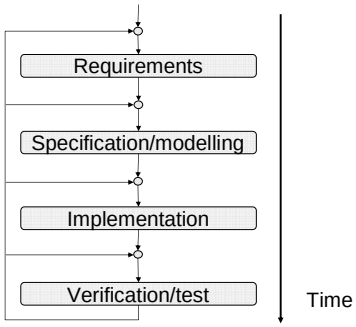
## Modeling

- By modeling we mean (in this context) different forms and methods used for description of a technical artifact
- Modeling of a technical artifact is done:
  - using different description techniques
  - for different purposes (analysis, synthesis, …)
  - from different perspectives
  - of different aspects (structure, behavior, …)
  - at different levels of abstraction



## Design and verification






## Simulation

**Simulation**
a means to imitate and analyze how a system (described by a model of the system) will behave in different situations (described by a model of its environment)

**Analysis requires two models representing**:

- The system under test
- The environment (including actors) of the system

**Often we want to analyse a part of a system**:

- A set of tasks (and related real world processes)
- A load rejection principle
- An algorithm (for scheduling or allocation)
- A resource sharing or communication protocol

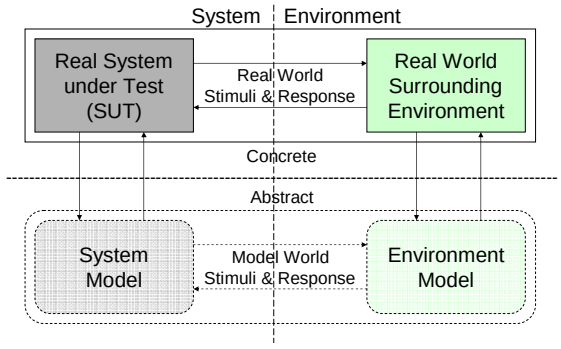


## Simulation cont.

- Simulation is used to test or analyse a systems behaviour in real world scenarios
- This by help of a model of the system and a model of its environment (capturing their behaviours in both time and space)
- Distinguish model and reality
- Distinguish system and environment



## Reality/Model System/Environment

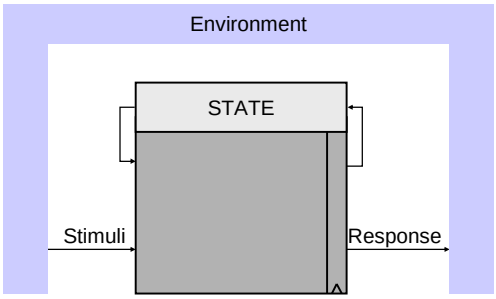





## Simulation cont.

- Systems can be modelled and simulated for different purposes and with varying degrees of detail or abstraction
- To enable simulation of a system we need to create a model of its dynamic behaviour
- That is a model of the systems response to its environmental stimuli and internal state

## Stimuli-state-response model





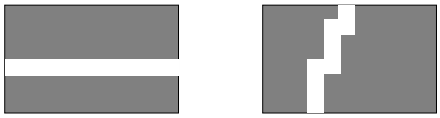

## Simulation cont.

- A model **must not be complete or perfectly accurate**
- But complete and accurate enough to answer our questions
- It should focus on the detail or problem that we want to know more about
- We can often replace one detailed and complex model or simulation with several partial and simpler models
- This makes it important to **be aware of the scope and purpose** of:
  - Each model
  - Stimuli pattern
  - Simulation based analysis



## Horizontal and vertical models

- Horizontal layer of a layered design selected and modeled at a suitable abstraction level
- Vertical slice, cutting through a design, selected to focus on a function or characteristics of a system, modeled at one or several abstraction levels





## Environment model

- The environment can be modelled as a set of pre and post conditions/states and sequences of events to be used as stimuli/response patterns
- Such an environment model should correspond to real world conditions and changes influencing the system to be analysed
- The ordering of stimuli and response events is a temporal ordering (with more or less abstract view of time depending on the modelling needs)
- The temporal relations both between different stimuli and between stimuli and response is important

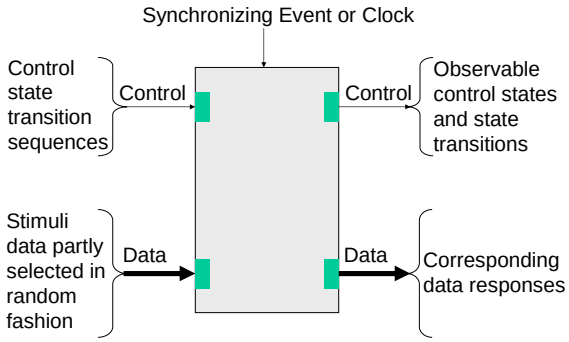


## Environment model, cont.

- The environment can also be modelled and tested using existing models of surrounding systems
- These surrounding systems interact with and generate stimuli to the SUT
- A complete verification of the SUT requires that all important states and state transitions are exercised



## Model of system under test



## Modelling of stimuli patterns

- The model must be set to interesting states using program controlled stimuli sequences
- A control state transition should be analysed for different data combinations (exhaustive test of all combinations is often practically infeasible)
- Data stimuli combinations can sometimes be created by use of random or standard pattern generators



## Pre and Post Conditions

pre($input_{pre}$)

execute(output = program($input_{pre}$))

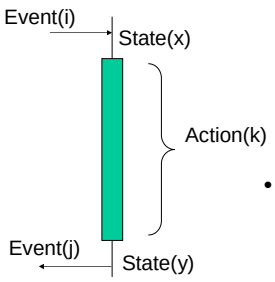
post($output_{post}$)

Read:

if a set of input conditions holds before the execution of a program then the result is assumed to be a set of expected output conditions after its execution



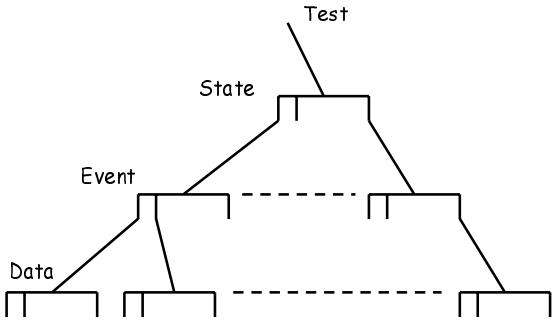
## Test Case




- We should design our system such that we can bring the system to different states (and data pre conditions) without too large effort
- We can see testing as a set of event or message triggered state transitions (resulting in actions, events and data post conditions)

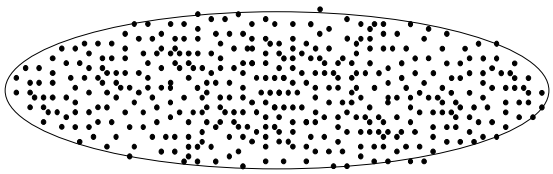


## Test Case: Tree Structure






## Test Coverage Distribution

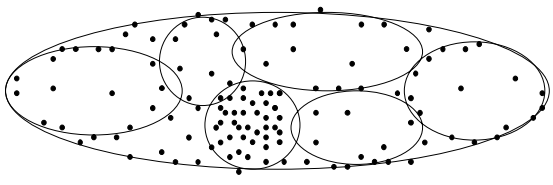


Cover the whole SUT with test cases

- Both inside and outside its value domain boarders

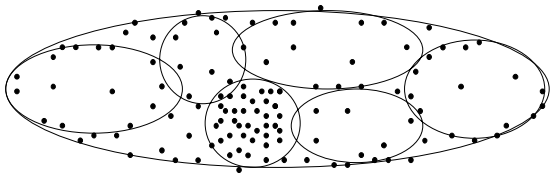


## Cluster of error prone SUT



- If we run the tests and find a cluster of errors in some part (e.g., code, task or component) we should analyse and redesign that part rather than just try to debug it, doing small changes

## Complex SUT require more tests



- Use a good programmer/designer to make complicated algorithms, components and tasks.
- Apply more tests to complicated units.
- Normalize the error rate before judging the quality of complex unit.



## Test Coverage

- What is tested, how many percent of all possible (real world) cases are covered?
- What should be tested?
- Where should we focus our efforts?
- Should we make our best to get it right in the first try or should we try, debug and try again?
- Statistical measures can be used to indicate the test coverage and the quality of the test.



## Statistical Testing

- Testing can be performed as a statistical experiment
- A representative subset of all possible uses of the SUT is randomly generated
- The performance on a subset is used as a basis for conclusions about the general performance
- A “sample” is used to draw conclusions about the whole design or “population”



## Statistical Quality Control

- Incremental development processes are suitable for statistical quality control
- Each increment of the process is compared with standard to determine if the process is “in control”
- If the quality standard is not meet developers return to the design stage
- Feedback produced in each increment is used for project management and process improvement



## Mathematical model or program

The system can be modelled:

- **declarative** using mathematical function and logical condition terms
- **imperative** using program implementation oriented terms

Control and signal processing systems can be modelled using:

- established mathematical modelling methods
- complemented with conditional control or guard expressions to tell when different mathematical models are valid



## Modelling or testing of a program

- It is not obvious that a pure computer program needs to be modelled and simulated in order to be understood better
- This since it is already a description at a rather high abstraction level
- Rather, it is common that a program (or a part of it) is tested as it is
- Using test patterns as models of its use or environment

## Early version program as model

- A computer program can be more or less complete when tested
- We can get important analysis information early by running and evaluating a preliminary simplified version of the program
- Such a program must not be valid under all possible conditions – if those conditions are reasonably well known

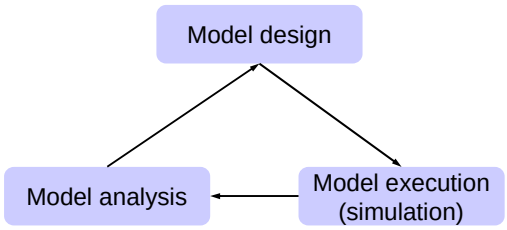


## Early version program, cont.

- A preliminary version (prototype) of a program used as model can later – if it stands the test – be extended to a complete version
- In other cases one may prefer to restart and make a production version from scratch, just using knowledge obtained by testing the prototype
- Such learning's and experiences should result in an refined requirement specification for the program



## What is computer simulation?




Analysis by help of simulation and modelling embodies the principle of "learning by doing"



## Response analysis

- Response, from the modelled SUT, to stimuli from its environment – should be analyzed by comparison with the expected response
- As designers we must thus have or acquire some idea about what is reasonable to expect as response to a given stimuli in a given state



## Response analysis, cont.

- In a model with internal state the response is often a function of its stimuli sequence history

  $r(t) = f(s(t-1), s(t-2), \ldots, s(t-n))$

  where f = function, r = response, s = stimuli and t = timepoint
- State dependent behaviour is sometimes called sequential or "**state full**" behaviour
- The opposite is called non state dependent, "**state less**" or combinatorial behaviour



## Stimuli - response analysis

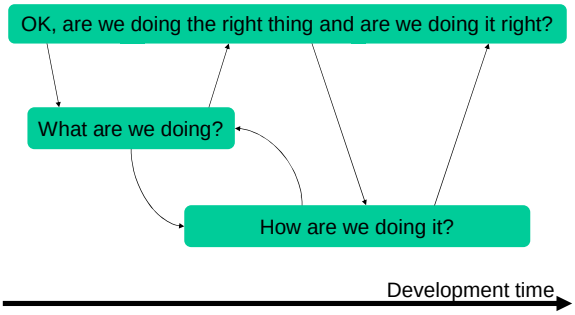
- A test is defined as a set of related stimuli and response patterns
- The designer can analyse if the system is correct, by comparing:
  - two different models of the same SOT
  - model and real SOT
  - model and environment (as expected stimuli response patterns)
  - real SOT and real environment compatibility

## Level of abstraction

- Create a model to answer your questions
- The answers can give rise to new questions
- Chose a level of abstraction suitable for the given purpose, e.g. how detailed or accurate the model must be for a certain question or analysis need



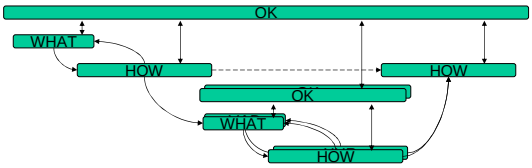
## What – How – Is it OK?





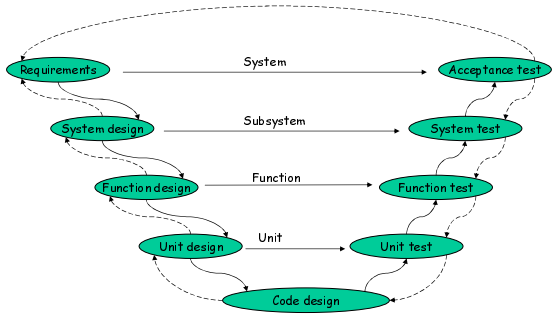

## Hierarchical Development

1. A system can be described as a set of components that are connected to each other in a hierarchical structure
2. We can partition and execute the development process following the system structure






## The V model





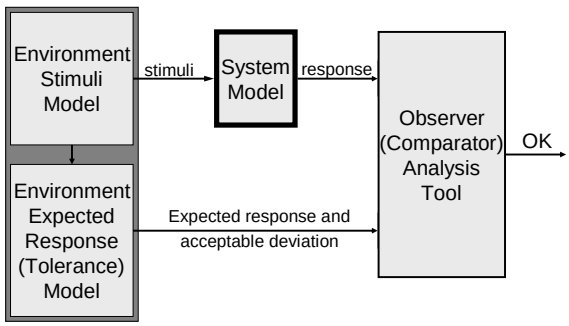

## Setup of simulation experiment

The simulation setup can be divided in:

1. A model of the system
2. A model of the environment, including
   a) stimuli
   b) expected responses
   c) temporal ordering of stimuli and response pairs
3. A tool to observe and record responses with support for comparisons and analysis



## Simulation setup, cont.



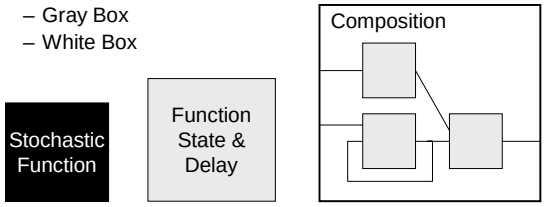
## Environment model

- The environment can be defined by models of other surrounding systems communicating with and generating stimuli to the modelled system
- An alternative is to describe the environment directly by how data sent from and to the system change over time
- Such stimuli and responses can be defined in a file or be created by a (pseudo or real) random generator
- Efficient comparisons require synchronization of:
  - environment
    - stimuli
    - expected response
  - model (SUT)
  - observer



## Abstraction and information hiding

- A model can be more or less transparent and thus hide or expose details
- Examples of abstraction (hiding detail) levels are:
  - Black Box
  - Gray Box
  - White Box






## Abstraction level metaphors

**Black box**
describes the function of a system (leaving out timing and internal state). Each input stimuli is related to probability of all possible output or the most common output case.

**Grey box**
adds state transition and other temporal behaviour such as execution and queuing delay to the functional model.

**White box**
captures the system's internal **composition** of subsystems (components) and their **connections.**

The **behaviour of a white box model:**
is captured indirectly by the behaviour of its internal parts and their interaction. Each such internal part can in its turn have a black, grey or white model.



## Observer

An observer should capture all data and events of interest:

- Log the system's output response history
- Enable analysis of stimuli and response relationships
- Compare the expected response with the output from the system model

For these purposes the observer contains:

- probe, log, filter and viewer functions



## Why use simulation?

- Complex dynamic behaviour due to many variables or interacting components
- Underlying variables and their relationships are nonlinear
- Model contains random variables
- The model output can often be visualized, as in a 3D computer animation



## Simulation Models

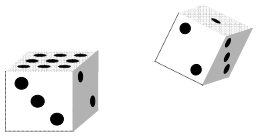
Models can take many forms, including:

- Discrete or continuous
- Declarative, procedural, functional, or constraint
- Spatial or temporal
- Symbolic or numeric (integer, float, complex)

Models can also have:

- Several forms (multi-model)
- Several abstraction levels (multi-level)
- Focus on particular views or aspects

## Generation of stimuli



- Create stimuli (e.g. task computation times and periods) using some statistically based arrival distribution reflecting the reality
  e.g. random, independent arrival process with defined mean and standard deviation



## Examples of simulations

**Scheduling algorithm** (static or dynamic)
by applying random task sets

**Communication protocol**
by setting up traffic between sources and receivers, using different protocol message distributions

**Overload handling**
by applying randomly intense event or message (activating new jobs) arrival processes (or a ramp, step or pulse arrival function)



## Examples of simulations, cont.

**Worst case response time under overload**
e.g. effect of different prioritization or rejection principles

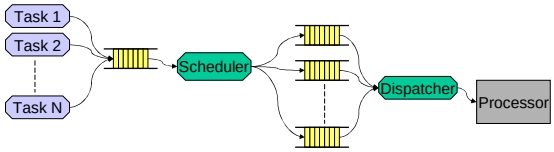
**Behaviour in case of disturbance**
by modelling of the disturbance (e.g., an error, fault or failure or other aperiodic or sporadic job arrival)

**Algorithm for distributed task allocation**
by applying random task sizes and varying number of processing resources

**Distributed task allocation and scheduling**
analysed for different task sets, processor configurations, allocations, schedules or scheduling methods



## Example: Simulation of scheduler




- The stimuli can be a set of tasks triggered at periodic, aperiodic or sporadic intervals to be handled by the scheduler and dispatcher
- This stimuli can be aimed to evaluate the function or characteristics of the scheduler or dispatcher



## Simulation Languages and Tools

- Any programming language can be used to create simulation models, both of system and environment
- Mathlab, Simulink, OPNET, NS-2 and NS-3 are examples of language/tool environments that can be used for simulation of distributed real-time systems
- Simulation is used in many disciplines, using more or less specialized tools



## Execution of simulation

- Create a computer program which steps through time or an event queue while updating the state and event variables in your model
  - Follow the dynamic evolution of the simulation time using event scheduling
  - Employ small time increments using time slicing
  - To speed up the simulation you can execute the program on a parallel computer