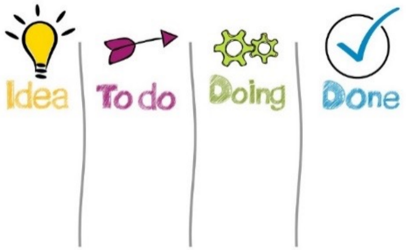
VO Softwareentwicklungsprozess



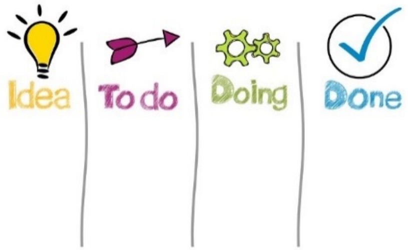
https://youtu.be/h0DJzBm-zJo

Alexander Felfernig und Trang Tran

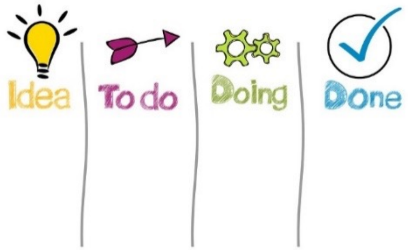
Institut für Softwaretechnologie, Inffeldgasse 16b/2

- Software Process & Product Quality -

Software Quality

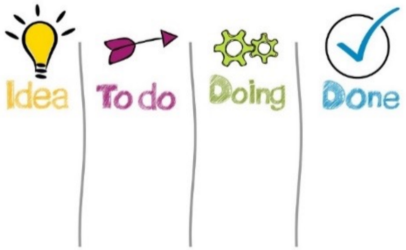


- Desirable properties of process & product
- **Aspects**:
  - Process quality: punctuality, product impact, …
  - Product quality: usability, maintainability, …
- Positive correlation between process & product quality
- **Maturity models**: measure process quality, certification aspect (examples: CMM(I), SPICE)
- **Metrics**: measure artifact quality, e.g., metrics for requirements quality and code complexity



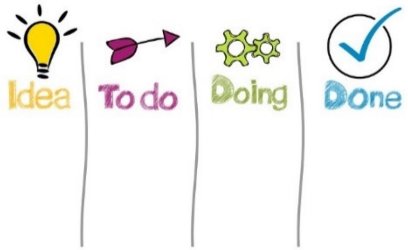
Process Maturity Models

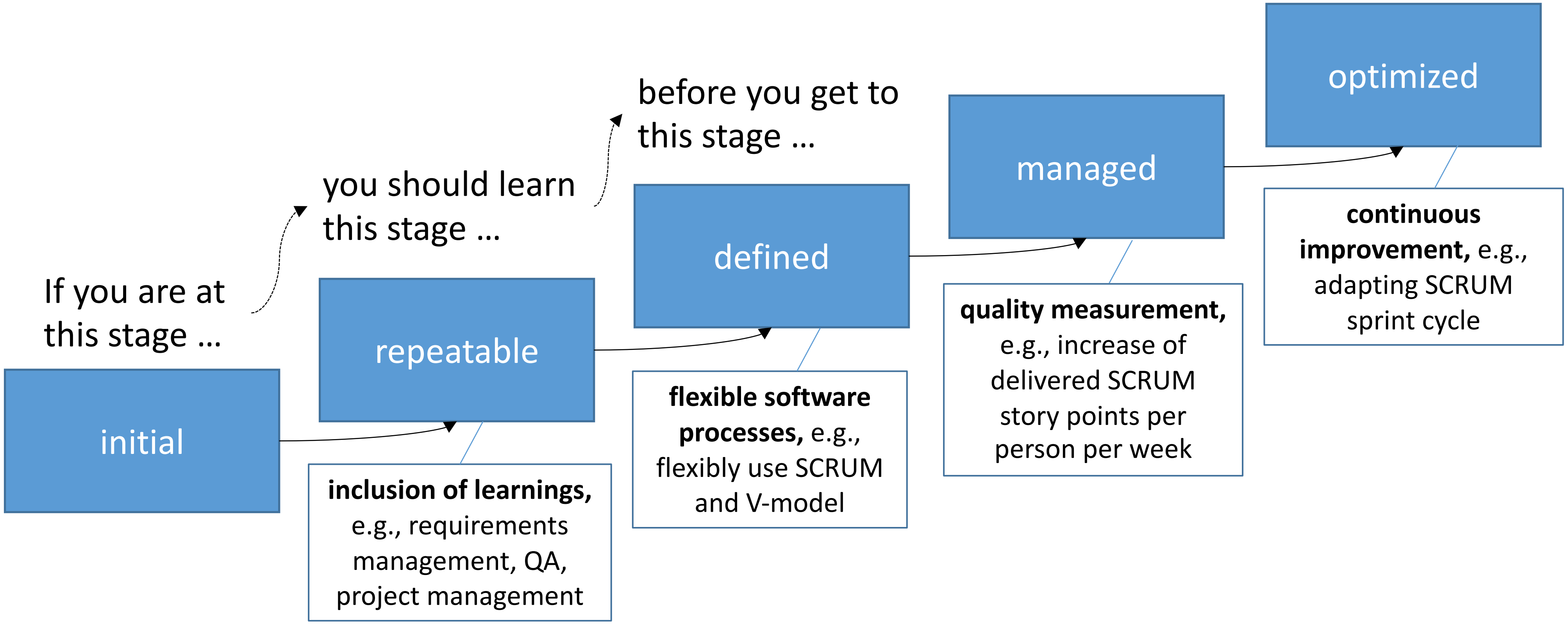
Software Process Maturity



- Process maturity models/frameworks: help to assess the effectiveness of a process and which capabilities are needed for the next level
- Guidance on how to gain control of processes
- Different levels with increasing process maturity
- Example maturity model: CMM(I) (“I” = “integrated”, further development of CMM)

CMM(I): Getting to the Next Level





If you are at this stage …

you should learn this stage …

before you get to this stage …

initial

repeatable

defined

managed

optimized

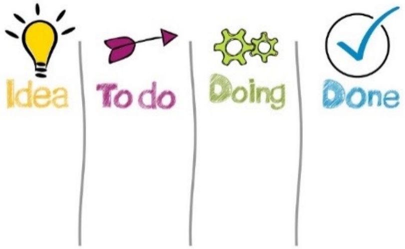
inclusion of learnings, e.g., requirements management, QA, project management

flexible software processes, e.g., flexibly use SCRUM and V-model

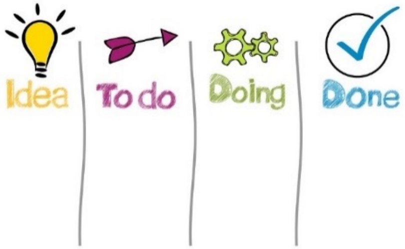
quality measurement, e.g., increase of delivered SCRUM story points per person per week

continuous improvement, e.g., adapting SCRUM sprint cycle

The 5 CMM Levels

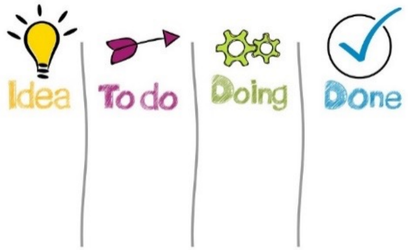


ID	Level	Example Aspects
1	**initial**	no process, no learnings, “ad-hoc” software development, artefacts not provided in-time, success due to “heroes”
2	**repeatable**	learnings from the past, some process areas are organized, e.g., project & requirements engineering, quality assurance
3	**defined**	(adaptive) software process, i.e., flexibility with regard to new types of projects, organizational training
4	**managed**	quality measurement, e.g., measurement of process productivity and developed artifacts
5	**optimized**	continuous improvement, e.g., strength/weaknesses analysis

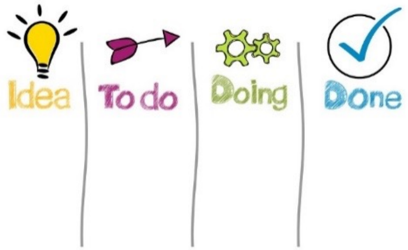


Metrics

Software Process Tasks & Metrics

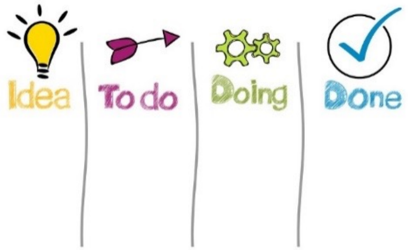


Process Task	Example Artefacts	Example Metric
Req. Engineering (RE)	Textual Requirements	**Quality of Requirements**
Design	Class Hierarchies	**OO Design Metrics**
Implementation	Code	**Complexity Metrics**
Validation	User Interface	**System Usability Scale**
Verification	Test Suite	**Coverage Metrics**
Evolution	Runtime	**Stability & Performance**

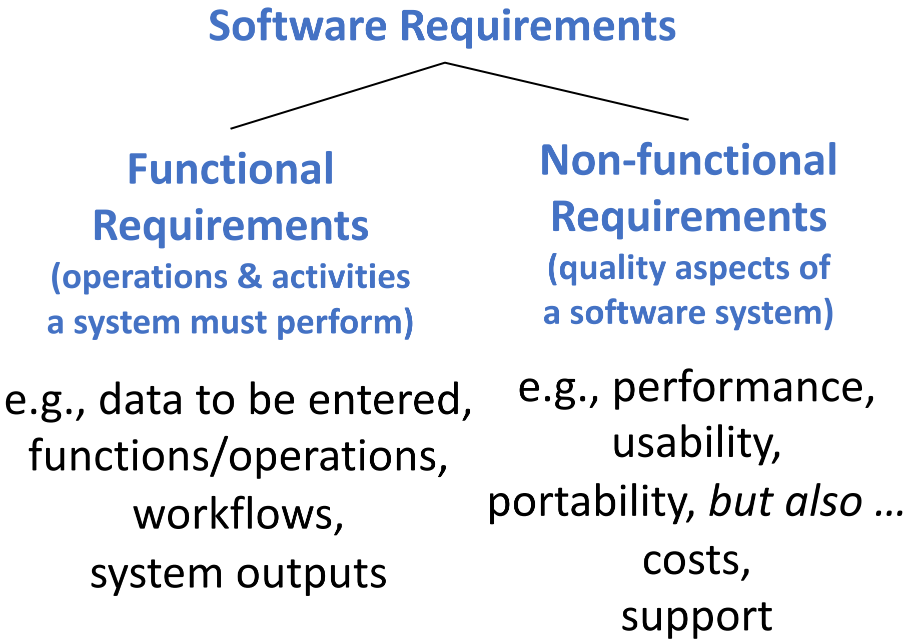


Quality of Requirements

Requirements



A software requirement describes what a customer or user expects from a product in terms of conditions, properties, goals, benefits.



Software Requirements

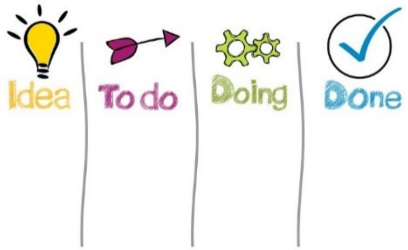
Functional Requirements (operations & activities a system must perform)

e.g., data to be entered, functions/operations, workflows, system outputs

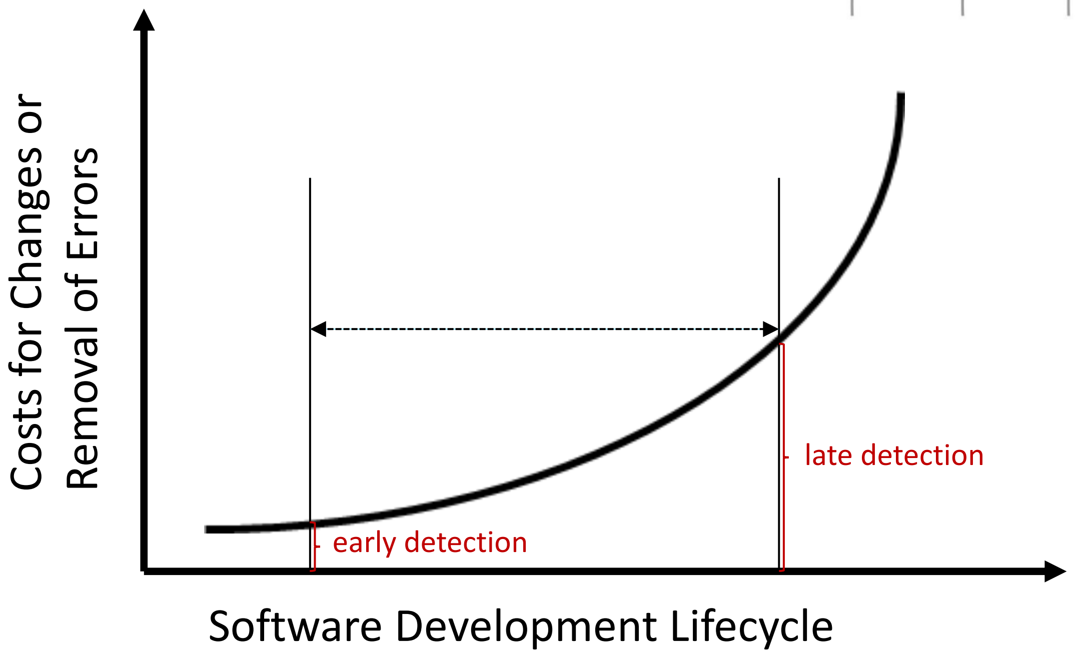
Non-functional Requirements (quality aspects of a software system)

e.g., performance, usability, portability, *but also …* costs, support

Why Requirements Engineering (RE)?

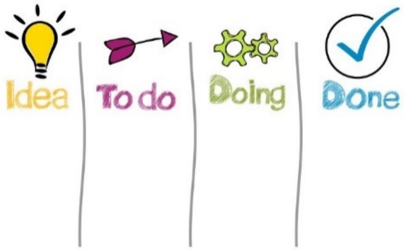


- 39% of projects completed successfully
- 18% cancelled
- 43% completed over budget
- Avg. RE project efforts: 2-5%
- But: erroneous requirements have the worst effect …



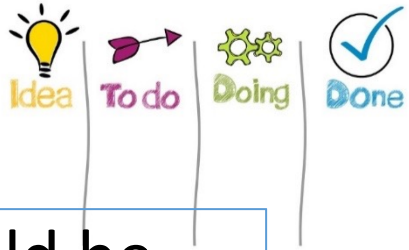
- **Savings (RE)**: reduced development & adaptation + reduced error correction + reduced opportunity costs (up to 40% of total costs!)

Quality Characteristics of Requirements



- Unambiguity
- Testability (verifiable)
- Clarity (simple, concise)
- Correctness
- Understandability
- Feasibility (possible, realistic)
- Independence
- Atomicity
- Necessity
- Implementation-freeness (abstract)

Property: Unambiguity

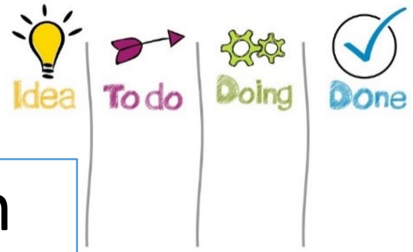


- **Example**: „no passwords longer than 25 characters should be accepted by the system“.
- **Issues**: more than 25 not enterable? The system should show an error message? Text should be truncated?

- **Example‘ (better)**: „no passwords longer than 25 characters should be accepted. If a user tries to exceed this limit when entering a password, the system should show an error message and inform the user to change the entered password correspondingly“.

Dimension	Calculation (basic)
Unambiguity [0..1]	$1 - \frac{\#stakeholders\ who\ evaluated\ R\ as\ ambiguous}{\#stakeholders\ who\ evaluated\ R}$

Property: Testability

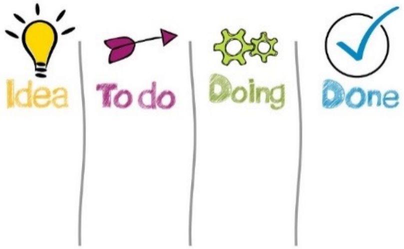


- **Example**: „In the customer administration, customer search should be supported on the basis of surname, address, etc.“
- **Issues**: what is meant with „etc.“? Tester should be able to check if the requirement has been implemented correctly.
- **Sources of limited testability**: **non-specific terms** (e.g., „etc.“, „and/or“), **adverbs** (e.g., „quickly“), **adjectives** (e.g., „maintainable“), and **vague words** (e.g., „manage“)

- **Example‘**: In the customer administration, customer search should be supported on the basis of the attributes surname, address, and age.

Dimension	Calculation (basic)
Testability [0..1]	$1 - \frac{\#non - specific\ words\ in\ R}{\#words\ in\ R}$

Property: Clarity

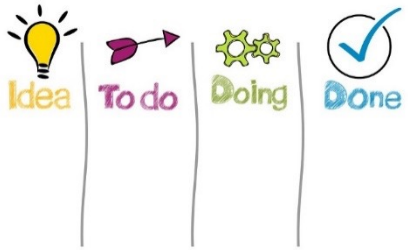


- **Example**: When searching for an invoice, customers typically know their customer-id, but sometimes not, then the system should prove it‘s flexibility and allow to search for the invoice based on the customer‘s surname + date of purchase.
- **Issues**: which search criteria are allowed?

- **Example‘**: When searching for an invoice, the system shall identify the invoice based on the customer-id or the customer‘s surname + date of purchase.

Dimension	Calculation (basic)
Clarity [0..1]	$\frac{\#stakeholders\ who\ evaluated\ R\ as\ clear}{\#stakeholders\ who\ evaluated\ R}$

Property: Correctness

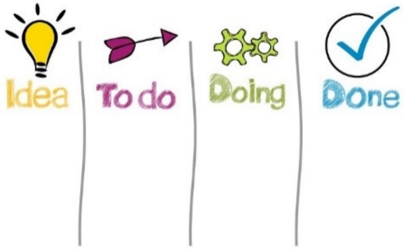


- **Example**: the price shown should include 19% value-added tax.
- **Issues**: the application is developed for a customer in Austria.

- **Example‘**: the price shown should include value-added tax which is country-specific (see the following table …).

Dimension	Calculation (basic)
Correctness [0..1]	$\frac{\#stakeholders\ who\ evaluated\ R\ as\ correct}{\#stakeholders\ who\ evaluated\ R}$

Property: Understandability

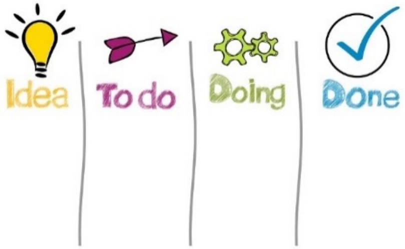


- **Example**: Customer must enter user-id and password before switches to payment view.
- **Issues**: grammatical errors …

- **Example‘**: Customers must enter their user-id and password before switching to the payment view.

Dimension	Calculation (basic)
Understandability [0..1]	$1-\frac{\lvert\{s\ in\ sentences: erroneous(s)\}\rvert}{\lvert\{s\ in\ sentences\}\rvert}$

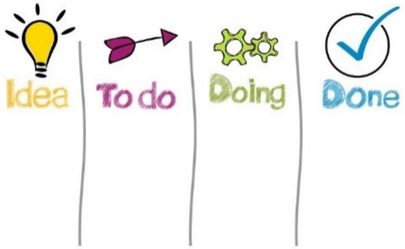
Property: Feasibility



- **Example**: the system shall provide a UI that enables the recommendation of items based on automated emotion detection.
- **Issue**: the requirement may not be feasible with the given resources.

Dimension	Calculation (basic)
Feasibility [0..1]	$\frac{\#stakeholders\ who\ estimated\ R\ as\ feasible}{\#stakeholders\ who\ estimated\ R}$

Property: Independence

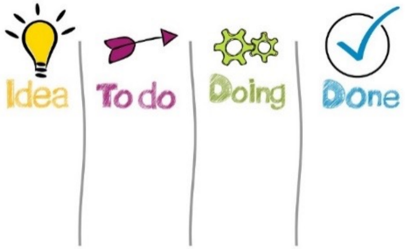


- **Example**: **Requirement 1 (R1)**: the list of recommended items shall include item name and item price. **Requirement 2 (R2)**: it shall be sorted by item name.
- **Issue**: in order to understand R2, R1 has to be understood as well. If the order of requirements changes, R2 cannot be understood anymore!

- **Example‘**: R2: the list of recommended items shall be sorted by item name.

Dimension	Calculation (basic)
Independence (R1,R2)	$1 - \frac{\lvert keywords(R1) \cap keywords(R2) \rvert}{\lvert keywords(R1) \cup keywords(R2) \rvert}$

Property: Atomicity



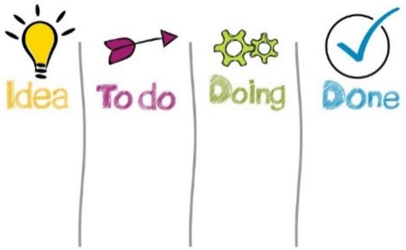
- **Example**: **R1**: the system shall support customer search, store new customers, delete customers, and generate invoices.
- **Issue**: the requirement is not a single traceable element!

- **Example**: **R1**: the system shall allow to search for customers. **R2**: the system show allow to store new customers. **R3**: the system shall allow to delete customers. **R4**: the system shall allow to generate invoices.

Dimension	Calculation (basic)
Atomicity – cohesion (s1,s2)	$\frac{\lvert keywords(s1) \cap keywords(s2)\rvert}{\lvert keywords(s1) \cup keywords(s2)\rvert}$

Remark: si can be regarded as sentences or subclauses.

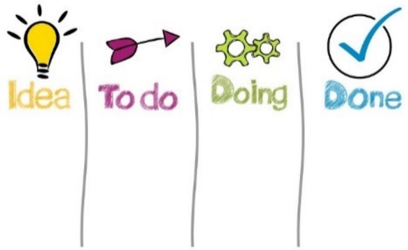
Property: Necessity



- A requirement is unnecessary, if none of the stakeholders is in the need of this requirement.
- **Example**: additional requirements are added by developers under the assumption that stakeholders could need this.

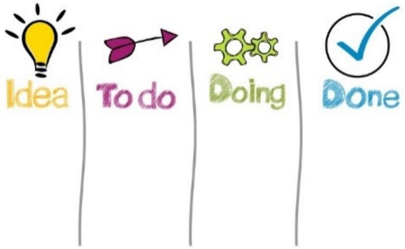
Dimension	Calculation (basic)
Necessity (support)	$\frac{\#stakeholders\ who\ are\ accepting\ R}{\#stakeholders\ who\ analyzed\ R}$

Property: Implementation-freeness



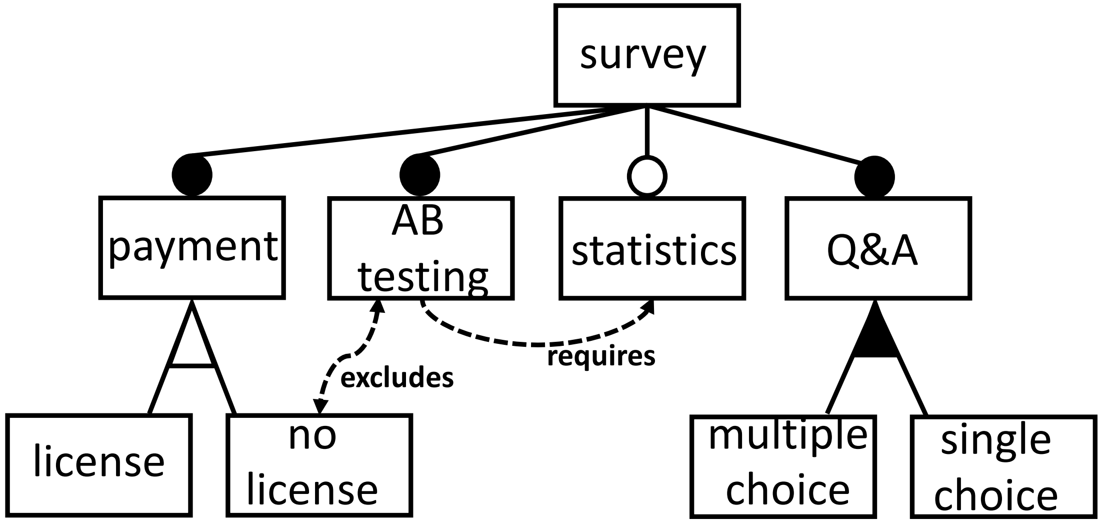
- Requirements should not include unnecessary implementation and design details. Focus on the „what“ but not on the „how?“
- Example: The system shall support the collection of customer data which are then stored in a MySQL table.
- Example‘: The system shall support the collection and storage of customer data.

Dimension	Calculation (basic)
Implementation-freeness	$1 - \frac{\#extracted\ technical\ keywords}{\#extracted\ keywords}$

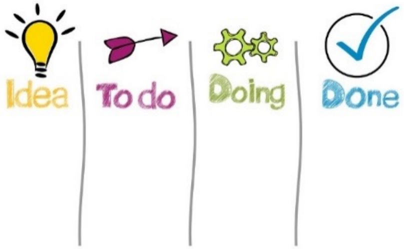


Quality Metrics for Feature Models

Feature Model Quality

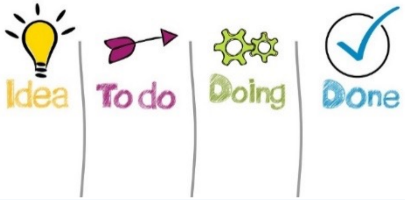


Name	Metric	Example
dead features f in feature model FM	$\frac{\|\{f\ in\ features(FM): no\ solution\ (\{f\ =\ true\} \cup\ relationships(FM))\}\|}{\|\{f\ in\ features(FM)\}\|}$	feature „no license“
false optional features f in feature model FM	$\frac{\|\{f\ in\ features(FM): no\ solution\ (\{f\ =\ false\} \cup\ relationships(FM))\}\|}{\|\{f\ in\ features(FM)\}\|}$	feature „statistics”

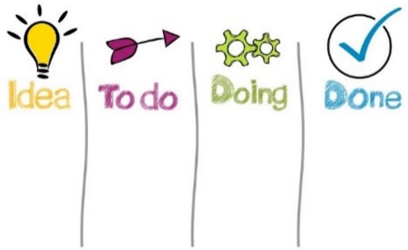


OO Design Metrics

OO Design & Implementation Metrics

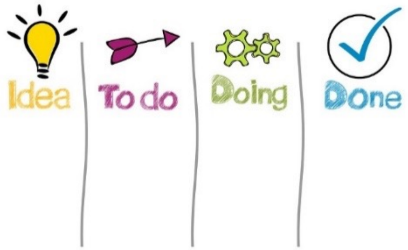


Name	Metric	Impact
avg. methods (m) per class c (MC)	$\frac{\sum_{c\ in\ classes} \lvert\{m: m\ in\ c\}\rvert}{\lvert classes\rvert}$	the more methods in c, the higher the adaptation efforts in c and c's child classes
avg. depth of classes c inheritance tree (DIT)	$\frac{\sum_{c\ in\ classes} depth\ in\ tree(c)}{\lvert classes\rvert}$	the deeper a class in the inheritance tree, the more methods are inherited
avg. number of children of class c (NOC)	$\frac{\sum_{c\ in\ classes} \lvert children(c)\rvert}{\lvert classes\rvert}$	the more children cc in c, the higher the likelihood of improper abstraction
avg. coupling of c with other classes c' (CBO)	$\frac{\sum_{c\ in\ classes} \lvert\{c': c'\ connected\ to\ c\}\rvert}{\lvert classes\rvert}$	a higher degree of direct coupling prevents object reuse (detrimental to modular design!)
avg. response for a class (RFC)	$\frac{\sum_{c\ in\ classes} \lvert\{c': c'\ activated\ by\ c\}\rvert}{\lvert classes\rvert}$	the higher the number of potentially activated methods, the higher the adaptation efforts
avg. lack of cohesion in methods (LCOM)	$\frac{\sum_{(m1,m2)\ in\ c} \frac{\lvert attr(m1) \cap attr(m2)\rvert}{\lvert attr(m1\ ) \cup attr(m2)\rvert}}{\lvert\{(m1, m2)\ in\ c\}\rvert}$	low cohesion indicates that a class c could be split. In this context, attr(mi) denotes the attributes referred to by methods mi

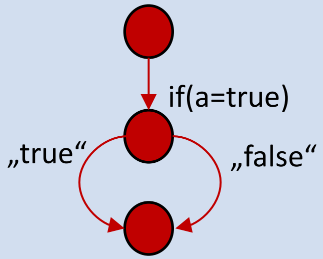


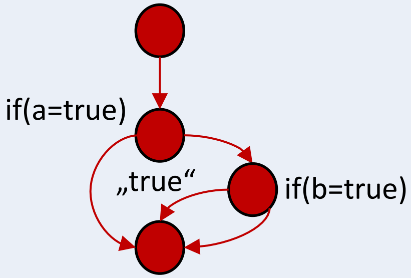
Complexity Metrics

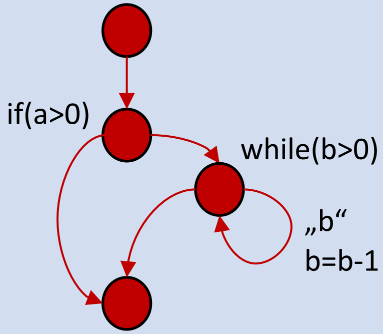
McCabe Structural Complexity



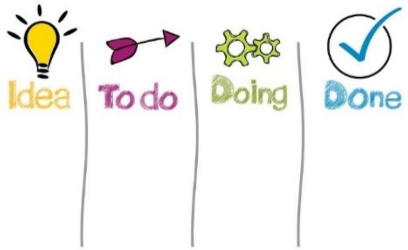
Java Code	Graph	CYC

class McCabeTest1 {
public void test1(boolean a) {
if (a=true) {System.out.println("true");}
else {System.out.println("false");};}
}
		2

class McCabeTest2 {
public void test1(boolean a, boolean b) {
if (a=true) {
if (b=true)
{System.out.println("true");}}}
}
		3

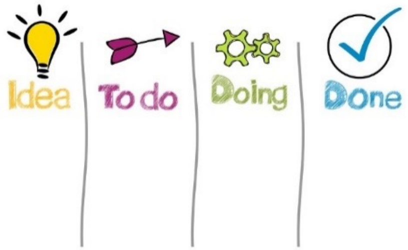
class McCabeTest3 {
public void test1(int a, int b) {
if (a>0) {
while (b>0) {
System.out.println(b);
b = b - 1;}}}
}
		3

- Estimating a code‘s structural complexity
- An example in OOP contexts: method complexity
- McCabe‘s cyclomatic complexity: CYC=D+1
- D: binary decision in flow graph



System Usability Scale (SUS)

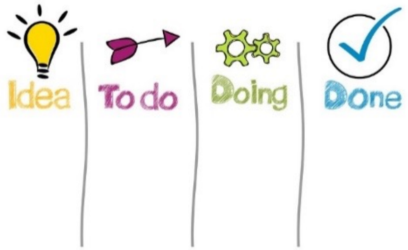
System Usability Scale (SUS)



- SUS is a "quick and dirty" approach to measuring usability (10 items questionnaire) - see: https://bit.ly/2zqvE5x

- Questions:
  1. I think that I would like to use this system frequently.
  2. I found the system unnecessarily complex.
  3. I thought the system was easy to use.
  4. I think that I would need the support of a technical person to be able to use this system.
  5. I found the various functions in this system were well integrated.
  6. I thought there was too much inconsistency in this system.
  7. I would imagine that most people would learn to use this system very quickly.
  8. I found the system very cumbersome to use.
  9. I felt very confident using the system.
  10. I needed to learn a lot of things before I could get going with this system.

SUS Example

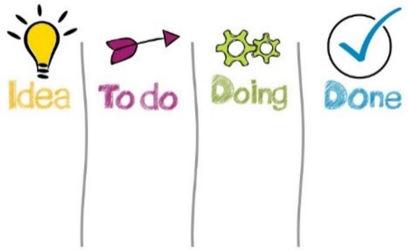


Question	Answer	odd Qs +1, 5-even
Q1	4	3
Q2	1	4
Q3	5	4
Q4	1	4
Q5	4	3
Q6	2	3
Q7	5	4
Q8	1	4
Q9	5	4
Q10	1	4

- SUS scale of answers: 1.0 .. 5.0 (strongly disagree .. strongly agree)
- Subtract 1.0 from each answer to an odd-numbered question
- Subtract the answer to each even-numbered question from 5.0
- Add-up the total score (37.0)
- Multiply the result with 2.5 (92.5)
- Result interpretation:
  - Average of individual respondents
  - >=80.3: people love your UI
  - >=68.0: your UI is ok, but could be improved
  - Otherwise: usability is your priority now!

Further Example Metrics

Verification & Evolution

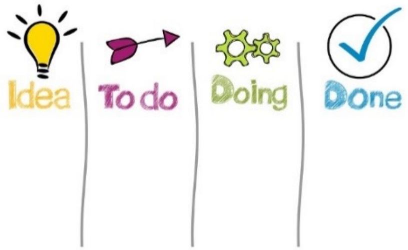


- Quality of test suite: $COV = \frac{tested\ lines\ of\ code}{total\ lines\ of\ code}$

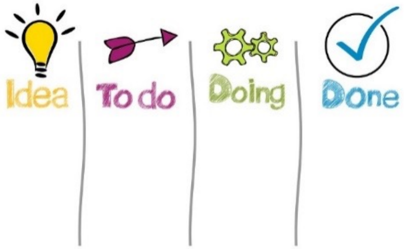
- Mean time to repair: $MTTR = \frac{total\ maintenance\ time}{number\ of\ repairs}$

- Mean time between failures: $MTBF = \frac{total\ operational\ time}{number\ of\ failures}$

Repetition (R2)



- 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 Quality“ and answer the questions.
- Your answers will be taken into account as mentioned in the organization slides.



Thank You!

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

References

- **[GFL2013]** G. Génova, J. Fuentes, and J. Llorens. A Framework to Measure and Improve the Quality of Textual Requirements, Requirements Engineering, 18:25-41, 2013.
- **[CK1994]** S. Chidamber and C. Kemerer. A metrics suite for object oriented design, IEEE Transactions on Software Engineering, 28:476-492, 1994.
- **[MAR2000]** R. Martin. Design Principles and Design Patterns, pp. 1-34, 2000. Link to paper: https://bit.ly/2yLYufQ
- **[BRO2013]** J. Brooke. SUS – A Retrospective, Journal of Usability Studies, 8(2):29-40, 2013.