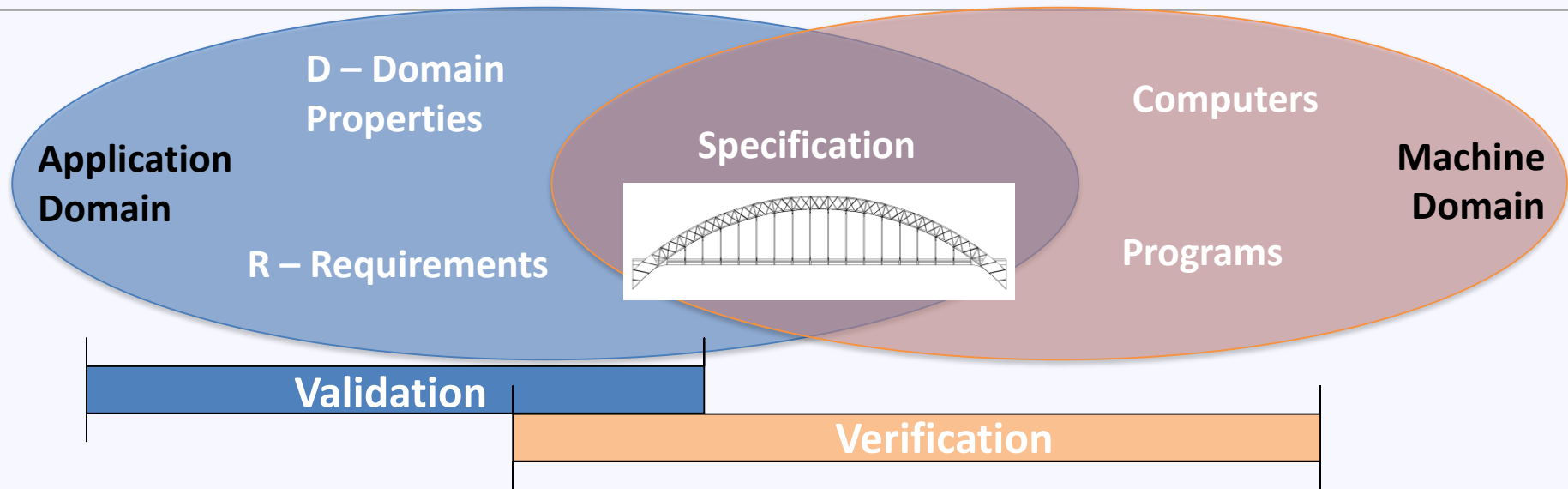


**DIT045/DAT355**  
**Requirements and User Experience**

**Lecture 11: Verification &  
Validation**  
**RE, UX Research Highlights**

Jennifer Horkoff  
[jenho@chalmers.se](mailto:jenho@chalmers.se)

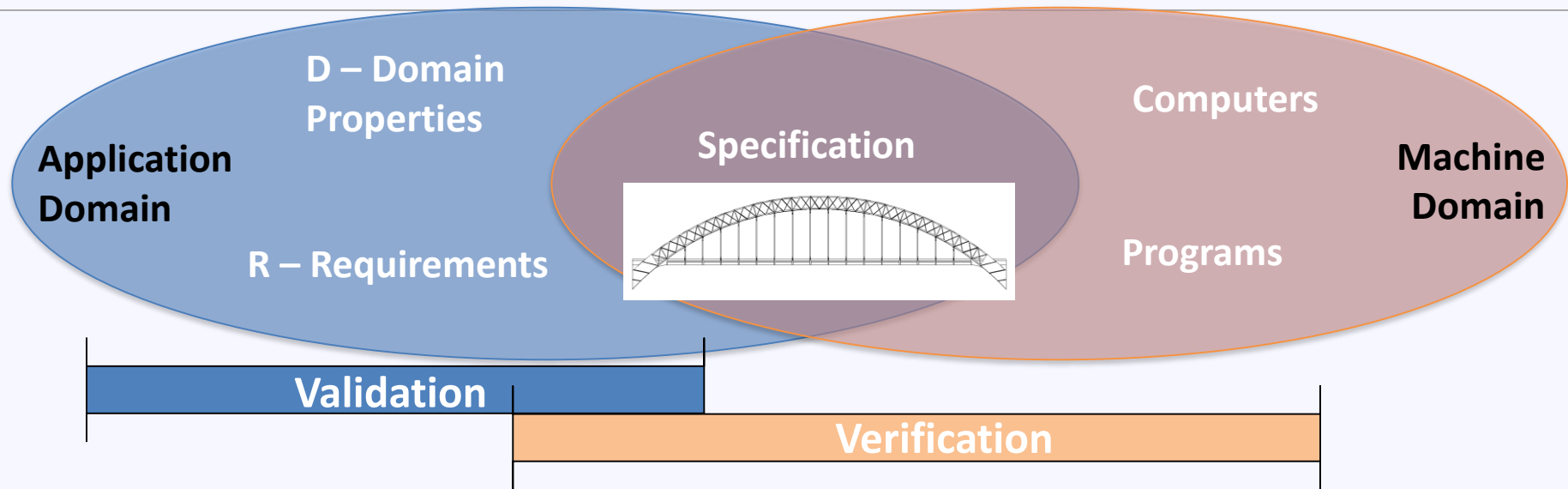
# Validation & Verification



- **Validation:** Are we building the right system?
  - Does our solution solve the real problem?
  - Did we account for the most important stakeholder needs?
- **Verification:** Are we building the system right?
  - Does our design meet the requirements?
  - Does the system do what we say it would do?
  - Are our requirements representations consistent?

(Zave & Jackson, Easterbrook)

# Validation & Verification Criteria



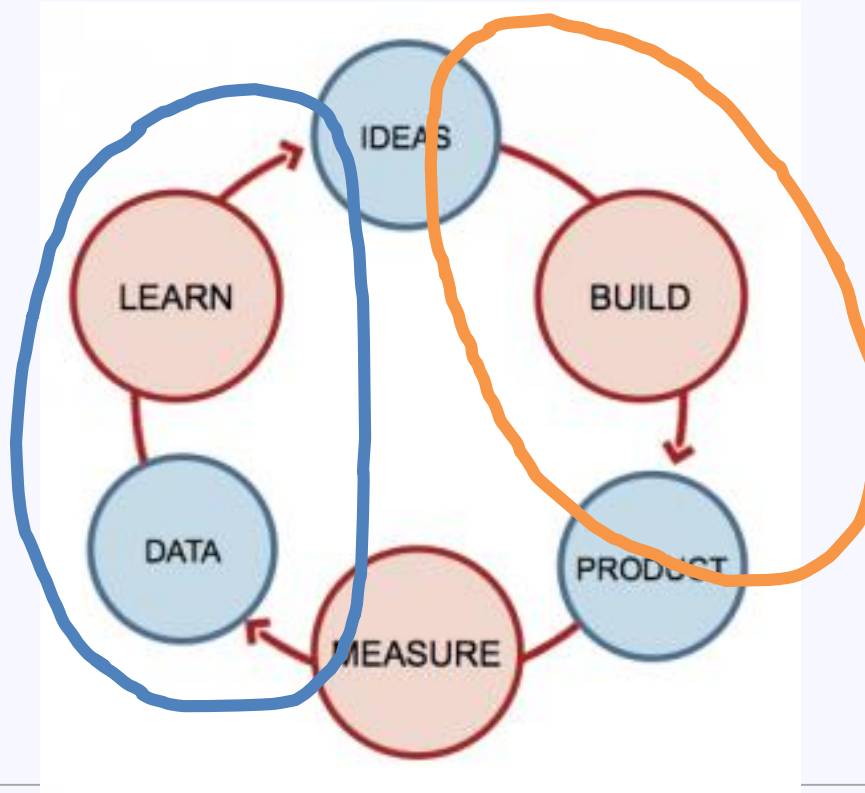
- **Example Validation Criteria**
  - Did we discover all the important/essential requirements (for the MVP, for example)?
  - Did we discover all the important domain properties/assumptions?
- **Example Verification Criteria**
  - The software running on a computer satisfies the specification
  - The specification, given the domain properties, satisfies the requirements

(Zave & Jackson, Easterbrook)

# Feedback Loop

- Recall Build-measure-learn feedback loop from Lean Startup
- How do validation and verification fit in?

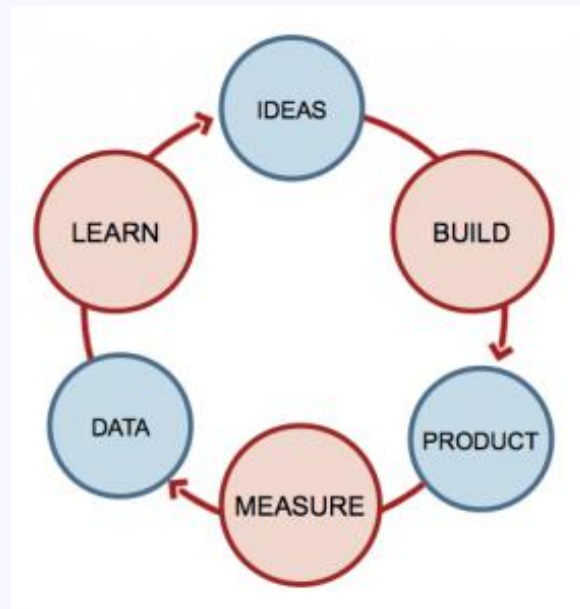
Validation is roughly here?  
Assuming we learn about the market



Verification is roughly here?  
Assuming testing is here

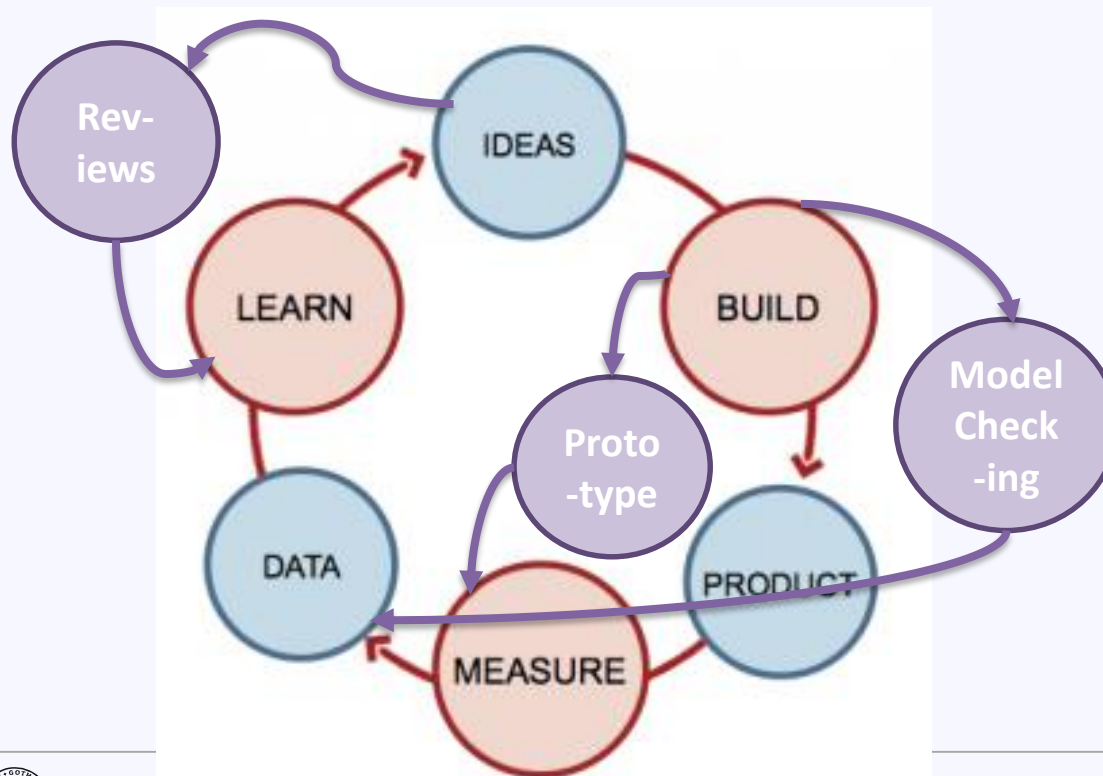
# Feedback Loop

- Challenge: the full loop requires a finished product
- Solution 1: MVP, work in an agile way, have product as soon as possible
- Solution 2: Emulate the solution, test the emulation



# Shortcuts in the inquiry cycle

- Emulate the solution:
  - Prototypes (see past lecture)
  - Model checking
  - Reviews, walkthroughs, inspections



(Easterbrook & Campbell)

# Model Checking

- Evaluating, Validating, and Analyzing the requirements models before listing the requirements or designing the system
- Checking for:
  - Errors like logical inconsistencies
  - Incompleteness
  - Confusions and misunderstandings
  - Satisfied goals
  - Process bottlenecks
  - Etc.
- What you can check for depends on what kind of models you create
  - The more formal and detailed your model, the more detailed your analysis can be → More upfront effort

# Model Checking V&V

- Verification:
  - Is the model well formed?
  - Are parts of each model consistent with each other?
- Validation:
  - Formal model checking
    - Does this property hold?
    - Will the system ever reach this state?
  - What if analysis
    - Reasoning about the consequences of particular requirements choices
  - Simulation
    - Where are the bottlenecks
    - Best for process models (we didn't really cover them, closest was customer journey map)

(Easterbrook & Campbell)



# Model Well-formedness

- Does the model follow the rules for this model
  - If not, the model is not capturing the information intended by the designers of the modeling language
  - Possible that the modelers do not have a common understanding of what they are modeling
  - Someone else would have trouble reading and interpreting the model later
- Examples:
  - Context diagram: system actor in center, all flows are data
  - Use case: system boundary, actors are actually actors, use cases are actions/functions, part of boundary between stakeholders and systems
  - Goal model: correct type of elements used, correct types of links used between elements, correct types of links used inside/outside actors

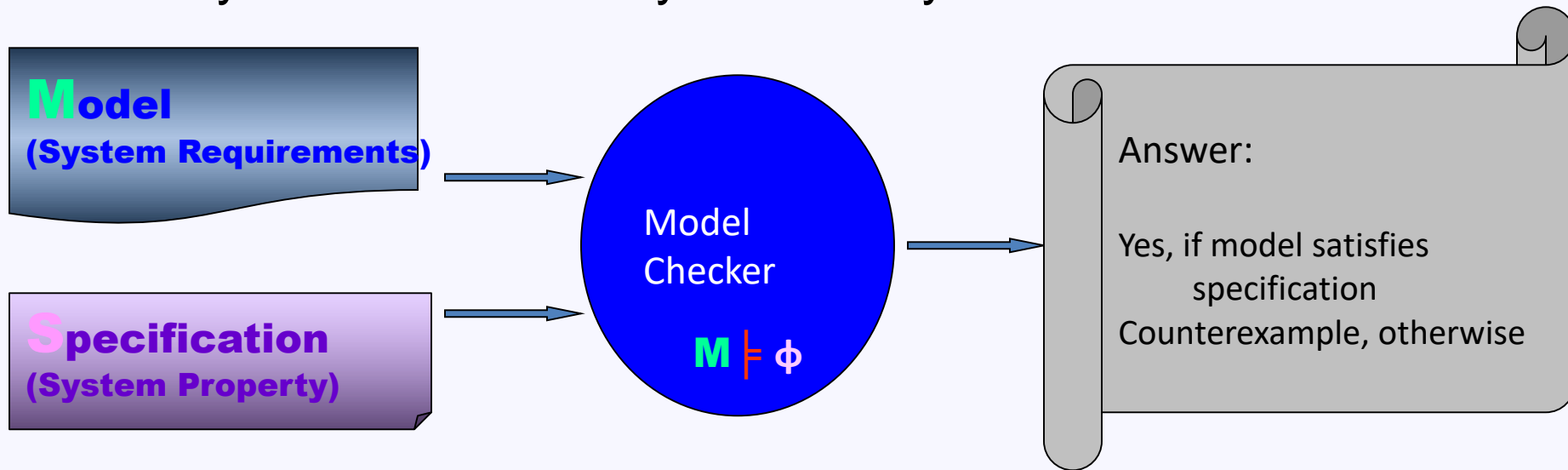
# Recall: Approximate Model Mappings

Context Diagram	Use Cases	Goal Models
System actor	System boundary	System actor
Other actors	Actors	Actors
Inputs/Outputs	Roughly map to use cases	Dependencies
	Use Case	(Usually) Task
		Qualities
		And/Or Refinement
		Contribution

- This is a form of verification for the requirements models
  - Inconsistencies = misunderstandings, incompleteness, etc.

# Formal Model Checking

- We don't do this in this course
- Very useful for safety critical systems

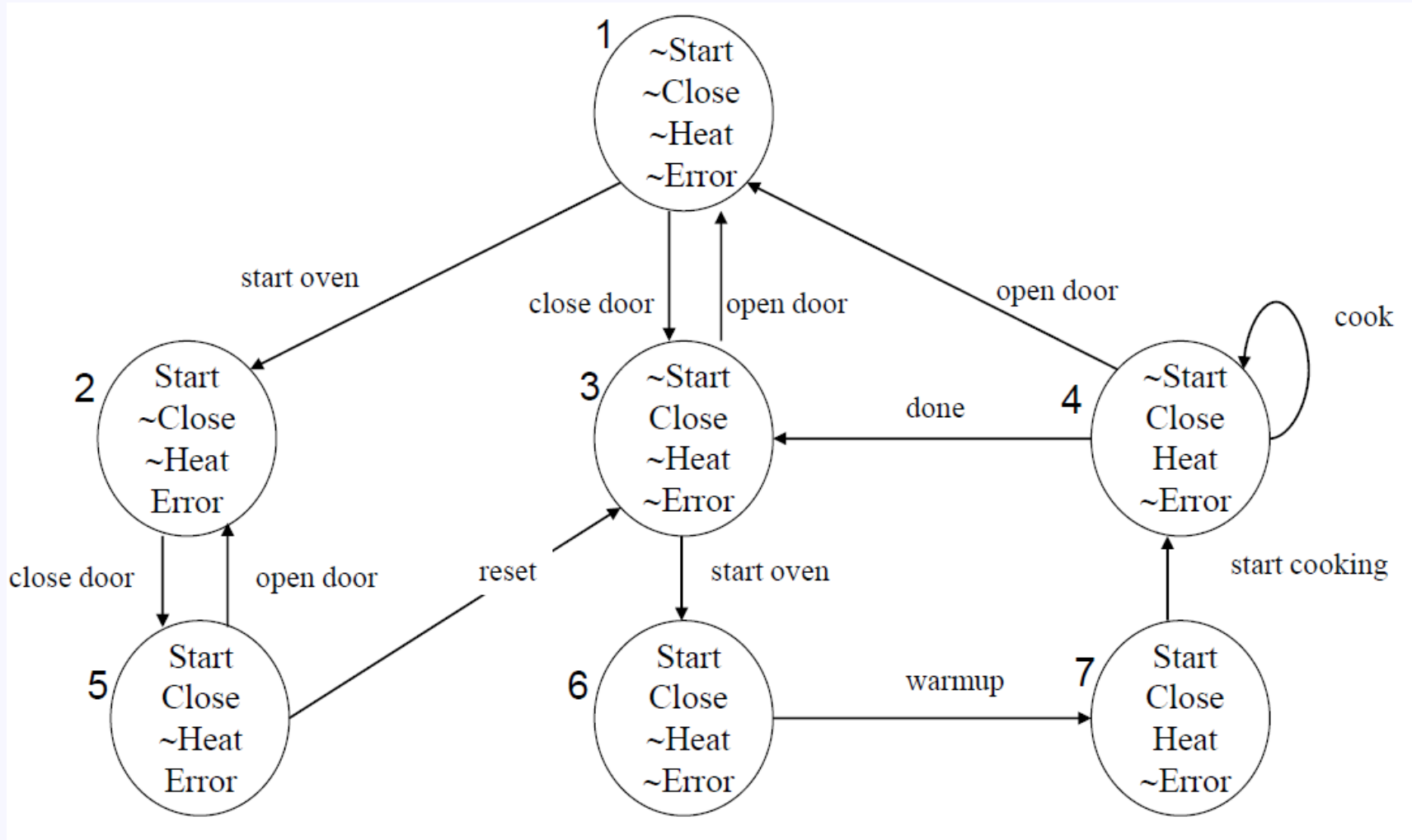


For increasing our confidence in the correctness of the model:

- ☐ Verification: The model satisfies important system properties
- ☐ Debugging: Study counter-examples, pinpoint the source of the error, correct the model, and try again

(Lawrence Chung)

# Model Checking Example



[http://www.dsi.unive.it/~avp/14\\_AVP\\_2013.pdf](http://www.dsi.unive.it/~avp/14_AVP_2013.pdf)

(Edmund M. Clarke, Jr.)

# Model Checking Properties

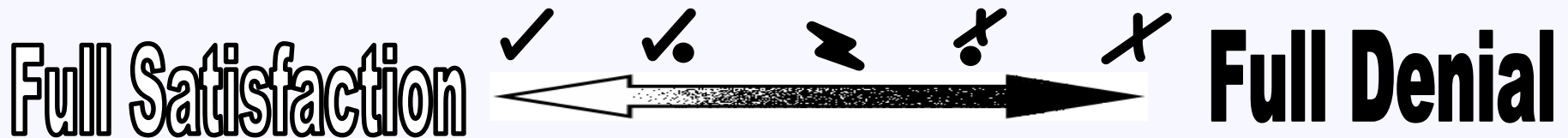
- We would like the microwave to have the following properties (among others):
  - – No heat while door is open
    - **AG**( *Heat*  $\rightarrow$  *Close*):
  - – If oven starts, it will eventually start cooking
    - **AG** ( *Start*  $\rightarrow$  **AF** *Heat* )
  - – It must be possible to correct errors
    - **AG**( *Error*  $\rightarrow$  **AF**  $\neg$  *Error* ):
- Does it? How do we prove it?
  - Answer: model checking

[http://www.dsi.unive.it/~avp/14\\_AVP\\_2013.pdf](http://www.dsi.unive.it/~avp/14_AVP_2013.pdf)

<https://www.cs.cmu.edu/~emc/15414-s14/lecture/ModelChecking.pdf>

# Goal Model Analysis

- Goal models can be assigned formal semantics (meaning)
- Analysis procedures can answer questions, e.g.,
  - What if a particular alternative is selected? How does this effect goals and actors in a model?
  - If I want to satisfy a (set of) goal(s), what alternatives should I select? What tasks should I implement?

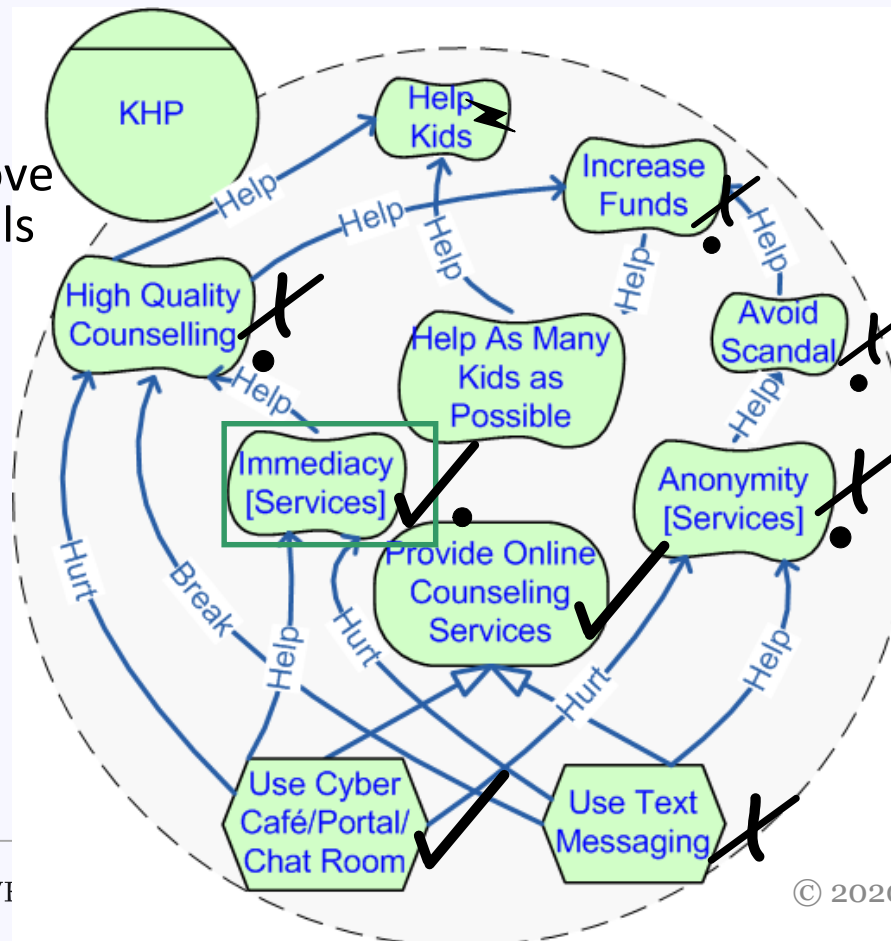


- Tradeoff between information and precision

# Reasoning Example Procedure

- Evaluation based on an analysis question:
  - If the Organization implements Chat Room, but not Text Messaging, what effect will this have on goals?
- Place Initial Labels reflecting Analysis Question

- Propagate labels
- Resolve labels
- Iterate on the above steps until all labels have been propagated
- Analyze result



## Human Intervention

**Immediacy** Receives the following Labels:

Partially satisfied from  
**Chat Room** ✓

Partially satisfied from  
**Text Messaging** ✓

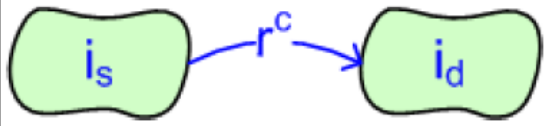
Select Label...

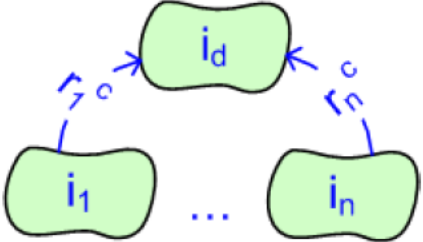
Select partially satisfied





# Model Checking Behind the Scenes

		$S$	$c = m : S(i_s) \rightarrow S(i_d)$	$c = b : S(i_s) \rightarrow D(i_d)$
			$c = hlp : S(i_s) \rightarrow PS(i_d)$	$c = hrt : S(i_s) \rightarrow PD(i_d)$
		$PS$	$c = m, hlp : PS(i_s) \rightarrow PS(i_d)$	$c = b, hrt : PS(i_s) \rightarrow PD(i_d)$
$V(i_s)$	$V(i_s) \rightarrow V(i_d)$	$PD$	$c = m, hlp : PD(i_s) \rightarrow PD(i_d)$	$c = b, hrt : PD(i_s) \rightarrow PS(i_d)$
$U$	$c = any : U(i_s) \rightarrow U(i_d)$	$D$	$c = m : D(i_s) \rightarrow D(i_d)$	$c = b, hrt : D(i_s) \rightarrow PS(i_d)$
$C$	$c = any : C(i_s) \rightarrow C(i_d)$		$c = hlp : D(i_s) \rightarrow PD(i_d)$	
		$v \in V$	$c = u : v(i_s) \rightarrow U(i_d)$	

		$S, PS$	$PS(i_d) \rightarrow (\text{for } r_j^c \in \text{Pos}, \bigvee_{j=1}^n PS(i_j) \vee \text{for } r_j^c \in \text{Neg}, \bigvee_{j=1}^n PD(i_j))$	
		$C$	$C(i_d) \rightarrow \left( \bigvee_{j=1}^n C(i_j) \vee (\text{for } r_j^c \in \text{Pos}, \bigvee_{j=1}^n PS(i_j) \wedge \text{for } r_j^c \in \text{Neg}, \bigvee_{j=1}^n PS(i_j)) \right.$	
			$\left. \vee (\text{for } r_j^c \in \text{Pos}, \bigvee_{j=1}^n PD(i_j) \wedge \text{for } r_j^c \in \text{Neg}, \bigvee_{j=1}^n PD(i_j)) \right)$	
$V(i_d)$	$V(i_d) \rightarrow V(i_1) \dots V(i_n)$	$D, PD$	$PD(i_d) \rightarrow (\text{for } r_j^c \in \text{Pos}, \bigvee_{j=1}^n PD(i_j) \vee \text{for } r_j^c \in \text{Neg}, \bigvee_{j=1}^n PS(i_j))$	
$U$	$U(i_d) \rightarrow \bigvee_{j=1}^n U(i_j)$			

# Reviews, Walkthroughs & Inspections

- Walkthroughs
  - developer technique (usually informal)
  - used by development teams to improve quality of product
  - focus is on finding defects
- Inspections
  - a process management tool (always formal)
  - used to improve quality of the development process
  - collect defect data to analyze the quality of the process
  - written output is important
  - major role in training junior staff and transferring expertise

(Easterbrook & Campbell)

# Structuring the Inspection

- Checklist
  - uses a checklist of questions/issues
  - review structured by issue on the list
- Walkthrough
  - one person presents the product step-by-step
  - review is structured by the product
- Round Robin
  - each reviewer in turn gets to raise an issue
  - review is structured by the review team
- Speed Review
  - each reviewer gets 3 minutes to review a chunk, then passes to the next person
  - good for assessing comprehensibility!

(Easterbrook & Campbell)

# Benefits of Formal Inspections (for code)

- Formal inspection works well for programming:
  - For applications programming:
    - more effective than testing
    - most reviewed programs run correctly first time
    - compare: 10-50 attempts for test/debug approach
  - Data from large projects (note: data is old now)
    - error reduction by a factor of 5; (10 in some reported cases)
    - improvement in productivity: 14% to 25%
    - percentage of errors found by inspection: 58% to 82%
    - cost reduction of 50%-80% for V&V (even including cost of inspection)
- Effects on staff competence:
  - increased morale, reduced turnover (Easterbrook & Campbell)
  - better estimation and scheduling
  - better management recognition of staff ability
- These benefits can also apply to requirements inspections

# Why Use Inspections

- Inspections are very effective
  - Code inspections are better than testing for finding defects
  - For Specifications, inspection is all we have (you can't "test" a spec!)
- Key ideas:
  - Preparation: reviewers inspect individually first
  - Collection meeting: reviewers meet to merge their defect lists
  - Log each defect, but don't spend time trying to fix it
  - Reviewers learn from one another when they compare their lists
  - Defect profiles from inspection are important for process improvement
- Wide choice of inspection techniques:
  - What roles to use in the meeting?
  - How to structure the meeting?
  - What kind of checklist to use?

(Easterbrook & Campbell)

# Experiments on Requirements Inspections

Porter, Adam A., Lawrence G. Votta, and Victor R. Basili. "Comparing detection methods for software requirements inspections: A replicated experiment." *IEEE Transactions on software Engineering* 21.6 (1995): 563-575.

- “Software requirements specifications (SRS) are often validated manually.
- Usually, reviewers use Ad Hoc or Checklist methods to uncover faults.
- These methods force all reviewers to rely on nonsystematic techniques to search for a wide variety of faults.
- We hypothesize that a Scenario-based method, in which each reviewer uses different, systematic techniques to search for different, specific classes of faults, will have a significantly higher success rate.

# Experiments on Requirements Inspections

---

- Forty eight graduate students in computer science participated in the experiment.
- They were assembled into sixteen, three-person teams.
- Each team inspected two SRS using some combination of Ad Hoc, Checklist or Scenario methods.
- For each inspection we performed four measurements:
  - (1) individual fault detection rate,
  - (2) team fault detection rate,
  - ...

# Experiments on Requirements Inspections

- The experimental results are that
  - (1) the Scenario method had a higher fault detection rate than either Ad Hoc or Checklist methods,
  - (2) Scenario reviewers were more effective at detecting the faults their scenarios are designed to uncover, and were no less effective at detecting other faults than both Ad Hoc or Checklist reviewers,
  - (3) Checklist reviewers were no more effective than Ad Hoc reviewers...”



# And then...

- Porter, Adam, and Lawrence Votta. "Comparing detection methods for software requirements inspections: A replication using professional subjects." *Empirical software engineering* 3.4 (1998): 355-379.
- “In previous work we evaluated this hypothesis using 48 graduate students in computer science as subjects. We now have replicated this experiment using 18 professional developers from Lucent Technologies...”
  - Found similar results

# And then...

- Sandahl, Kristian, et al. "An extended replication of an experiment for assessing methods for software requirements inspections." *Empirical Software Engineering* 3.4 (1998): 327-354.
- “We have performed an extended replication of the Porter-Votta-Basili experiment comparing the Scenario method and the Checklist method for inspecting requirements specifications using identical instruments...”
- “We found the requirements specification inspected and not the detection method to be the most probable explanation for the variance in defect detection rate.
- This suggests that it is important to gather knowledge of how a requirements specification can convey an understandable view of the product and to adapt inspection methods accordingly.
- Contrary to the original experiment, we can not significantly support the superiority of the Scenario method. ”

# Inspections/Walkthroughs & Agile

- Requirements inspections/walkthroughs are not so common now?
- Sure, but...
- Agile Backlog Refinement/Grooming
- (<https://www.agilealliance.org/glossary/backlog-grooming/>)
- Recall:
  - In agile, user stories are stored in a backlog (list of user stories)
  - There is a sprint backlog, and a product backlog

# Agile Backlog Grooming/Refinement

- “Backlog refinement (formerly known as backlog grooming) is when the product owner and some, or all, of the rest of the team review items on the backlog to ensure the backlog contains the appropriate items, that they are prioritized, and that the items at the top of the backlog are ready for delivery.
- Some of the activities that occur during this refinement of the backlog include:
  - removing user stories that no longer appear relevant
  - creating new user stories in response to newly discovered needs
  - re-assessing the relative priority of stories
  - assigning estimates to stories which have yet to receive one
  - correcting estimates in light of newly discovered information
  - splitting user stories which are high priority but too coarse grained to fit in an upcoming iteration”

<https://www.agilealliance.org/glossary/backlog-grooming/>

# Backlog Grooming/Refinement Benefits

- “The intent of backlog refinement is to ensure that the backlog remains populated with items that are relevant, detailed and estimated to a degree appropriate with their priority, and in keeping with current understanding of the project or product and its objectives.
- Unlike a more formal “requirements document” the backlog is understood as a dynamic body of information.
  - For instance, not all user stories need to have been broken down to a fine-grained level at the onset of the project, or given detailed estimates; but it is important that at any moment a “sufficient” number of stories should be ready for scheduling in the next few iterations.
- An Agile project is, no less than any other, subject to “scope creep”, in the form of user stories which do not really yield substantial value but were thought “good ideas at the time”, and entered into the backlog lest they be forgotten.”

# Inspections vs. Grooming

Inspections	Grooming
Find requirements defects	Break down user stories
Train staff	Keeping with current understanding
Better morale/retentions	? (probably)
Better estimations	Make estimates
? (probably)	Reduce scope creep
Prepare for design/implementation	Prepare for next sprint
?	Prioritization
Done once per section?	Continuous

# Summary

---

- Validation checks you are solving the right problem
  - Prototyping - gets customer feedback early
  - Inspection - domain experts read the spec carefully
  - Formal Analysis - mathematical analysis of your models
  - ...plus meetings & regular communication with stakeholders
- Verification checks your engineering steps are sound
  - Consistency checking - do the models agree with one another?
  - Traceability - do the design/code/test cases reflect the requirements?

(Easterbrook & Campbell)

# RE, UX Research Highlights



# Course Content

---

- Most of the time we teach you things that are relatively well-established
  - Much work in literature
  - Application in practice
  - Established enough to be in a textbook.
- Sometimes we teach you about more cutting-edge research
- What sort of research/open questions are there in RE and UX?

# RE Research Sample

- <https://re20.org/index.php/accepted-papers/>
- <https://refsq.org/2020/2020/conference-programme/accepted-papers>
- <https://link.springer.com/search?query=&search-within=Journal&facet-journal-id=766>

# UX Research

---

- <https://programs.sigchi.org/chi/2020/program>
- <https://dl.acm.org/citation.cfm?id=3300063&picked=prox>



### Knowledge, People, and Organizations

- How can we find the right knowledge at the right times?
  - in large-scale distributed organizations?
- How to validate that the software is aligned with the needs of the users?

### Adaptation & Change Management

- How can the system adapt to changes @runtime?
- How can we understand the impact of changes?
  - Implemented changes? Changing user needs?

### Business Intelligence & Data

- How to understand whether requirements are being met at runtime?
- How to engineer and manage data requirements?
- How to engineer and manage requirements for autonomous and adaptive systems?

### Continuous Development & Deployment

- What to implement next?
- How can we use requirements data to automate continuous testing activities?
- How can we continuously manage knowledge?

### Security & Privacy

- How to assess security and privacy threats?
- How to support compliance with laws and regulations?

### System Quality & Innovation

- How can we support creativity and innovation, giving organizations a competitive advantage?
- How to elicit, express and validate quality requirements?
- How to focus on the right features at the right time?

Using our approaches  
below

## Requirements Engineering

Supports industry to answer  
the questions above

### Continuous Feedback & Adaptation



- Analyze requirements (quality, consistency)
- Facilitate Data-driven requirements elicitation (experimentation and observation)
- Monitor requirements and adapt system to changing requirements

### Traceability Management

- Guidelines for traceability maintenance
- Challenges of traceability in a collaborative environment
- Method to map information flow from requirements to tests (to identify gaps & waste)
- Flexible and extendable open source tool



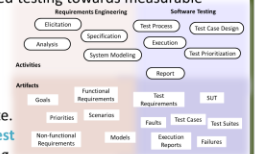
### Requirements Modeling & Analysis

- Analysis of socio-technical models to select amongst **alternative requirements**
- Map requirements to **runtime** business data
- Executable** functional requirements for simulation, V&V



### Aligning Requirements and Testing

- Meta-heuristics** for automated testing towards measurable quality/requirement
- Goal modeling artifacts for **TC prioritization/selection**
- Mapping **information from requirements** to testing to identify oracles, gaps & waste.
- Reusability & evaluation of **test processes** with goal modeling.



### Human Factors & Behavioral SE

- Enhance the creativity and "out of box" thinking of analysts and requirements engineers
- Measure "collective intelligence" and/or creativity of groups of requirements engineers and study correlation with requirements quality
- Measure traits/characteristics of software engineers and their work preferences/strengths
- Support emergent teams through social network analysis

### Quality Requirements

- Representation of **security** properties and verification in software architecture
- Method for **privacy** analysis and compliance with the General Data Protection Regulation (GDPR)
- Specification and automated testing of **performance** requirements
- Initial method for expressing and testing **robustness**



# UX & ID@GU | Chalmers

- <https://www.chalmers.se/en/departments/cse/organization/id/Pages/default.aspx>
- Research groups
  - Applied Robotics in Gothenburg
  - t2i Interaction Laboratory
  - Interaction Design and Children (IDAC)
  - Vehicle Interaction Design (V.IxD)
  - Gameplay Design
  - Visualization and digital representation

# Research Directions in RE & UX?

---

- RE + ML/AI
- ?