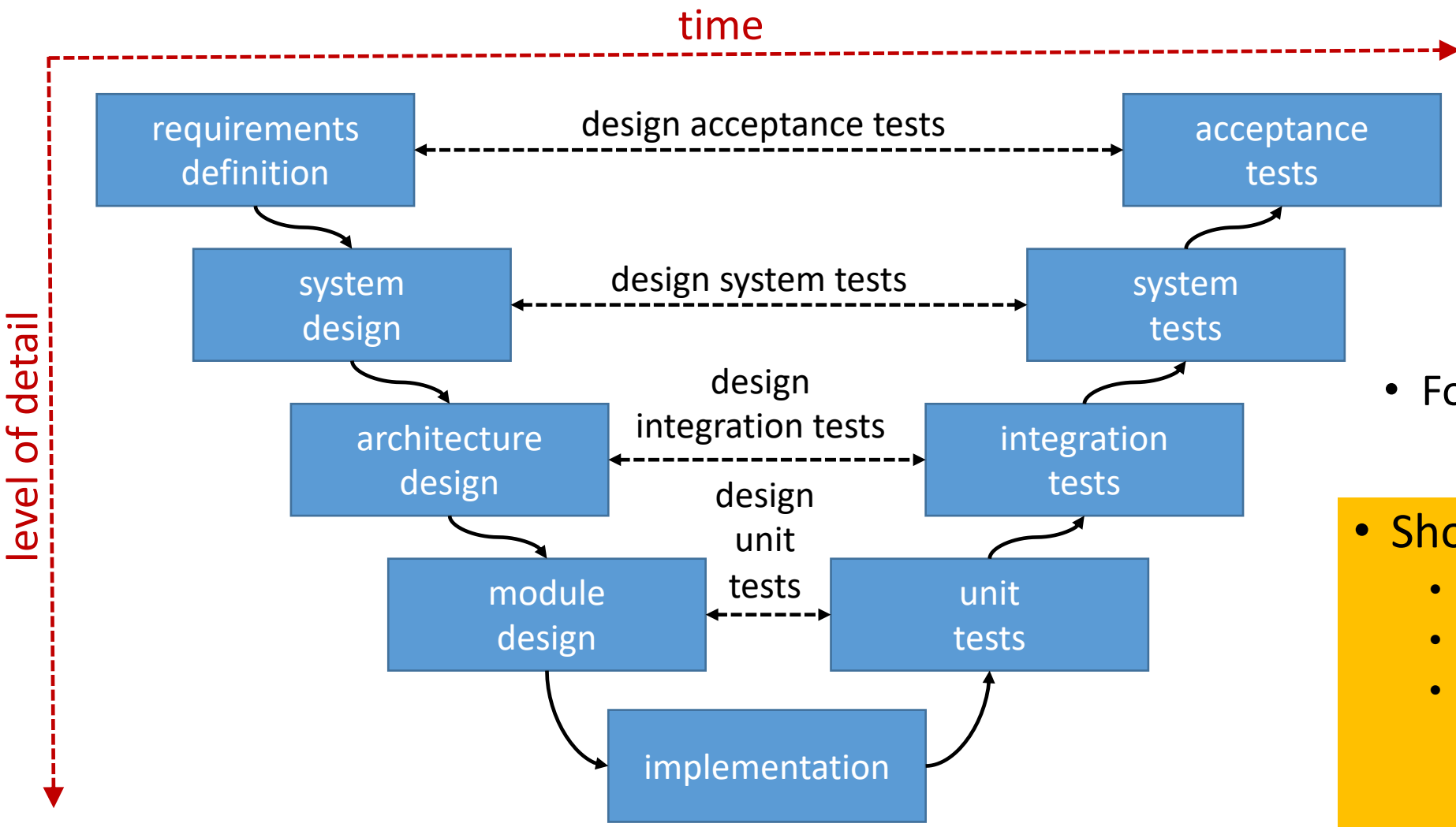


- Next task cannot be started before the previous one is completed
  - But also, no change of requirements during implementation phase
    - Software testing only possible near the end of the project („delayed testing“)

- Should be used when?
  - Clearly defined requirements, no changes, no risks
  - Well-known technological infrastructure (development environment, etc.)
  - Application: smaller projects, e.g., development of algorithms, compilers

Example: V-Model

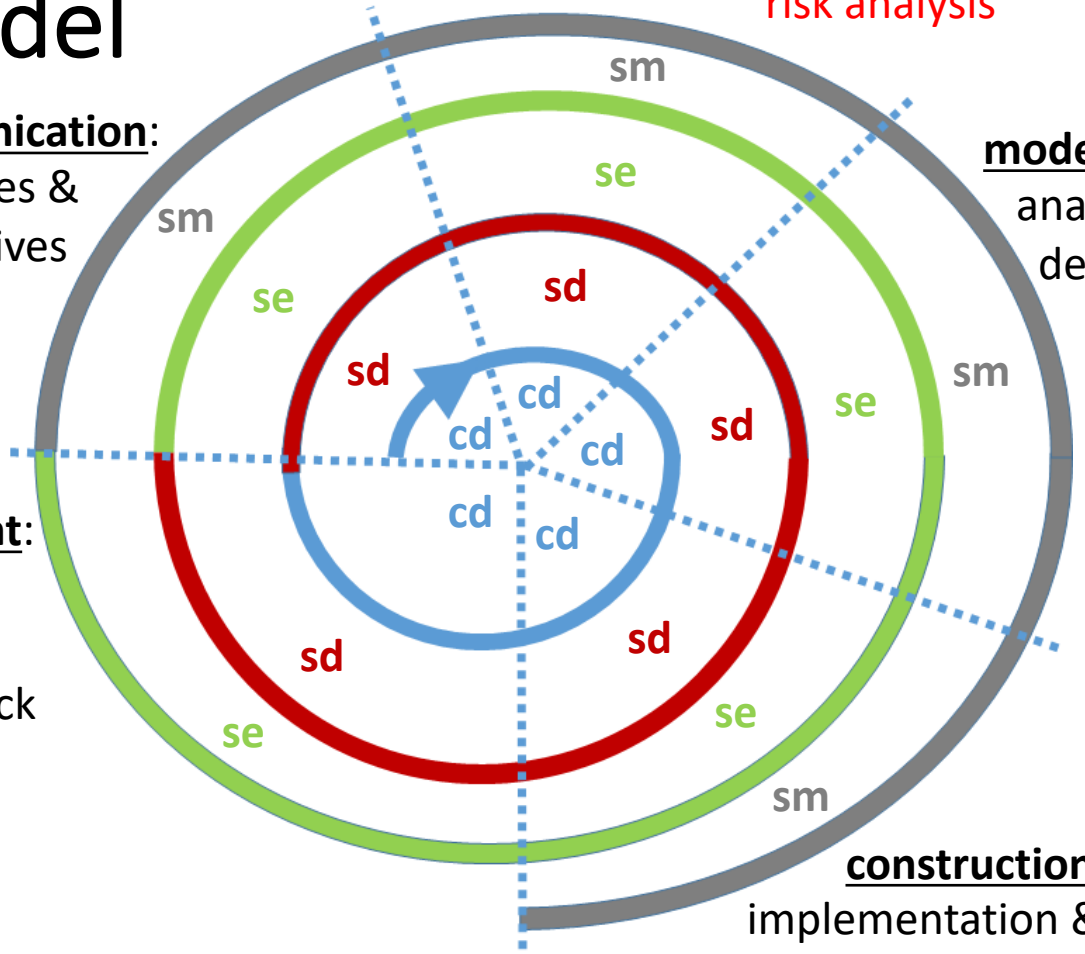


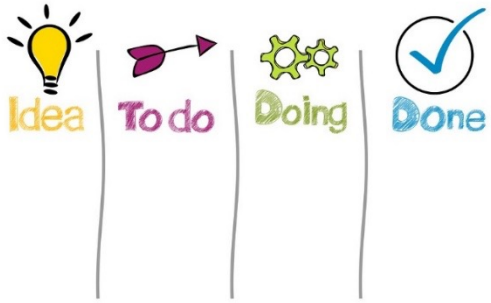


- Similar to Waterfall
- Additional test phases
- Focus on quality assurance

- Should be used when?
  - Unacceptable downtime
  - Unacceptable failures
  - Application: flight control, mechatronic systems, healthcare applications

Example: Spiral Model





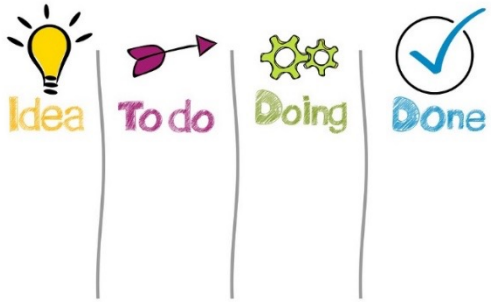
- focus on assessing risks
- intensive customer integration
- flexibility w.r.t. new requirements

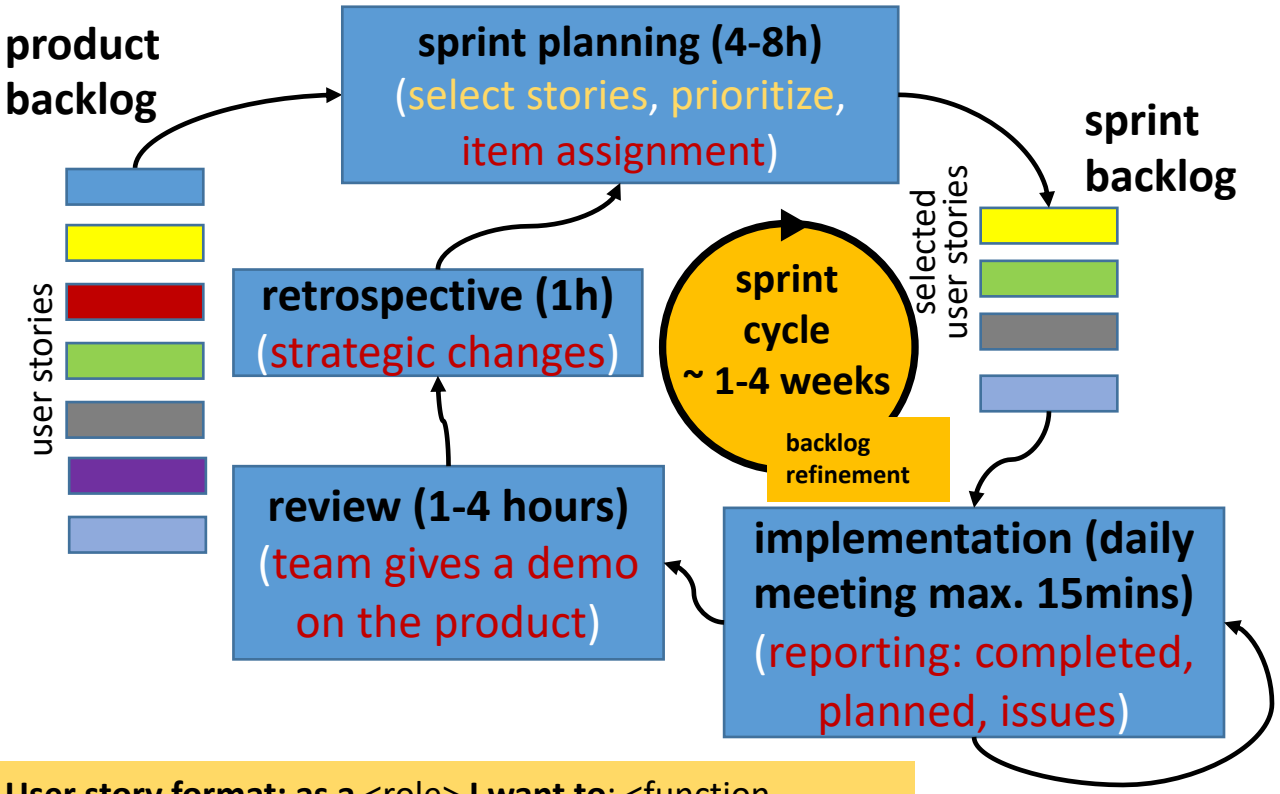
- Should be used when?
  - unclear business needs
  - large and complex project settings
  - research & development projects
  - new service development

Task areas:

concept development: cd
system development: sd
system enhancement: se
system maintenance: sm

Example: SCRUM





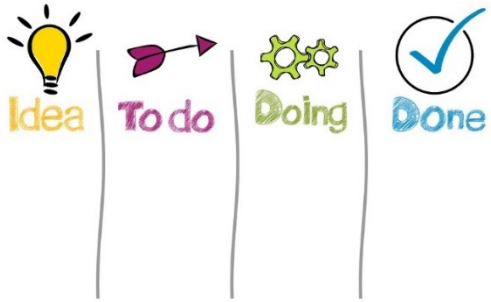
User story format: as a <role> **I want to**: <function-description> **so I can**: <statement about value>

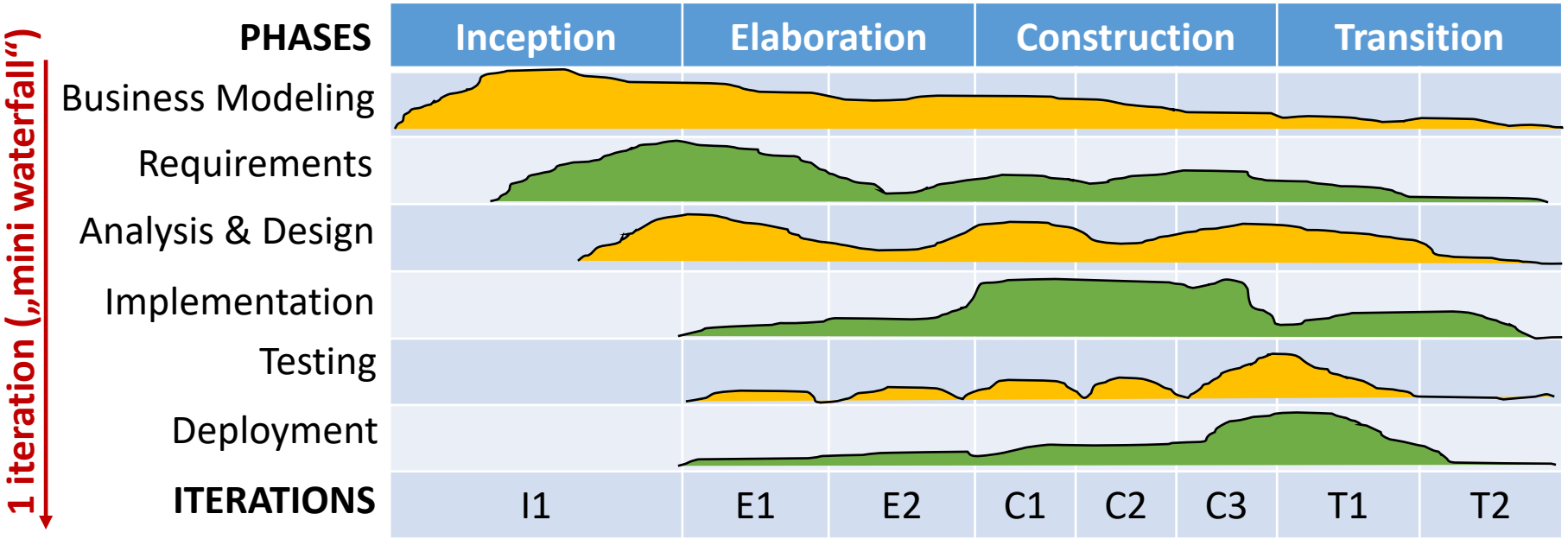
- Should be used when?
  - Startup-initiatives, early feedback required, unclear requirements
  - Larger projects with low degree of interdependencies

SCRUM roles

- product owner: represents customers, selects stories & prioritizes, owns backlog
- **small teams** (~7 persons, developers, analysts, QA), estimate/implement user stories, self-organized, work closely together in sprints
- Scrum master: coach, expert who proactively supports the team, does not assign tasks
- **product backlog**: desired items ~ user stories, e.g., features, bug fixes, …
- **sprint backlog**: todos for the sprint
- **task board**: done, in progress, 2b started
- **optimization**: DevOps

Example: Rational Unified Process (UP)

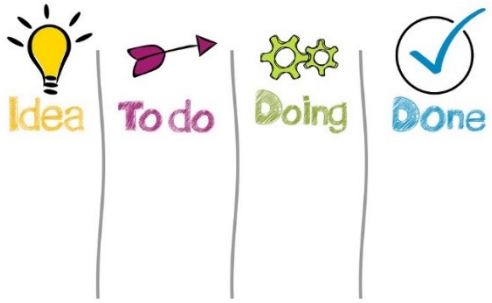




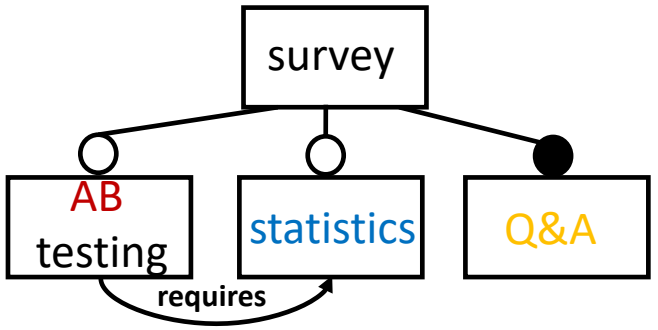
- iterative process with sequential elements
- 4 phases + milestones
- 1 iteration = „mini waterfall“
- a couple of iterations per phase

- Should be used when?
  - Use-case based development (UML)
  - High-risk and large projects

Example: Software Product Lines (SPL)



domain analysis



software configuration

Configure your survey tool!

Include A/B Tests? ○

Include Statistics ◉

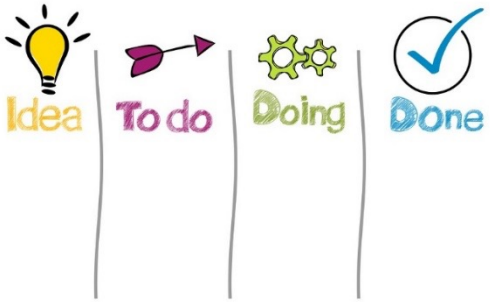
software generation

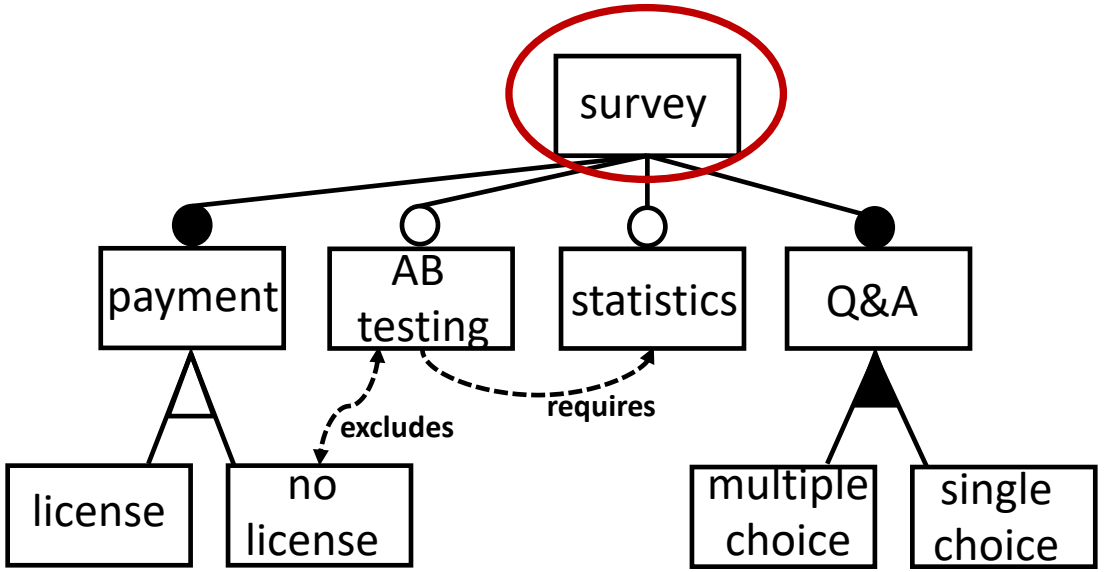
```
public survey(boolean hasABtesting,
              boolean hasStatistics) {
    QA qa = new QA();
    if(hasABtesting){ABTesting ab = new ABTesting();}
    if(hasStatistics){Statistics stat = new Statistics();}
}
```

- Focus on flexibility w.r.t. future customer requirements
- Variations are modeled as central part of the software
- Product family: set of software products with the same name
- Variability described by feature models

- Should be used when?
  - Startup-initiatives, early feedback required
  - Mass customization software for specific market segments

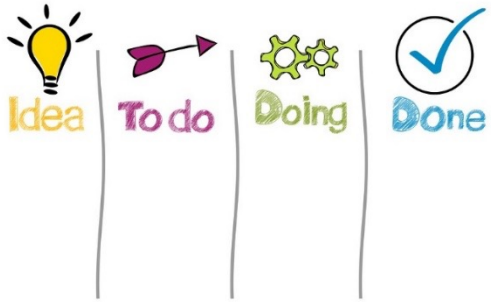
Feature Models

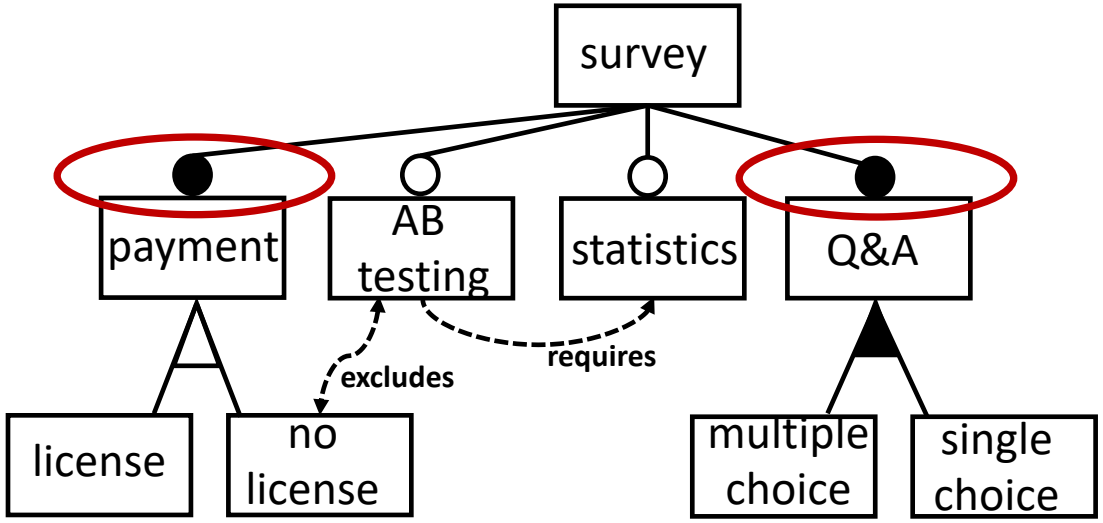




- Features indicate whether a specific function is supported
- A feature can either be „true“ (selected) or „false“ (unselected)
- Feature models can be formalized and used as a basis for software configuration
- „root“ feature (e.g., „survey“) is selected in every configuration (survey = true)

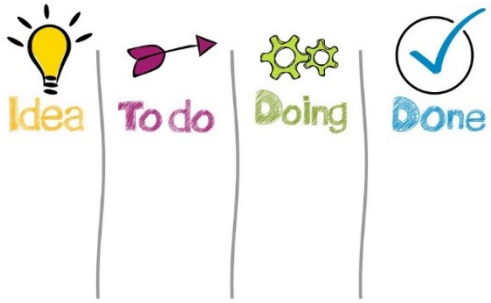
Mandatory Relationships

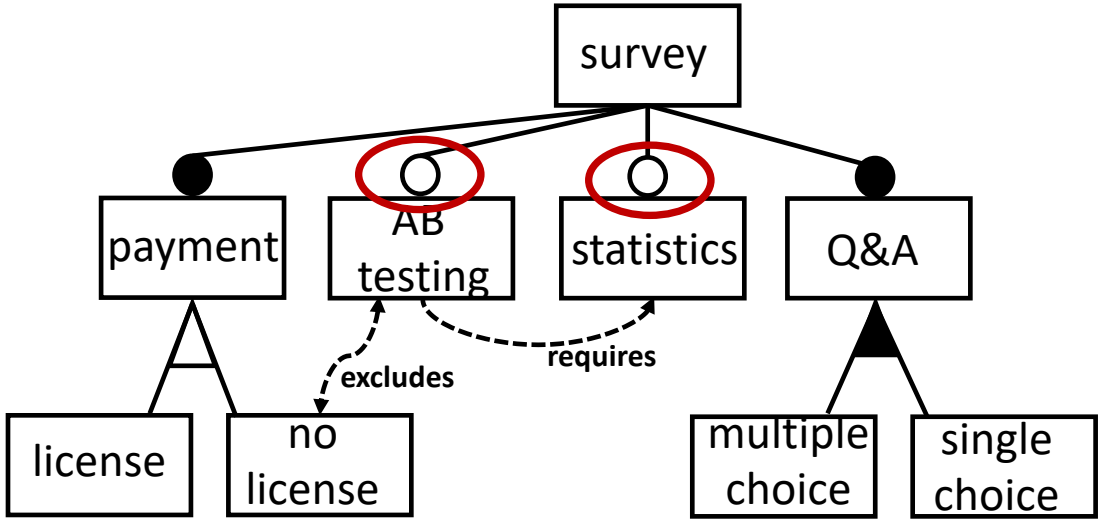




- Features „payment“ and „Q&A“ have to be selected
- more formally:
  - payment ↔ survey
  - Q&A ↔ survey

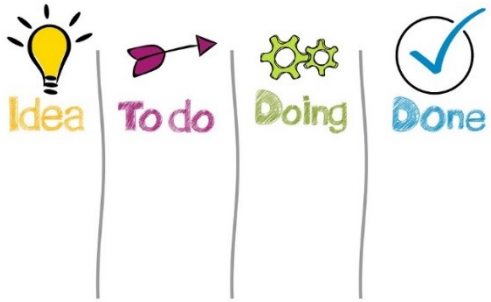
Optional Relationships

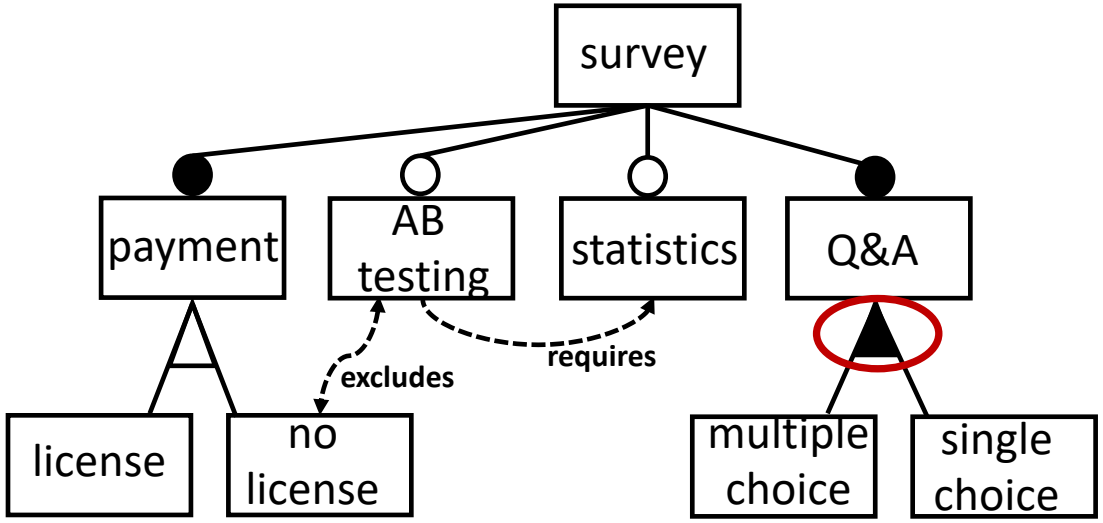




- Features „AB testing“ and „statistics“ could be selected
- more formally:
  - statistics → survey
  - AB testing → survey

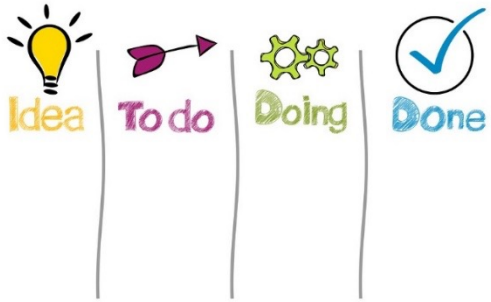
OR Relationships

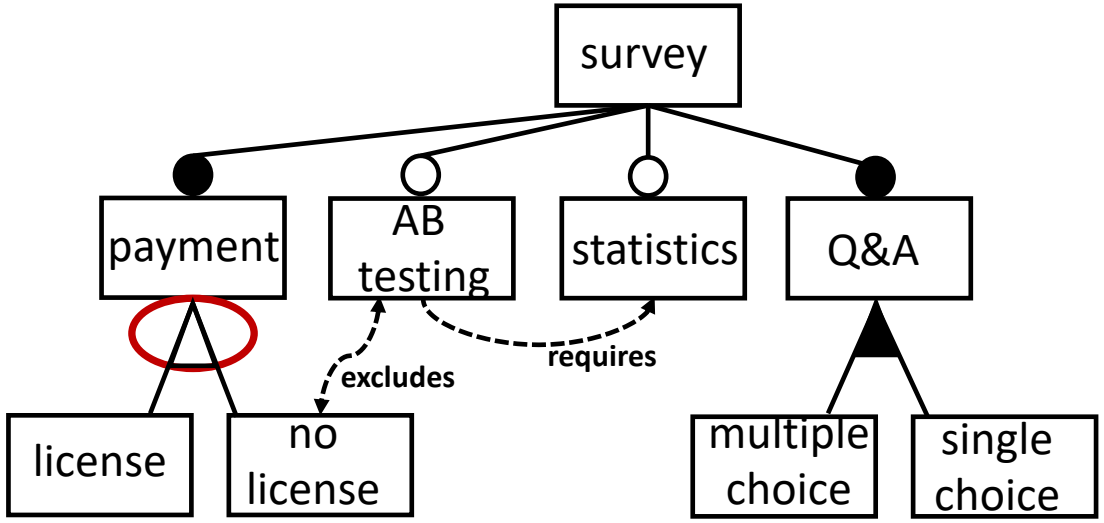




- At least one out of „multiple choice“ or „single choice“ has to be selected
- more formally:
  - Q&A $\leftrightarrow$ (multiple choice $\vee$ single choice)

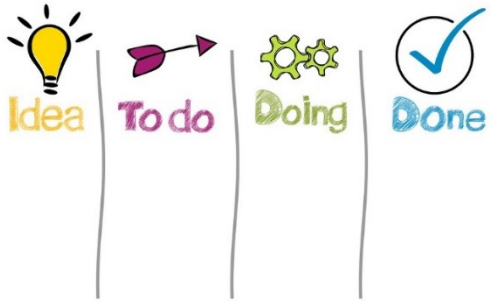
Alternative Relationships

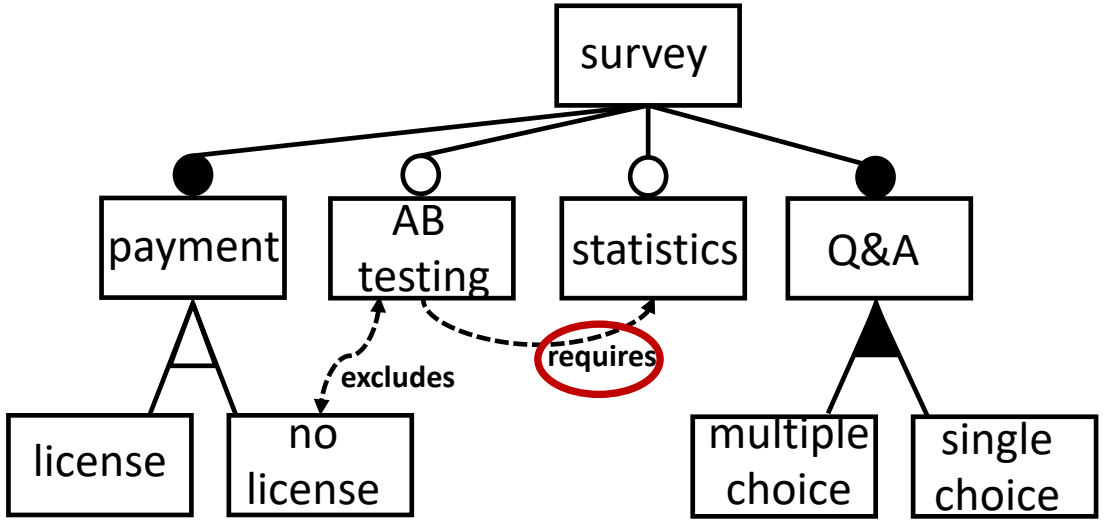




- Exactly one out of „license“ or „no license“ has to be selected (XOR semantics)
- more formally:
  - (license $\leftrightarrow$ (¬no license $\wedge$ payment)) $\wedge$ (no license $\leftrightarrow$ (¬license $\wedge$ payment))

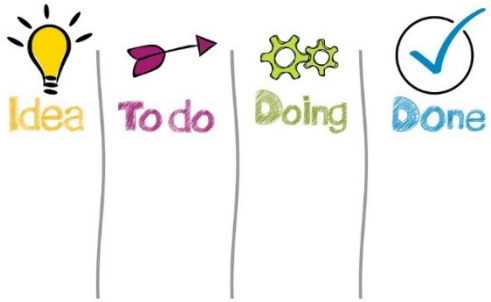
Requires Relationships

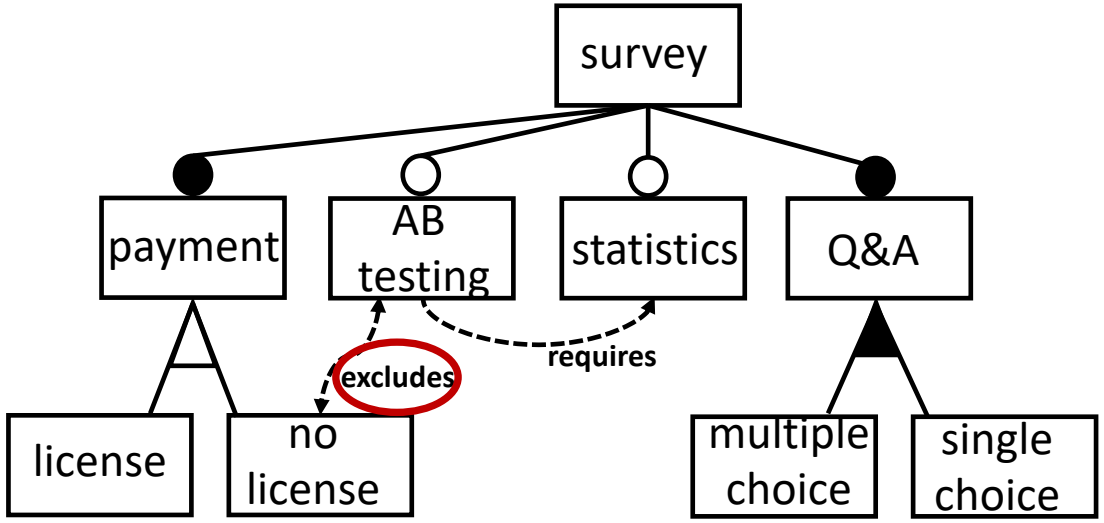




- „Requires“ = cross-tree constraint
- The inclusion of feature X requires the inclusion of Y
- more formally:
  - AB testing → statistics

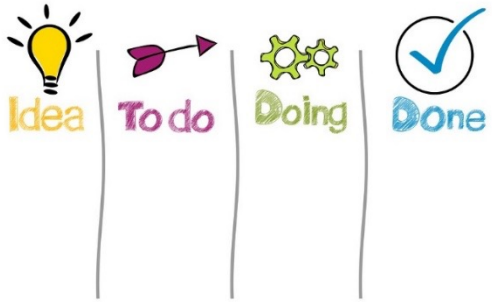
Excludes Relationships



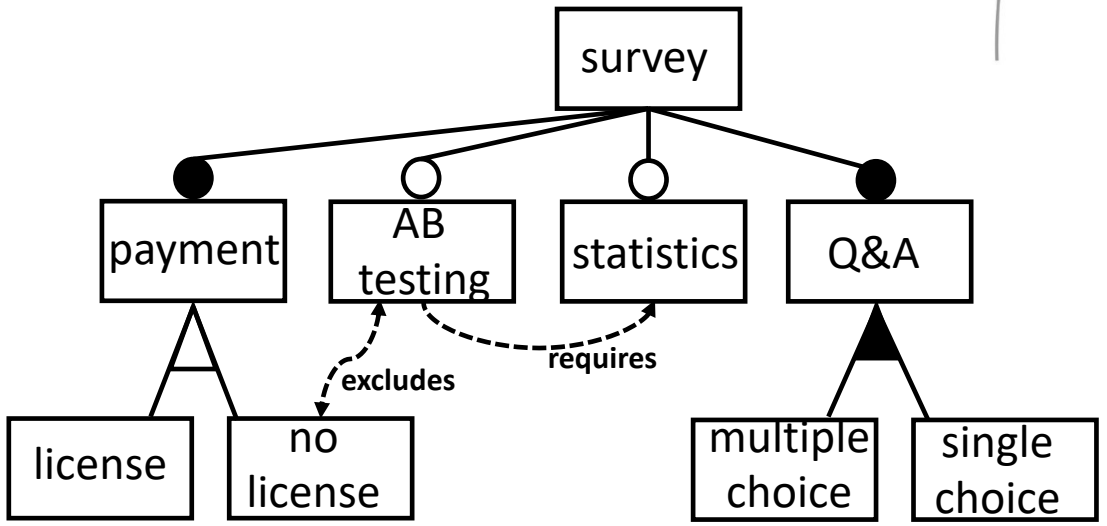


- „Excludes“ = cross-tree constraint
- Features X and Y must not be included in the same configuration
- more formally:
  - $\neg$(no license $\wedge$ AB testing)

Feature Model & Knowledge Base



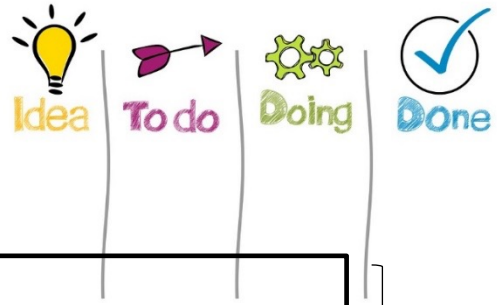
- survey = true
- payment $\leftrightarrow$ survey
- Q&A $\leftrightarrow$ survey
- statistics $\rightarrow$ survey
- AB testing $\rightarrow$ survey



- Q&A $\leftrightarrow$ (multiple choice $\vee$ single choice)
- (license $\leftrightarrow$ (¬no license $\wedge$ payment)) $\wedge$ (no license $\leftrightarrow$ (¬license $\wedge$ payment))
- AB testing $\rightarrow$ statistics
- $\neg$(no license $\wedge$ AB testing)

Knowledge base can be implemented with constraint solvers such as minizinc (www.minizinc.org)

MiniZinc Implementation



V+ D

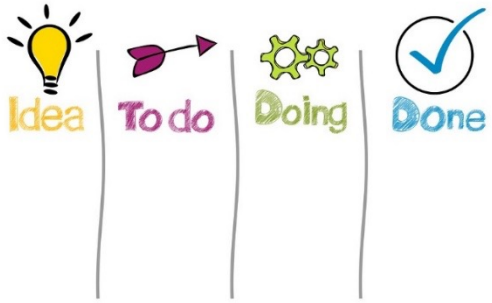
```
var bool: survey;
var bool: payment;
var bool: ABtesting;
var bool: statistics;
var bool: QA;
var bool: license;
var bool: nolicense;
var bool: multiplechoice;
var bool: singlechoice;
```

C

```
constraint survey = true;
constraint payment <-> survey;
constraint QA <-> survey;
constraint statistics -> survey;
constraint ABtesting -> survey;
constraint QA <-> (multiplechoice \/ singlechoice);
constraint (license <-> (not nolicense /\ payment))
/\ (nolicense <-> (not license /\ payment));
constraint ABtesting -> statistics;
constraint not(nolicense /\ ABtesting);
solve satisfy;
```

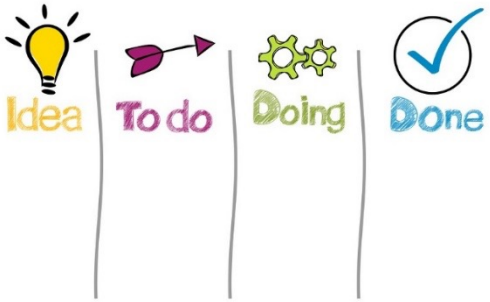
Constraint Satisfaction Problem (CSP): set of finite domain variables **V**=$\{v_1,..,v_n\}$, corresponding domain definitions **D**=$\{dom(v_1),..,dom(v_n)\}$, and a set of constraints **C**=$\{c_1..c_m\}$

Repetition (R1)

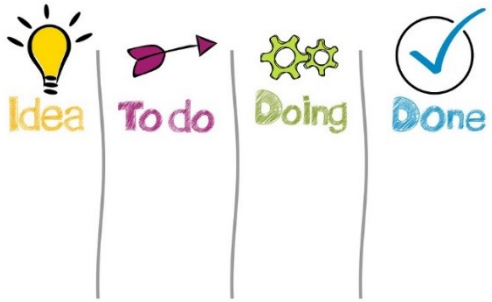


- Visit: https://checkr.tugraz.at/ (a TU Graz software).
- Login with your TU Graz student account (single sign-on supported).
- Enter the following participation code: **rJACCg** (note: you can try to answer the individual questions as often as you like!). No fixed time slots for the repetitions, **deadline for all repetitions**: **June 20th, 23:59:59.**
- Go to the category „Software Processes“ and answer the questions.
- Your answers will be taken into account as mentioned in the organization slides.

References



- **[BNE2007]** G. Booch, R. Maksimchuck, M. Engle, B. Young, J. Conallen, and K. Houston. Object-Oriented Analysis and Design with Applications, Addison Wesley, 2007.
- **[BSR2010]** D. Benavides, S. Segura, and A. Ruiz-Cortes, Automated analysis of feature models 20 years later: A literature review, Information Systems, 35(6):615-636, 2010.
- **[MMO2014]** S. Misra , M. Omorodion, L. Fernandez-Sanz. Overview on Software Process Models, their Benefits and Limitations, DOI: 10.4018/978-1-4666-5182-1.ch015, IGI Global, 2014.



Thank You!

Univ.-Prof. DI Dr. Alexander Felfernig
Dr. Trang Tran
Applied Artificial Intelligence
Graz University of Technology, Austria