# MLMI10
# Designing Intelligent Interactive Systems

# Lecture 7

Per Ola Kristensson

Lent 2022

# Key concepts

- Risk
- Verification and validation
- Cognitive dimensions of notation

# Risk

# Understanding Risk

# Risk

- A system attempts to carry out functions
- In carrying out these functions the system is expected to meet several requirements, such as performance, comfort, safety and cost requirements
- There is a probability of undesirable behaviour due to a system failing to perform as expected, which is:

  **Risk = Likelihood × Impact**

# Hazard, exposure and risk

- **Hazard**—the way in which an object or situation may cause harm
- **Exposure**—the extent to which the likely recipient of the harm is exposed to, or can be influenced by, the hazard
- In order for there to be a risk there must be **both** a hazard **and** an exposure to the hazard (without both these at the same time, there is no risk)
- **Risk**—the probability that harm will actually occur
- Risk = hazard **and** exposure

# Risk is pervasive

- Development risks
  - Delays, additional cost, staffing, rework capacity
- Management risks
  - Staffing
- Performance risks
  - Product quality
- Commercial risks
  - Market sentiment/competition, sourcing costs
- External risks
  - Political/legislative/overall sentiment in society
- AI risks
  - Bias, data sourcing, privacy, lack of transparency
- Other risks
  - “Unknown unknowns”

# Managing Risk

# Risk management

1. Risk analysis
   - System mapping
   - Hazard identification
   - Risk estimation
2. Risk evaluation
   - Risk acceptance decisions
3. Risk control
   - Reducing risk to acceptable levels
4. Risk monitoring
   - Continual verification that risks remain at acceptable levels

# Risk management

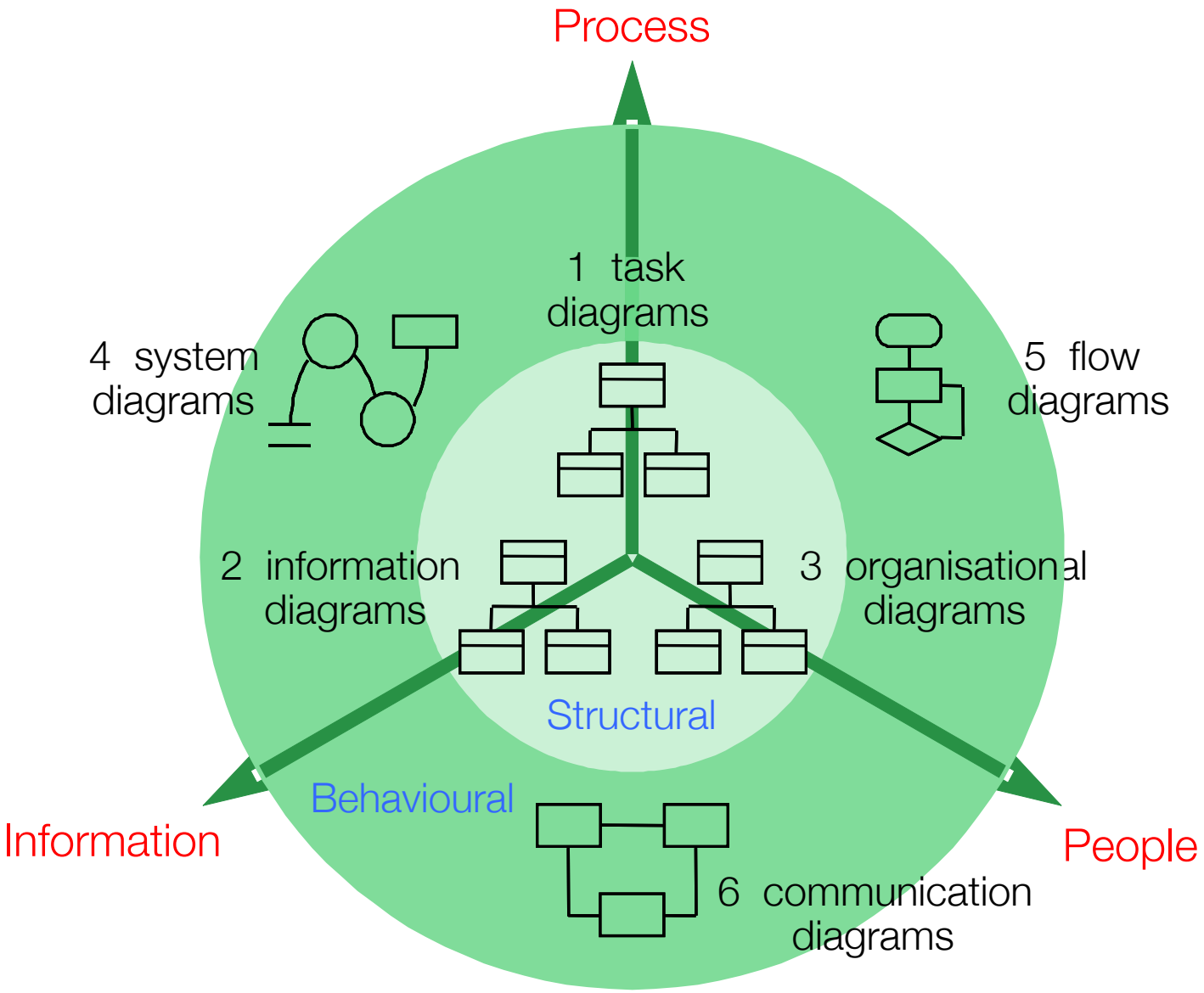
- Identify the system boundary concerning all relevant risks
  - This boundary is often extended beyond the interactive system itself
- Map out the system (system mapping)
  - Use various methods, such as diagrammatic methods, to visualise the entire system within the system boundary
- Identify hazards
- Identify risks
- Identify actions

# System Mapping

# System mapping

- The idea behind system mapping is to describe the system in way that allows effective risk assessment
- One of the most important aspects is setting the **system boundary**
  - This boundary defines the concerns of the system that will be reviewed
  - It is important everything risk related is confined within this system boundary
- System mapping can be carried out in multiple ways and design methods which we have seen before, such as FAST-diagrams, function structures, and outputs of a task analysis can assist the process
- **Handout on Moodle available**

# System mapping

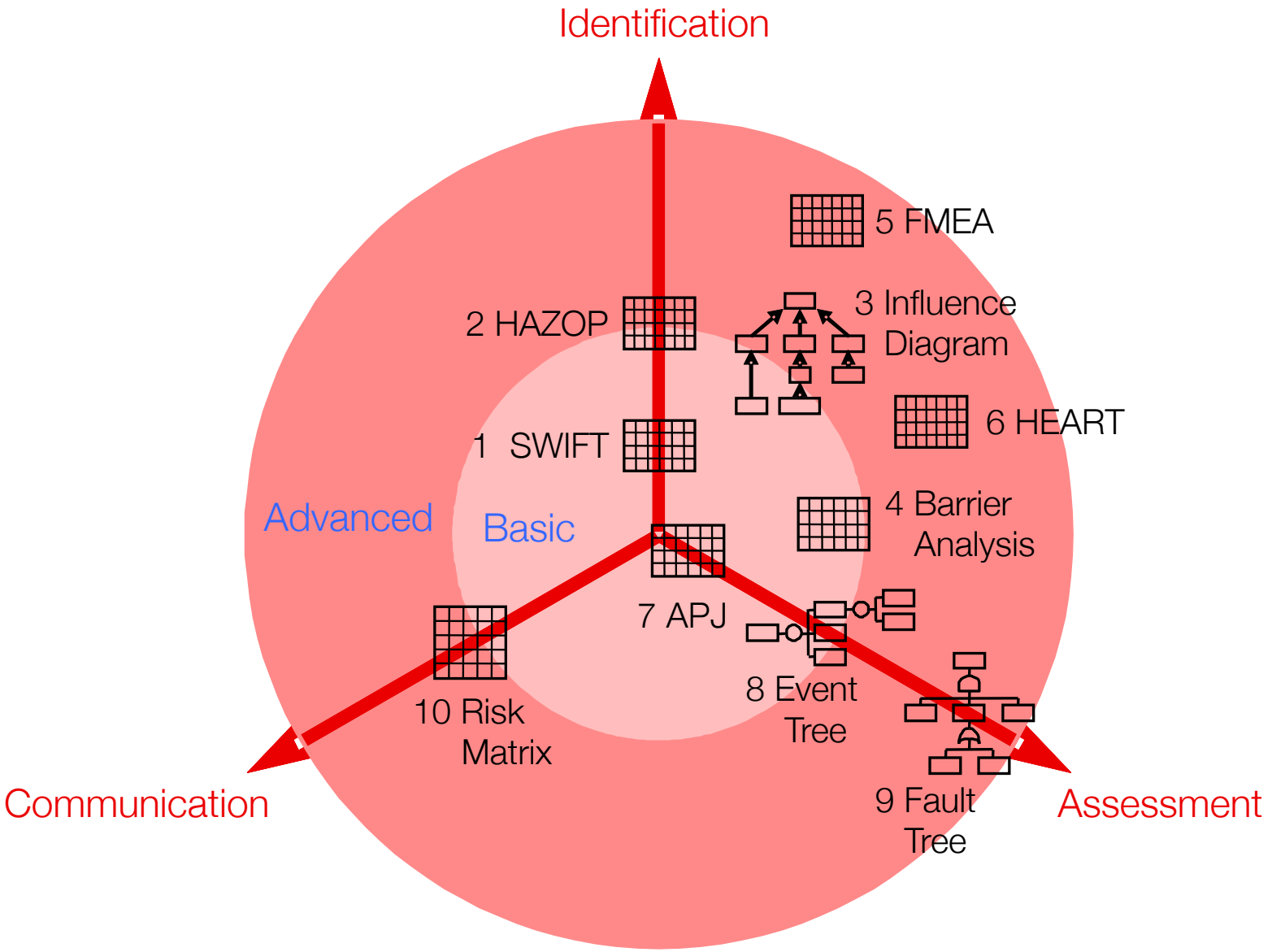


# Risk Assessment

# Risk assessment

- Risk assessment describes three activities:
  1. Risk identification
  2. Risk analysis
  3. Risk evaluation
- Various risk assessment methods have been developed for specific purposes
  - For example, focus may be on identifying hazards, identifying risks, prioritizing risks, communicating risk, etc.
- **Handout on Moodle available**

# Risk assessment



# Structured what-if technique (SWIFT)

- SWIFT is a structured team-based study that uses “what-if” questions to help a team think about and identify relevant hazards and risks
  - It focuses on deviations from normal operations and the impact they may have on a system, procedure or organisation
- SWIFT facilitates discussions to help the team explore differing scenarios, their consequences, causes and impacts
  - From these discussions hazards, risks and controls are identified and summarised
- SWIFT enables hazards and risks to be ranked, based on the perceived severity of the risk and suitability of the existing risk controls, to help identify relevant actions
  - It is most effective where a good description of the system is available
- Uses guidewords to prompt assessment, such as:
  - **wrong** *person, location, thing, time, understanding, process, …*
  - **failure** of *equipment, algorithm, controls, …*

# Structured what-if technique (SWIFT)

| id. | What-if questions | Hazards and risks | Relevant controls | Risk ranking | Action notes |
|---|---|---|---|---|---|
| X | how much? | | | | |
| A | | | | | |
| B | | | | | |
| C | | | | | |
| D | | | | | |
| E | | | | | |
| F | | | | | |
| G | | | | | |
| H | | | | | |
| I | | | | | |
| J | | | | | |

# Hazard and operability analysis (HAZOP)

- HAZOP is a team-based hazard identification method which provides a systematic and structured analysis of a system, focusing not only on hazards, but also on operability issues
  - It provides a qualitative assessment of the presence of hazards, their potential consequence and appropriate actions
- HAZOP focuses on deviation from the intended performance of the system
  - It can reveal shortcomings in the overall activity, the design of its component parts, proposed methods of operation, or interactions between these
- HAZOP is used where a good description of the system is available and draws on the expertise and understanding of a group of people who are experienced either in the specific or similar systems

# Hazard and operability analysis (HAZOP)

| id. | Prompt words | Deviation | Causes | Conse-quences | Existing barriers | Actions |
|---|---|---|---|---|---|---|
| X | less than | | | | | |
| A | | | | | | |
| B | | | | | | |
| C | | | | | | |
| D | | | | | | |
| E | | | | | | |
| F | | | | | | |
| G | | | | | | |
| H | | | | | | |
| I | | | | | | |
| J | | | | | | |

# Failure mode and effects analysis (FMEA)

- FMEA is a team-based technique that can be used to consider and identify the effects of human error on systems
  - It is a flexible approach that can be used to consider individual operator failure and/or team failures
- FMEA identifies the likelihood of the failure and severity of consequence for each failure mode identified by the team
  - The combined influence of severity and likelihood is then used to rank failure modes and prioritise corrective actions
- FMEA requires a team containing a range of experienced individuals who can offer a variety of different perspectives on a given system/process
  - It is most effective where a good description of the system is available

# Failure mode and effects analysis (FMEA)

| id. | Task steps | Failure modes | Causes | Likelihood/ severity | Recovery steps | Actions |
|---|---|---|---|---|---|---|
| X | Infusion | | | | | |
| A | | | | | | |
| B | | | | | | |
| C | | | | | | |
| D | | | | | | |
| E | | | | | | |
| F | | | | | | |
| G | | | | | | |
| H | | | | | | |
| I | | | | | | |
| J | | | | | | |

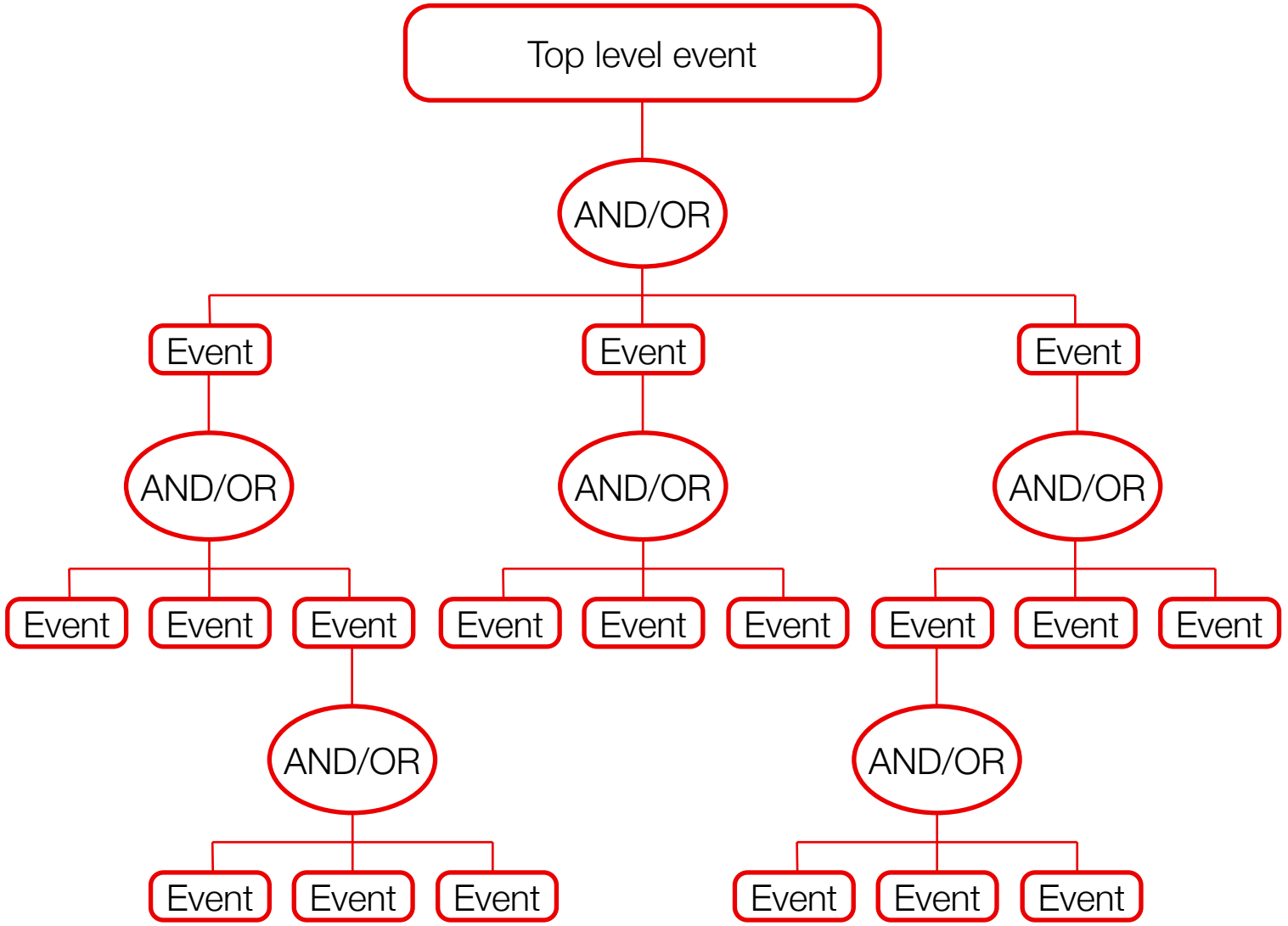
# Failure mode and effects analysis (FMEA)

- A failure modes and effects analysis (FMEA) assesses the consequence of failure of a single component in a system
- Failure may be total or partial and may affect more than one physical property
- Failure modes and effects analysis:
  - was developed for the automotive industry
  - assesses effects of single component failures
  - may be used to assess product or process designs
  - are used as a check on the adequacy of safety systems
  - are normally qualitative but can be qualitative (FMECA)
  - may be done by a team or individual

# Fault tree

- A fault tree is a graphical device for identifying and analysing factors that contribute to the occurrence of an adverse or undesired event
  - They are tree-like diagrams that pictorially represents such factors and their logical relationship to the adverse/undesired event
- Fault trees provide insight into the relevant causes of failure and highlight interrelationships between system components and potential weaknesses in system reliability
  - They can also be used to analyse the causes of human error, their impact on system reliability and when quantified help determine system risk
- Fault trees can be used either to quantitatively assess the likelihood of an undesirable event occurring or to qualitatively assess causes of failures

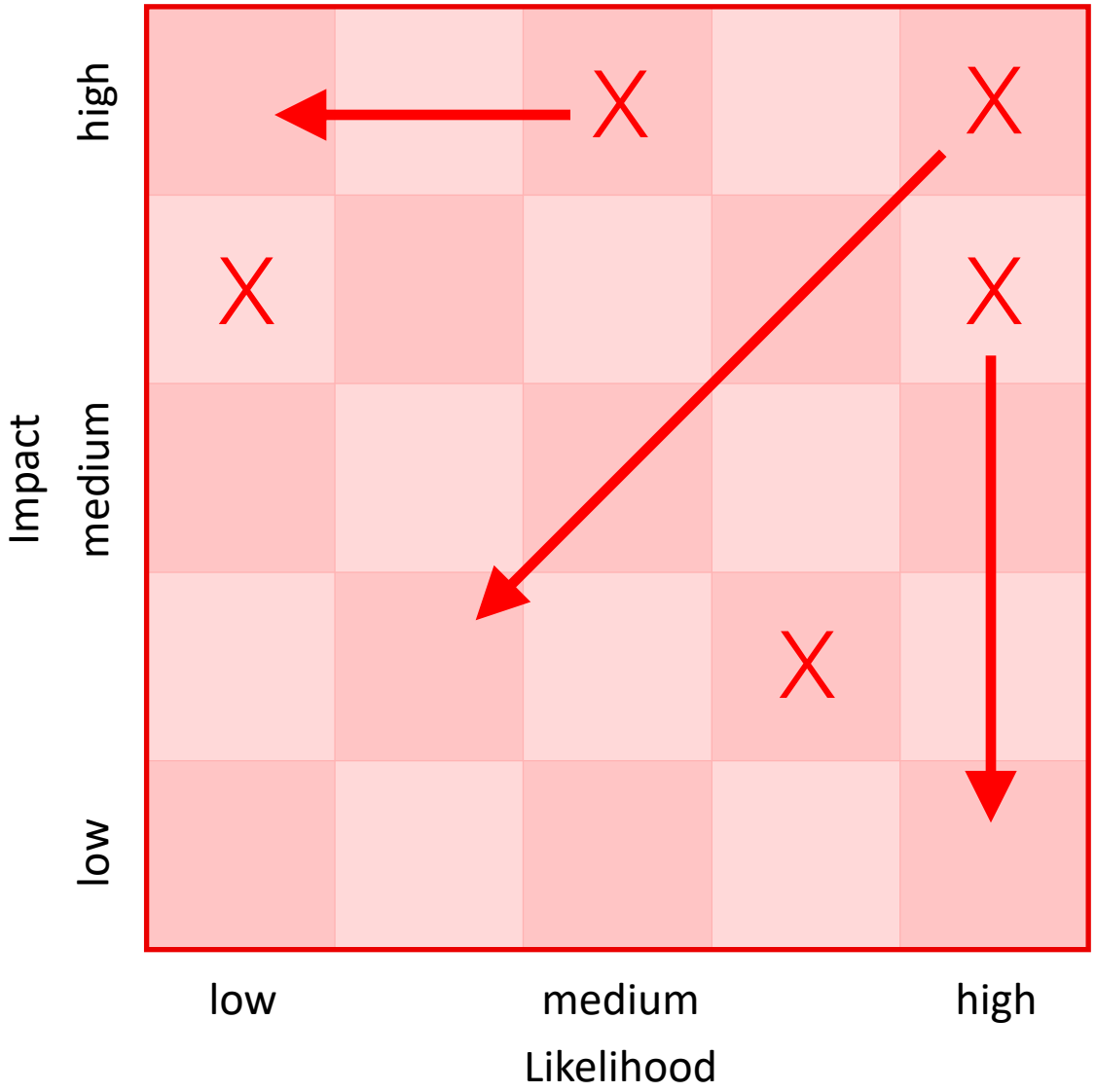
# Fault tree


Top level event
AND/OR
Event
Event
Event
AND/OR
AND/OR
AND/OR
Event
Event
Event
Event
Event
Event
Event
Event
Event
AND/OR
AND/OR
Event
Event
Event
Event
Event
Event

# Risk matrix

- Risk matrices allow individuals to rank the consequence and likelihood of an undesired event, act or activity to provide a risk rating
- They can be used in conjunction with a range of methods, such as HAZOP and FMEA, to rank and prioritise risk.
- Risk matrices are made up of two axes: likelihood and impact, often using a scoring system from 1 (low) to 5 (high)
- Impact can take many forms including injury to persons, property and/or profit
- Risk matrices may also be partitioned to into regions of acceptable and unacceptable risk
- In general entries in the upper right of the matrix are not desirable, while those in the lower are likely to be acceptable

# Risk matrix



# Verification and Validation

# Verification

- **Verification** is ensuring a system complies with the requirements specification
- This is why it is important requirements are testable and (ideally) quantifiable
- Precise verification strategies vary depending on the type of system that is being developed

# Validation

- **Validation** is ensuring the system is fulfilling actual needs
- It is a step beyond verification, which merely ensures the system complies with the requirements specification
- Validation strategies also vary depending on the type of system that is being developed

# Verification vs. validation

- Contrast:
  - **Verification:** Does my product meet the requirements set out in my requirements specification?
  - **Validation:** Am I designing the right product for operational needs?
- It is possible for a product to succeed in verification but fail in validation
  - For instance, if the product is correctly designed according to the requirements specification, but the requirements specification has failed to capture the true operational needs

# Verification vs. validation

- Verification:
  - “Have we built the thing right?”
- Validation:
  - “Have we built the right thing?”

# Verification

# Verification cross-reference matrix (VCRM)

- Specifies in detail how each requirement will be verified, usually has the following columns
  - Requirement ID
  - Requirement
  - Verification Method (code for verification procedure described elsewhere)
  - Allocation (component, software module, etc.)
  - Success criteria (free-text description clearly explaining what constitutes a success in terms of verification of the requirement)

# The four basic methods of verification

- **Inspection** (sometimes called **Examination**)
  - Nondestructive examination of system using at least one of the five human senses (typically visual, audio and haptic)
  - For example, visually check that every software dialog contains the desired information; visually check a heat fuse is correctly wired
- **Demonstration**
  - Check that manipulation of the system in terms of its intended use does indeed result in expected results
- **Test**
  - Feed predefined inputs, check expected outputs are observed
- **Analysis**
  - Verification using calculations, models and/or testing equipment to predict system characteristics, such as performance, based on for instance a sample set of components or sample of tests

# Verification requirement attributes

- Objective
  - The purpose/goal of this verification
- Method
  - The methods that need to be applied and a specification of the verification circumstances (lab tests, simulation, field test, etc.)
- Environment
  - A specification of the environments and any environmental conditions (wind, background noise, sun, heat, cold, stress level of participant, etc.)
- Special conditions (if applicable)
  - A specification of particular unique conditions, such as system configurations, necessary to carry out verification
- Success criteria
  - Expected results, might be specified within ranges

# Verification requirement attributes

- Objective
  - The purpose/goal of this v
- Method
  - The methods that need to the verification circumstances (lab tests, simulation, field test, etc.)
- **Environment**
  - A specification of the environments and any environmental conditions (wind, background noise, sun, heat, cold, stress level of participant, etc.)
- Special conditions (if applicable)
  - A specification of particular unique conditions, such as system configurations, necessary to carry out verification
- Success criteria
  - Expected results, might be specified within ranges

Failure to accurately representing the environment leads to a **verification-validation trap**

# Typical process

- Inspection planning
- Individual reviewing
- Defect evaluations at review meetings
- Updating of verification cross-reference matrix

# Typical process

- Inspection planning
- **Individual reviewing**
- Defect evaluations at review meetings
- Updating of verification cross-reference matrix

- **Implication: the reviewers (testers) are actually stakeholders**

# Traceability (IEEE standard 1059)

- A key concept in verification is **traceability**
- Traceability means it is possible to link attributes of the complete systems to the requirements via a **trace**
- Traceability is the ability to verify:
  1. Requirements have been carried forward to design specifications (conceptual/embodiment design),
  2. which in turn have been implemented (detailed design),
  3. which in turn have been included in test plans and any test cases, and
  4. which in turn is checked to have resulted in effects of implemented requirements having been made available to the stakeholders / end-users.
- **Traceability analysis** is the task for following each trace for consistency, completeness and correctness to verify points 1-4 above

# Modelling

- As in all engineering, **modelling** can be used to verify some requirements, for instance:
  - Task time modelling, estimating task times by breaking tasks into a series of primitive tasks with either constant times or times governed by formulaic relationships
  - Error modelling by perturbing sensor input (sensitivity analysis)
  - Engineering models

# Code correctness

- Unit testing: automatic test suits that probe the system and compare actual system output against expected system output
- Code review: quality assurance engineers review the code for correctness and compliance
- Static program analysis: analyses the code offline in order to identify program bugs and performance bottle-necks
- Dynamic program analysis: analyses the code online (i.e. when the system is running) in order to identify program bugs and performance bottle-necks
- System verification: formal model analysis

# Expected user interface behaviour

- Unit testing
- Automatic GUI testing: a variant of unit testing in which the computer simulates user behaviour, possibly using a cognitive architecture or a statistical generative model
- Human testing: typically by either carrying out tasks in pre-defined scripts or by specifying set goals
- Usability inspection:
  - Testing every component (function carrier) that has a user concern against a set of guidelines, such as Nielsen's heuristics or Shneiderman's golden rules

# Validation

# Validation

- Problem-dependent
- Can be carried out as part of a design staged in separate phases (milestones)
- In contract-based research validation is often conflated with success criteria for accepting a work package
- In consumer products, the ultimate validation is market success and validation may be carried out to understand why a product succeeds or fails

- A validation addresses the following questions:
  - Did we address the solution-neutral problem statement?
  - Was the solution-neutral problem statement correct?

# Usability inspection

- Heuristic evaluation (widely practiced)
  - Identify a suitable set of heuristics (such as Nielsen's heuristics or Shneiderman's eight golden rules)
  - Inspect all elements of the interface and judging it
  - Typically more than one expert is used in real products
- Cognitive walkthrough (task-specific)
  - Walk through how a user would need to achieve a task and ask these four typical questions:
  - Will the user try to achieve the effect that the subtask has?
  - Will the user notice that the correct action is available?
  - Will the user understand that the wanted subtask can be achieved by the action?
  - Does the user receive appropriate feedback?

# A/B testing

- Compares performance of System A vs. System B (a competitor or baseline system, essentially the control)
- Typically a single independent variable with two levels, such as System A vs. System B
- Dependent variables are task dependent, but common measures include time, accuracy, cognitive workload, user ratings
- Typically statistical significance tests, such as analysis of variance (ANOVA) or non-parametric variants are used to determine if one system performs
- Typically used when **objective user performance** criteria are critical to the product (such as a new mouse or touchpad)
- **High internal validity** (confounding factors are explicitly controlled so it is possible to make causal inferences)
- **Low external validity** (difficult to make a well-controlled experiment model reality)

# *In-situ* evaluation

- A product prototype is *deployed* to a sample of the target audience for some period of time
- Multiple data collection strategies are possible, including field visits (observation), using the Experience Sampling Method (ESM), diary studies, or system logging (or a combination)
- Provides **context**
- **Low internal validity** (ability to isolate confounding factors is low)
- **High external validity** (relevance)

# Evaluation

# Evaluation

- Evaluation is a term that is commonly used and often conflated with some implicitly defined V & V (verification and validation) process
  - However, as we have seen verification is a rigorous and a distinct process separate from validation
- The word **evaluation** in this context is better thought of as some form of evaluation process that indicates success based on ability to answer a research question
  - Is method A faster than B for some standard task?
  - No real attempt to verification beyond ‘validating the instrument’: ensuring external factors are minimised (such as noise factors), the comparison is carried out fairly and is reproducible
- Engineering research papers often carry out an evaluation
- The purpose of an evaluation is usually to verify that the hypotheses that lead to a particular design indeed resulted in the predicted benefits

# Verification and Validation Tactics

# Verification scope

- Allocation in the VCRM is often cross-cutting (affecting more than one component)
- Verification methods may address such components in isolation; however, accurate verification will require addressing such component allocation on a module basis
- Often useful to go back to a function model and think about how every function can be verified

# Sensitivity and validation

- Any model should be examined based on the following two aspects:
- **Sensitivity (sensitivity analysis)**
  - Sensitivity in dependent variables (outputs) when manipulating independent variables (parameters; or inputs)
  - Often useful to think about sensitivity of an output variable (or a set of output variables) as a function of perturbations or parameter choices of both controllable and uncontrollable parameters
- **Validity (model validation)**
  - Predictive ability of model on unseen data that accurate models drawing data from deployment environments

# Avoiding verification-validation traps

- A verification-validation trap is a design process outcome in which a system (or module) passes verification but the complete system design fails in validation due to the verification being incomplete in representing the environment the system is exposed to in the validation phase
- The environment must be accurately captured when specifying the verification requirements attributes
- Acquiring appropriate data is often challenging
- Experience-sampling method (ESM) is one way of acquiring *in-situ* data
- May be necessary to build a ‘passive’ device that merely collects sensor data that is deployed to paid test users for a set amount of time

# Using verification to drive design

- Setting realistic verification targets (possibly coupled to realistic environments) at an early stage and verifying early can help identifying suitable sensor and inference solutions at the conceptual design stage and onwards
- Before finalising  a function or function sequence in the function model it is worth considering “how am I going to verify this function?”

# Verifying non-trivial user concerns

- **Learning curve**
  - Exposure to interface over prolonged amount of time until learning saturates
  - Rate of learning is nonlinear
  - Do you need the learning curve or is it just expert performance? Estimates of expert behaviour can be obtained by saturating learning on subset o actions (for instance, repeated button pressing sequences)
- **Just noticeable difference (JND); Weber fractions**
  - Common requirement for feedback, in particular audio or haptic feedback
  - Typical verification strategy involves a stair-casing strategy of modifying stimulus delta until a reliable difference is detected
  - Efficient well-documented stair-casing strategies exist in the literature
- **Nausea, perceived cognitive overload, etc.**
  - Established survey methods exist in the literature
- **Objective cognitive overloading tests**
  - Interleave primary task with secondary task; measure effect on time and accuracy
- **Ensuring a low false alarm rate**
  - Often delicate balance between a high true positive rate and a sufficiently low false positive rate
  - Cost of false positives can be modelled as, for example, decision time + action time of user; or in terms of consequences to risky behaviour (ignoring signals)

# Cognitive Dimensions of Notation

# Cognitive dimensions of notation

- A **vocabulary** for discussions about the design of software, in particular software which isn't easily evaluated, such as programming languages and programming tools, smart wearable devices, and software where humans solve tasks in collaboration with machine assistance (such as human-in-the-loop systems)
- A design vocabulary  means that design principles and implications are **explicit** and enables engineers to discuss **trade-offs**

- **See handout on Moodle for a detailed example**
- Resource site: http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/
- Book chapter (*Notational systems–the cognitive dimensions of notations framework* ): http://www.cl.cam.ac.uk/~afb21/publications/BlackwellGreen-CDsChapter.pdf

# Notational systems

- Systems with novel graphical elements that are composed according to novel visual grammars (rules)
- Can be simple and complex systems
- Examples:
  - Visual programming languages
  - Central heating controls
  - New pervasive and ubiquitous systems
  - AI-infused user interfaces

# Types of notation use activity (Blackwell and Green 2003)

- **Incrementation:** adding cards to a card file, formulas to a spreadsheet or statements to a program
- **Transcription:** copying book details to an index card; converting a formula into spreadsheet or code terms
- **Modification:** changing the index terms in a library catalogue; changing layout of a spreadsheet; modifying spreadsheet or program for a different problem
- **Exploratory design:** sketching; design of typography, software, etc; other cases where the final outcome cannot be envisaged and has to be 'discovered'
- **Searching:** hunting for a known target, such as where a function is called
- **Exploratory understanding:** discovering a structure or an algorithm, or discovering the basis of classification

Source: Blackwell, A.F. and Green, T.R.G. 2003. Notational systems–the cognitive dimensions of notations framework. In Carroll, J.M. (Ed.) *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science.* San Francisco: Morgan Kaufmann, pp. 103–134.

# Cognitive dimensions of notation (Blackwell and Green 2003)

- **Viscosity:** resistance to change
- **Visibility:** ability to view components easily
- **Premature commitment:** constraints on the order of doing things
- **Hidden dependencies:** important links between entities are not visible
- **Role-expressiveness:** the purpose of an entity is readily inferred
- **Error-proneness:** the notation invites mistakes and the system gives little protection
- **Abstraction:** types and availability of abstraction mechanisms
- **Secondary notation:** extra information in means other than formal syntax
- **Closeness of mapping:** closeness of representation to domain
- **Consistency:** similar semantics are expressed in similar syntactic forms
- **Diffuseness:** verbosity of language
- **Hard mental operations:** high demand on cognitive resources
- **Provisionality:** degree of commitment to actions or marks
- **Progressive evaluation:** work-to-date can be checked at any time

Source: Blackwell, A.F. and Green, T.R.G. 2003. Notational systems–the cognitive dimensions of notations framework. In Carroll, J.M. (Ed.) *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science.* San Francisco: Morgan Kaufmann, pp. 103–134.

# Example

- Consider a smart ring for controlling a heating system:
  - The ring visualises temperature by colour change (heatmap scale)
  - When the user wishes to adjust the temperature the user touches the ring and strokes it counter-clockwise to dial down the temperature and opposite to dial it up
  - The ring colour momentarily changes in response to user adjustment and remains in this state for five seconds at which point the ring colour reverts to its normal behaviour
- This system is **viscous**, has high **consistency** and has low **demand on mental operations**; nevertheless, but it can be argued it has several analytic challenges:
  - The system has poor **visibility** as the temperature adjustment method is not apparent (poor perceived affordance)
  - The **mapping is both close and not close** to the task. On the one hand, the ring motion models a dial on a heating controller although the degree of **role expressiveness** in the design is up for debate. On the other hand, the fact the user is adjusting the temperature **in the future** is not apparent.
  - There are **hidden dependencies** as the fact the user is manipulating the target temperature in a control system is not revealed.