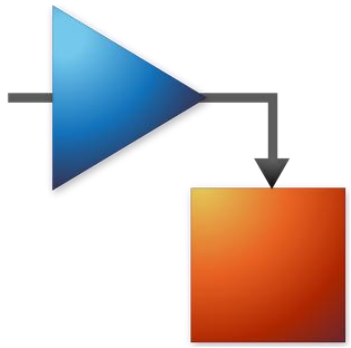
Discussion: First Consultation

- UML
- What to discuss?
- Interactions during consultation

Date	Teams
Mon April. 8th 3-5pm	Team 1-6
Wen April. 10th 3-5pm	Team 7-12
Mon April. 15th 3-5pm	Team 13-18

Lecture 12: Simulink & Stateflow

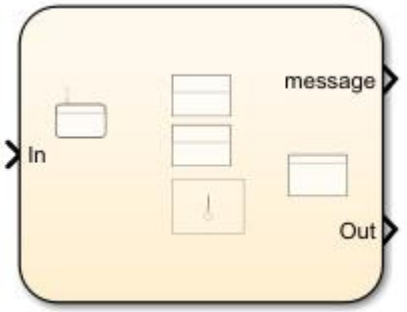


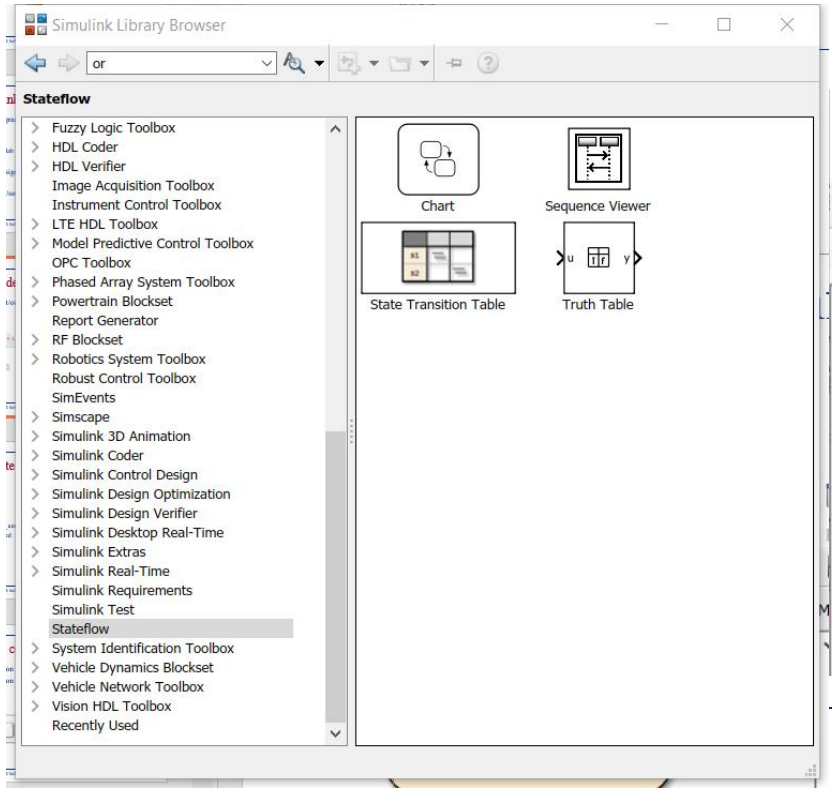
Simulink & Stateflow

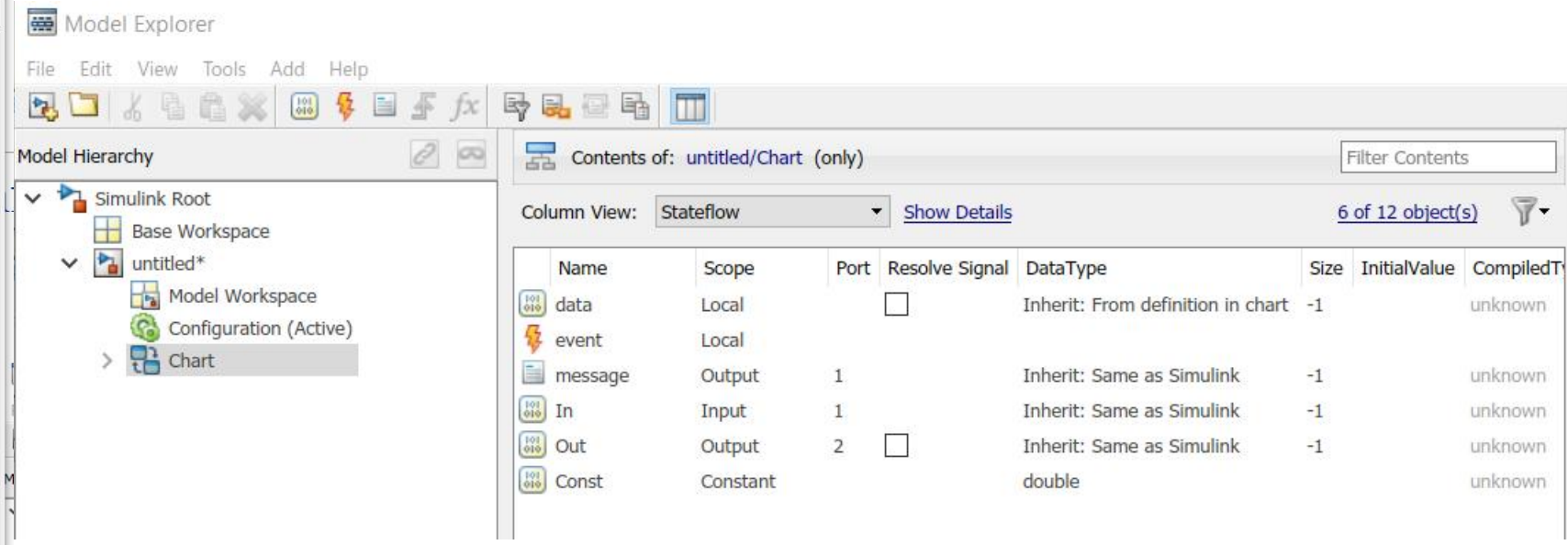
- A graphical modeling/programming language
- Developed by Mathworks
  - Highly integrated with Matlab
- Has a full model-based design toolchain
- Widely adapted by system/software developers
- Rich expressiveness

Model Explorer

- Define and configure input/output, event/message, variables.

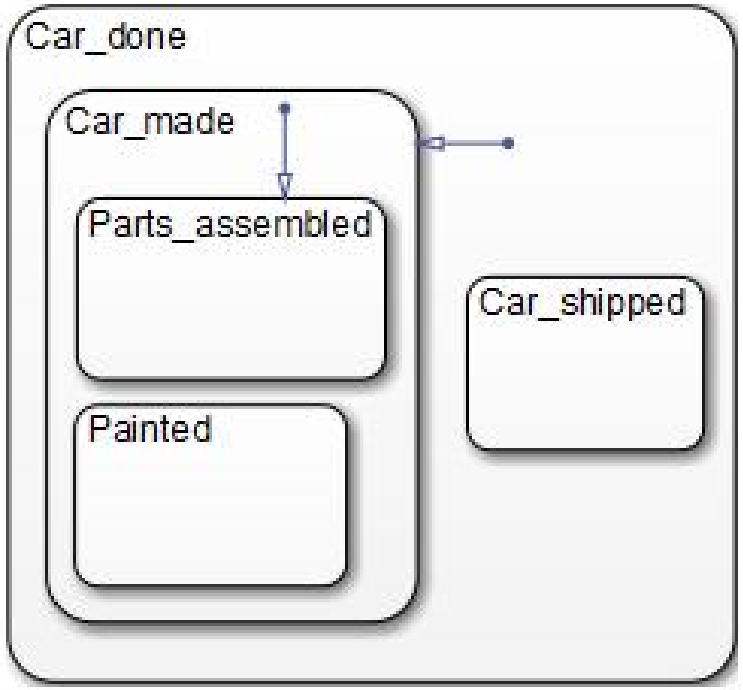






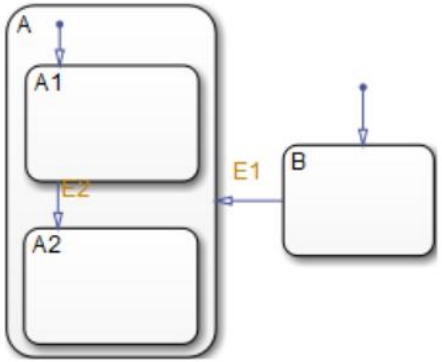
State Hierarchy

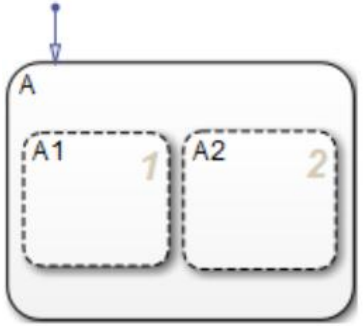
- Object oriented modeling
  - /Car_done
  - /Car_done.Car_made
  - /Car_done.Car_shipped
  - /Car_done.Car_made.Parts_assembled
  - /Car_done.Car_made.Painted

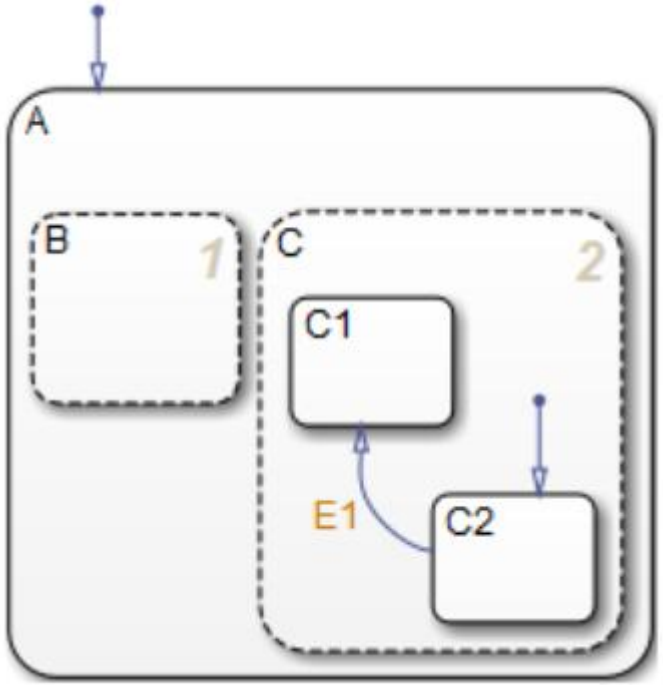


State compositions

- "OR"/exclusive composition
- "AND"/parallel composition

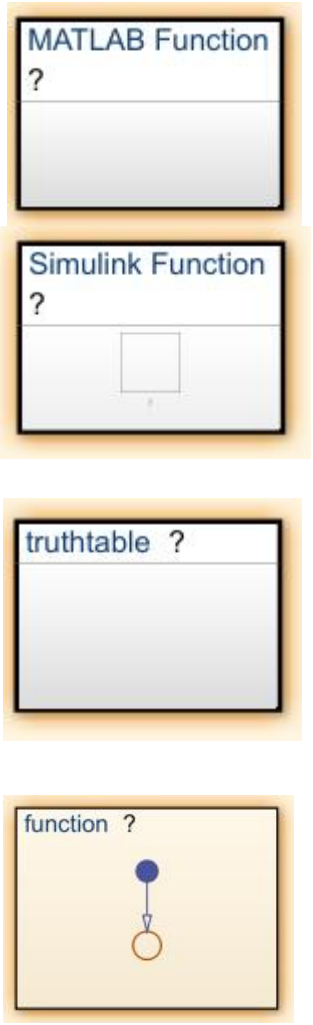






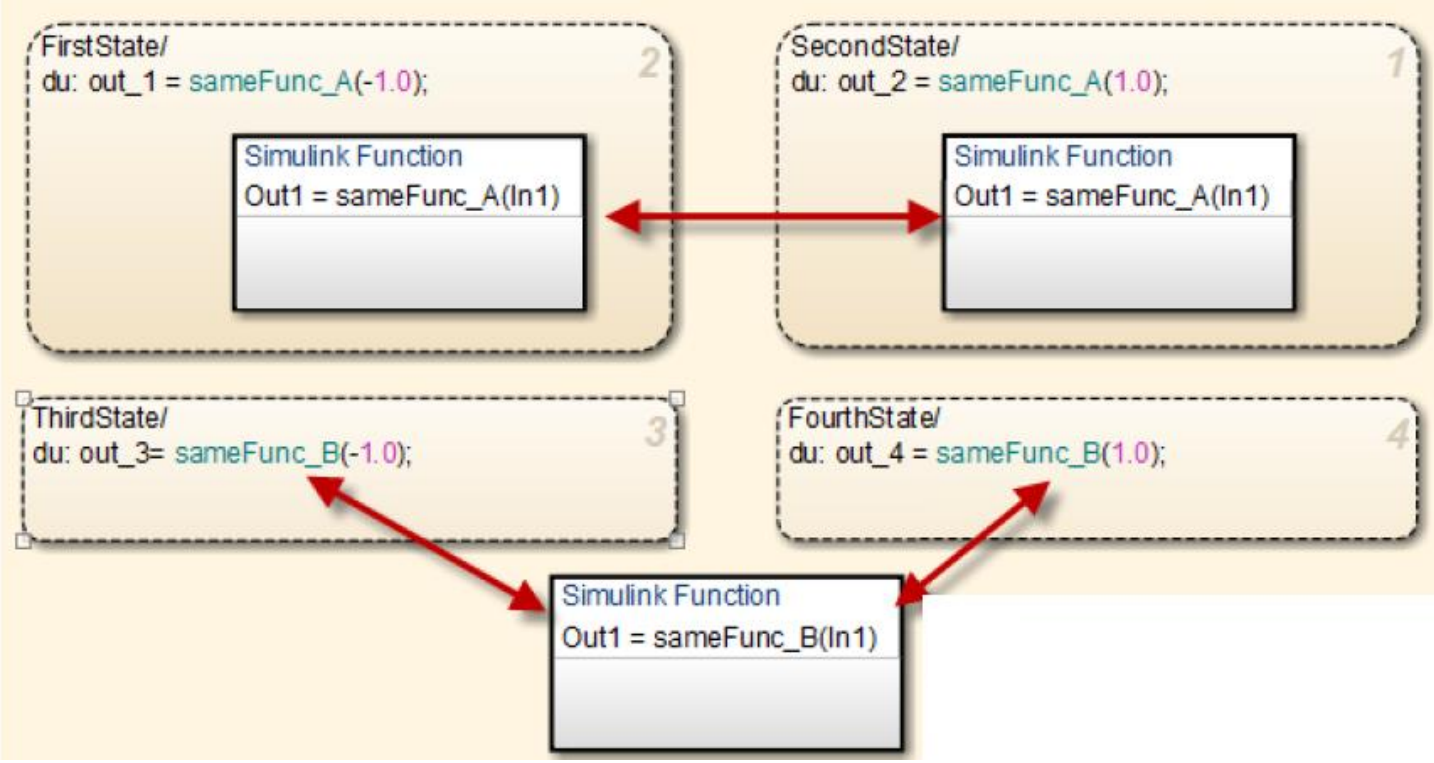
Embedded Functions

- Matlab function
- Simulink function
- Truthtable
- Graphical function



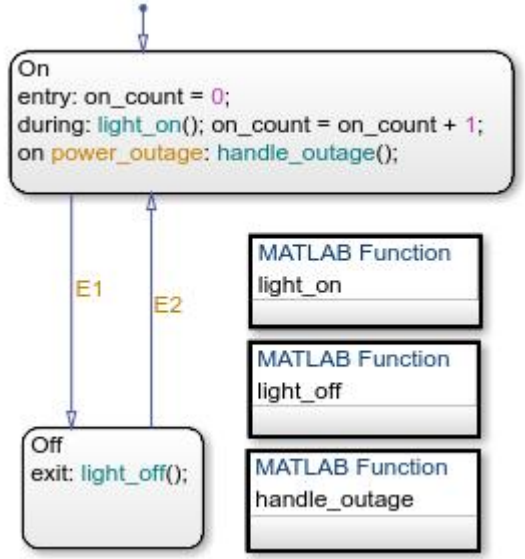
Function availability

- Only available to states at the same level



State Actions

- entry: (en:) entry actions
  - Action occurs on a time step when the state becomes active.
- during: (du:) during actions
  - Action occurs on a time step when the state is already active and the chart does not transition out of the state.
- exit: (ex:) exit actions
  - Action occurs on a time step when the chart transitions out of the state.
- on event_name: on event_name actions
- on message_name: on message_name actions



Reduce redundancy

```
en:
  fcn1();
  fcn2();
du: fcn1();
ex: fcn1();
```

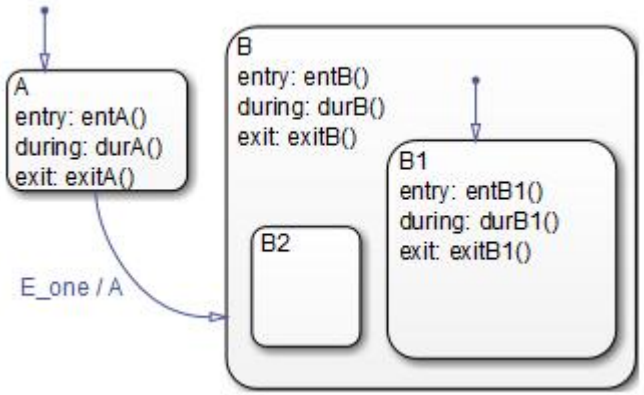
```
en, du, ex: fcn1();
en: fcn2();
```

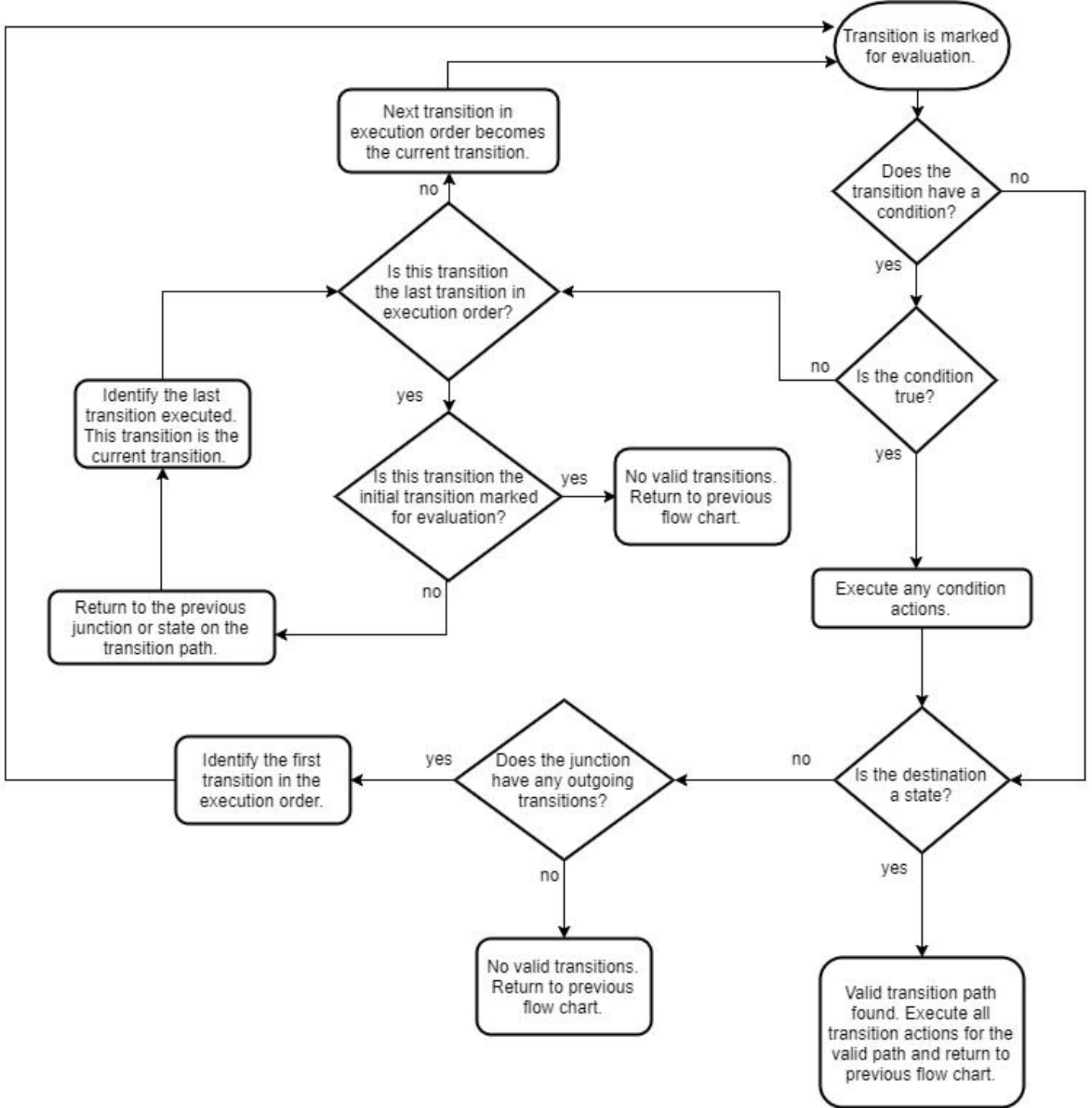
Transition

- event_or_message[condition]{condition_action}/transition_action
- Condition
  - Boolean expression that specifies that a transition path is valid if the expression is true; part of a transition label
- Condition actions
  - Executes after the condition for the transition is evaluated as true, but before the transition to the destination is determined to be valid
- Transition actions
  - Executes after the transition to the destination is determined to be valid

Default transitions

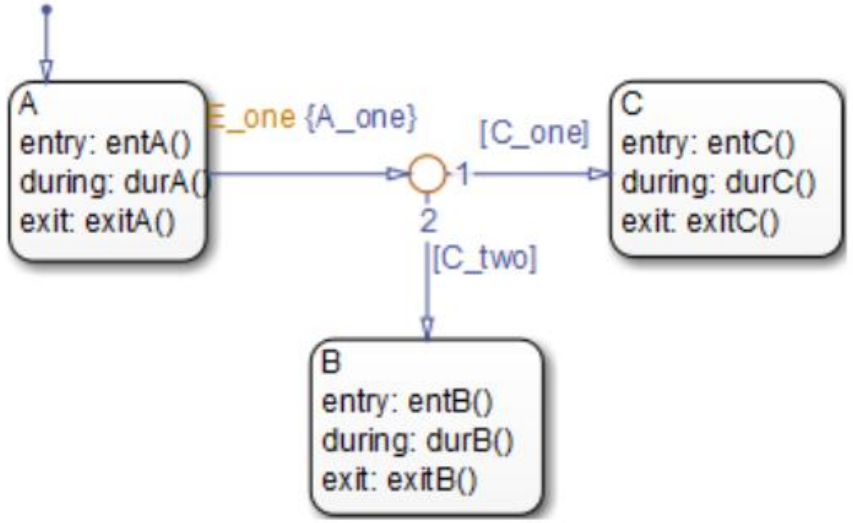
- State A is active. Event E_one occurs and awakens the chart
- State A exit actions (exitA()) execute and complete.
- State A is marked inactive.
- The transition action, A, is executed and completed.
- State B is marked active.
- State B entry actions (entB()) execute and complete.
- State B detects a valid default transition to state B.B1.
- State B.B1 is marked active.
- State B.B1 entry actions (entB1()) execute and complete.
- The chart goes back to sleep.




Transition is marked for evaluation.
Does the transition have a condition?
no
yes
Is the condition true?
no
yes
Execute any condition actions.
Is the destination a state?
no
yes
Valid transition path found. Execute all transition actions for the valid path and return to previous flow chart.
Does the junction have any outgoing transitions?
yes
no
Identify the first transition in the execution order.
No valid transitions. Return to previous flow chart.
Is this transition the last transition in execution order?
no
yes
Is this transition the initial transition marked for evaluation?
yes
no
No valid transitions. Return to previous flow chart.
Identify the last transition executed. This transition is the current transition.
Return to the previous junction or state on the transition path.
Next transition in execution order becomes the current transition.

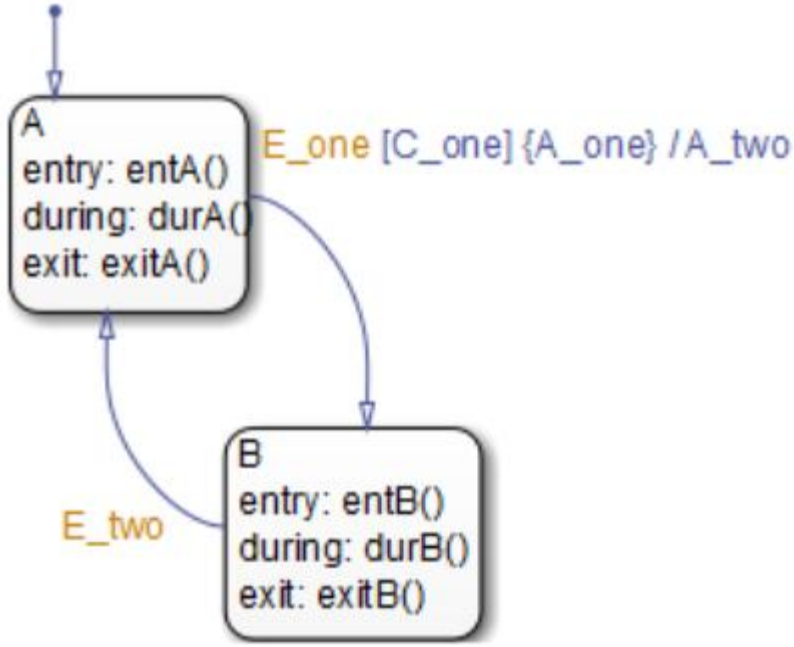
Condition Action Behavior

- E_one happened when state A is active. C_one and C_two are false
- A valid transition segment from state A to a connective junction is detected.
- The condition action A_one is immediately executed and completed. State A is still active.
- No complete transitions is valid.
- State A during actions (durA()) execute and complete.
- State A remains active.
- The chart goes back to sleep.



Condition and Transition Action Behavior

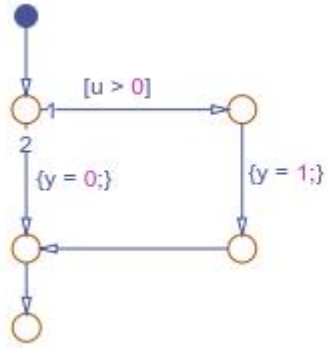
- E_one happened and awaked the chart.
- The condition C_one is true. The condition action A_one is immediately executed.
- State A is still active.
- State A exit actions (ExitA()) execute and complete.
- State A is marked inactive.
- The transition action A_two is executed.
- State B is marked active.
- State B entry actions (entB()) execute.
- The chart goes back to sleep.



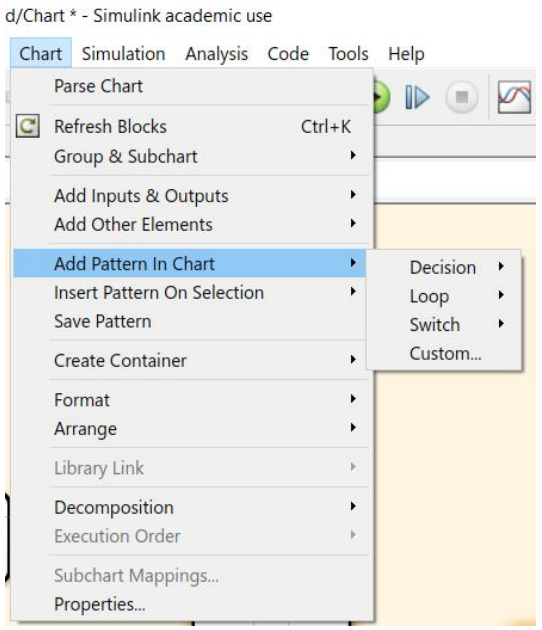
Flow chart

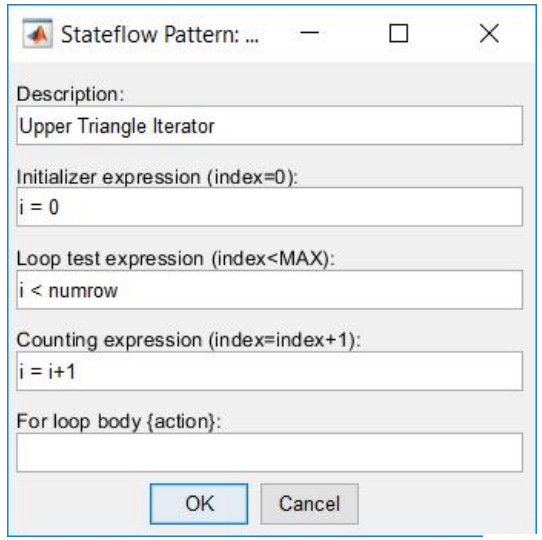
- No time consumption during execution
- Can be used for graphical function definition

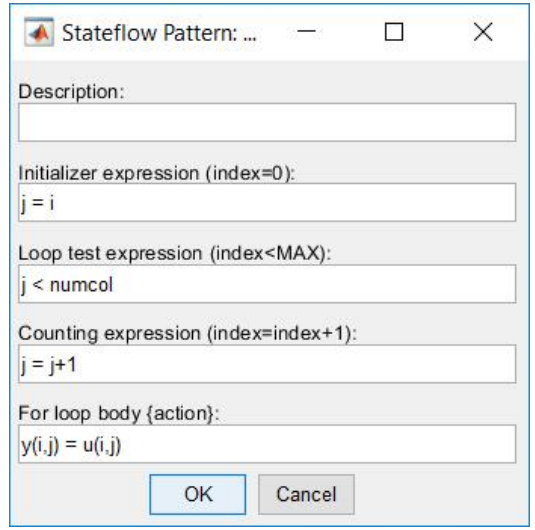
```
if u > 0
   y = 1;
else
   y = 0;
end
```

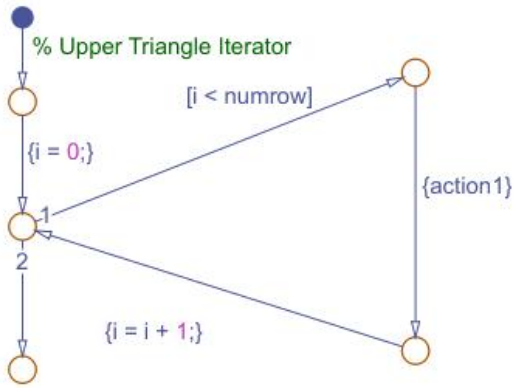


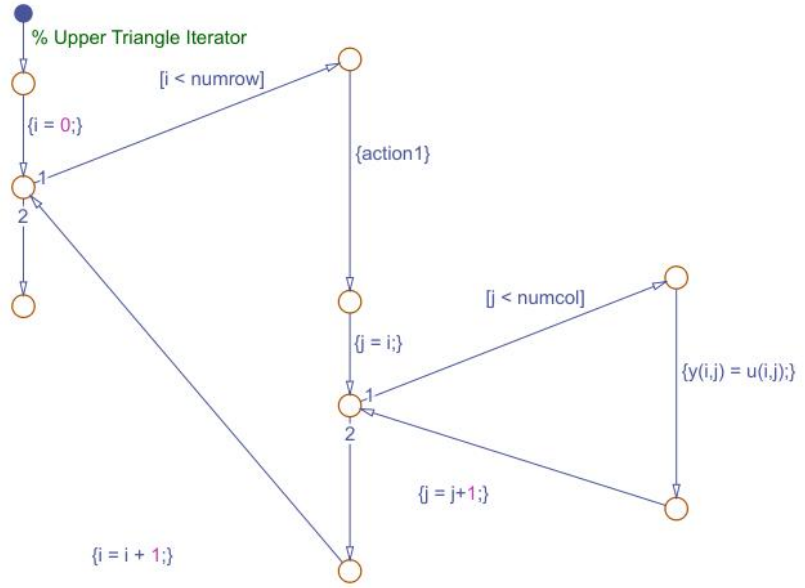
Add pattern in chart







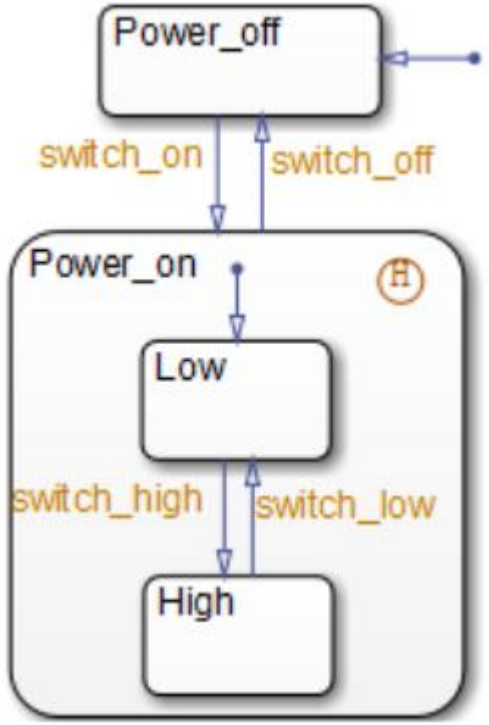




History Junction

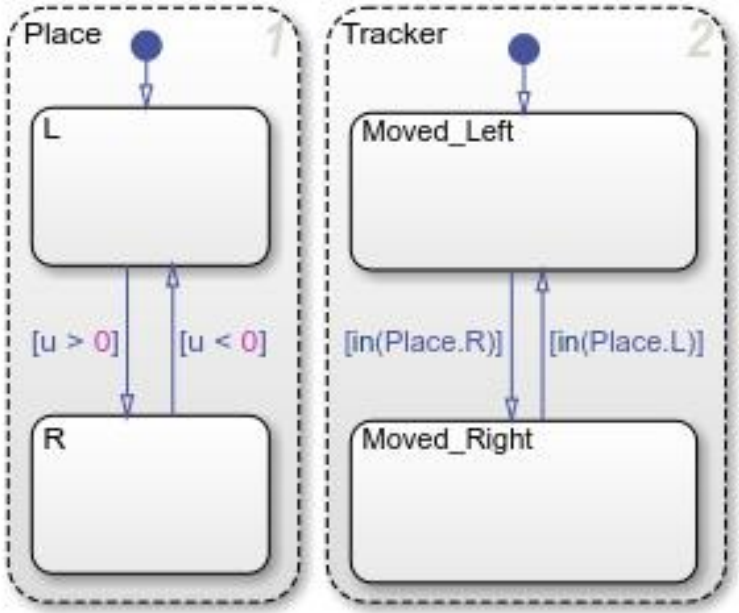
- Restores the state that is on the same level of the composite state as the history state itself

- If the system was switched off when the system was at the “High” state, when the system is switched back on, it will start from the “High” state



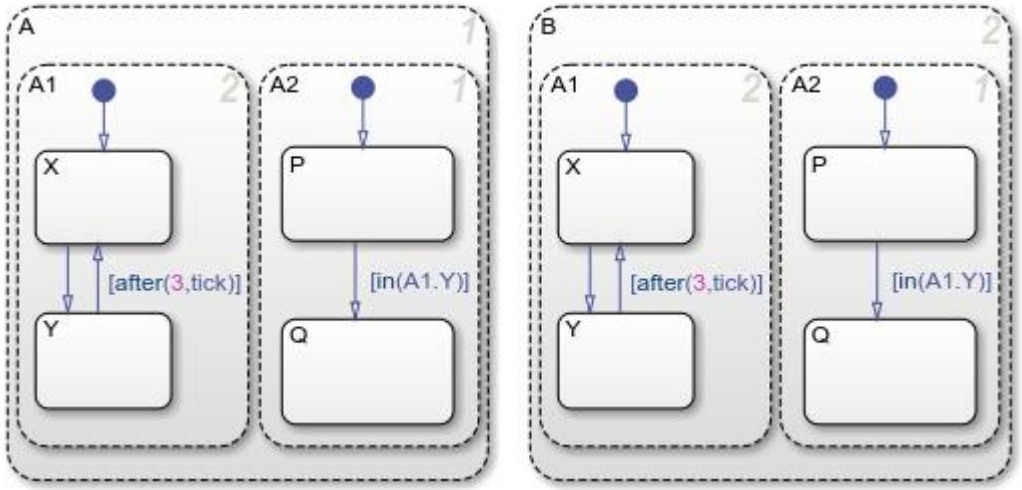
Check State Activity by Using in() Operator

- We can use in() operator to reference status of other parallel states
  - Return 1 if the referenced state is also active
- Starting point
  - If in state action, start from the containing state
  - If on transition, start from the parent state
- Search up the state hierarchy until the chart level is reached



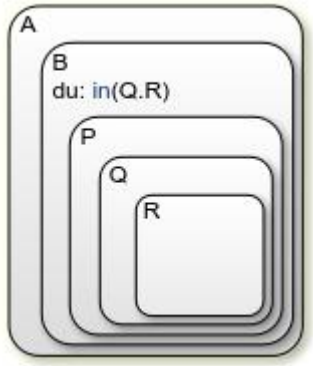
In() operator example

- In(A1.Y) in both A and B only find local copies of A1.Y
- Because at the chart level, there is no A1.Y

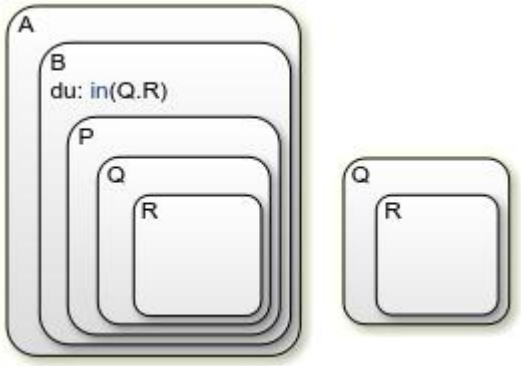


In() operator example (cont.)

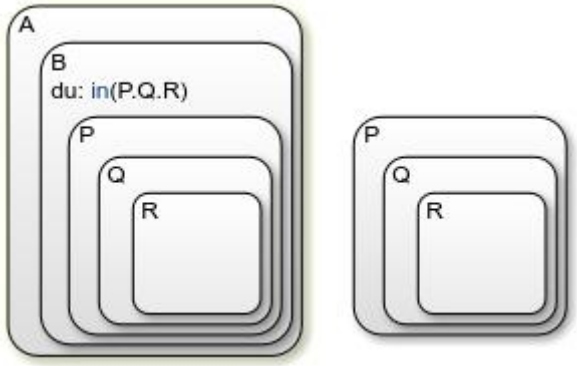
No match



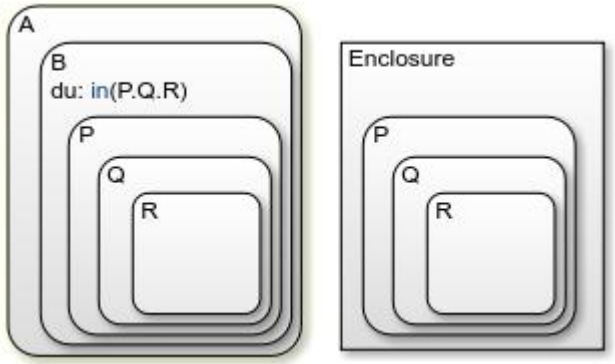
In(P.Q.R) will do
Wrong match



In(B.P.Q.R) will do
Multiple match

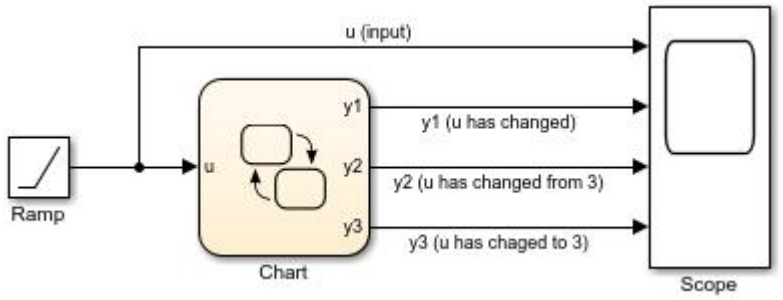


Use enclosure to ensure local match

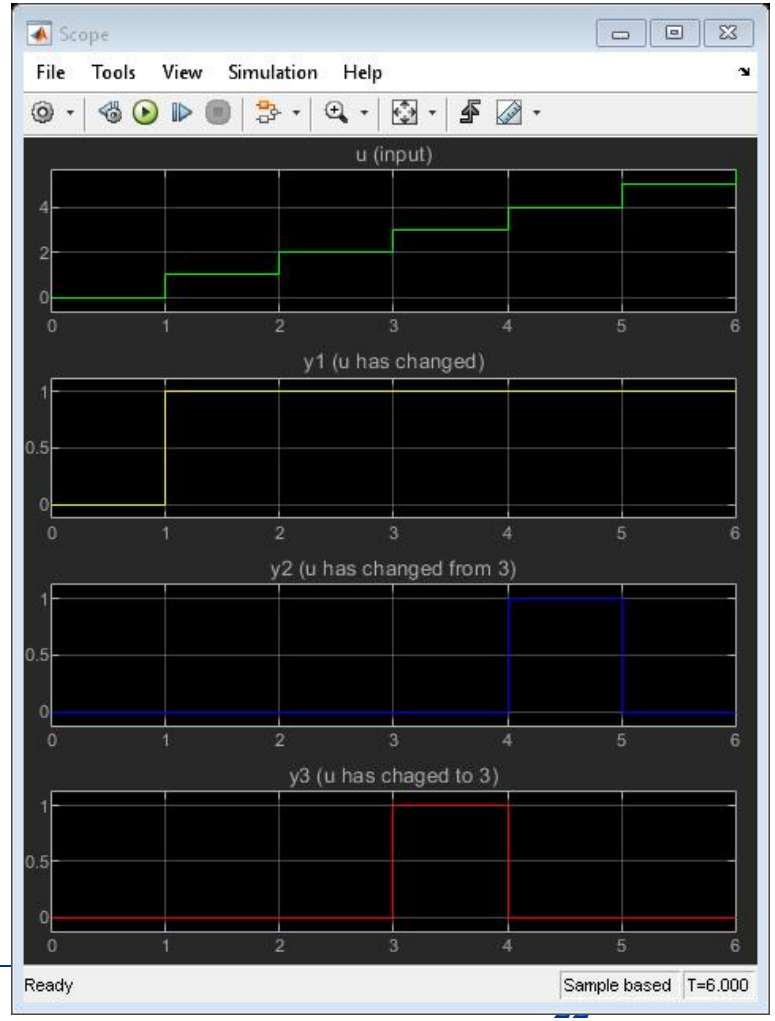


Detect data change

- hasChanged(u)
  - Detects changes in data value from the beginning of the last time step to the beginning of the current time step.
- hasChangedFrom(u,v)
  - Detects changes in data value from a specified value at the beginning of the last time step to a different value at the beginning of the current time step.
- hasChangedTo(u,v)
  - Detects changes in data value to a specified value at the beginning of the current time step from a different value at the beginning of the last time step.

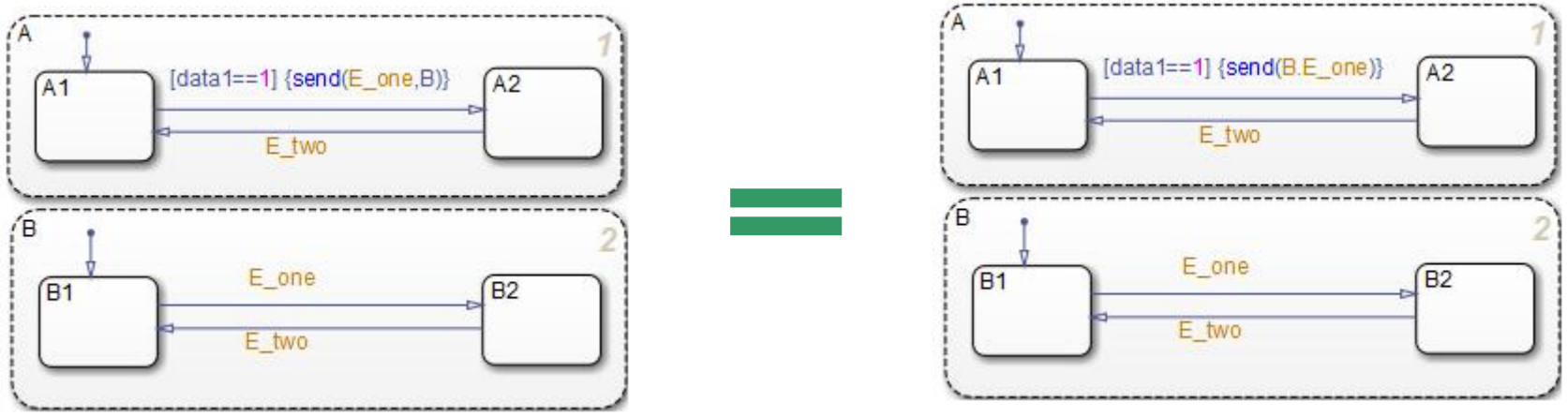



```
{
y1 = hasChanged(u);
y2 = hasChangedFrom(u,3);
y3 = hasChangedTo(u,3);
}
```



Broadcast Local Events to Synchronize Parallel States

- send(event_name,state_name)
- event_name is broadcast to its owning state (state_name) and any offspring of that state in the hierarchy.
- The receiving state must be active during the event broadcast.
- An action in one chart cannot broadcast events to states in another chart.

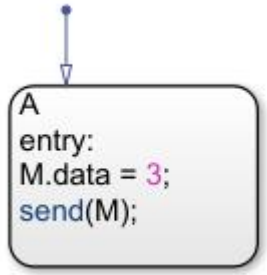


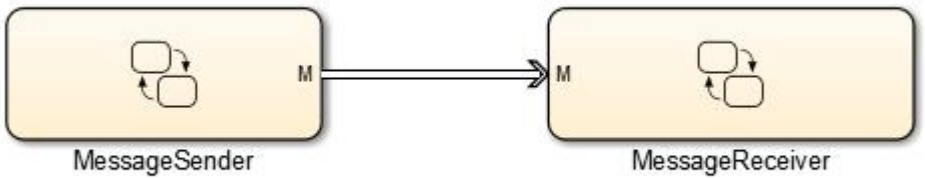
Implicit Events

- change(data_name) or chg(data_name)
  - generates a local event when writing a value to the variable data_name
  - Data_name has to be at chart level or lower
- enter(state_name) or en(state_name)
  - generates a local event when the specified state_name is entered
- exit(state_name) or ex(state_name)
  - generates a local event when the specified state_name is exited
- Tick/wakeup
  - generates a local event when the chart of the action being evaluated awakens

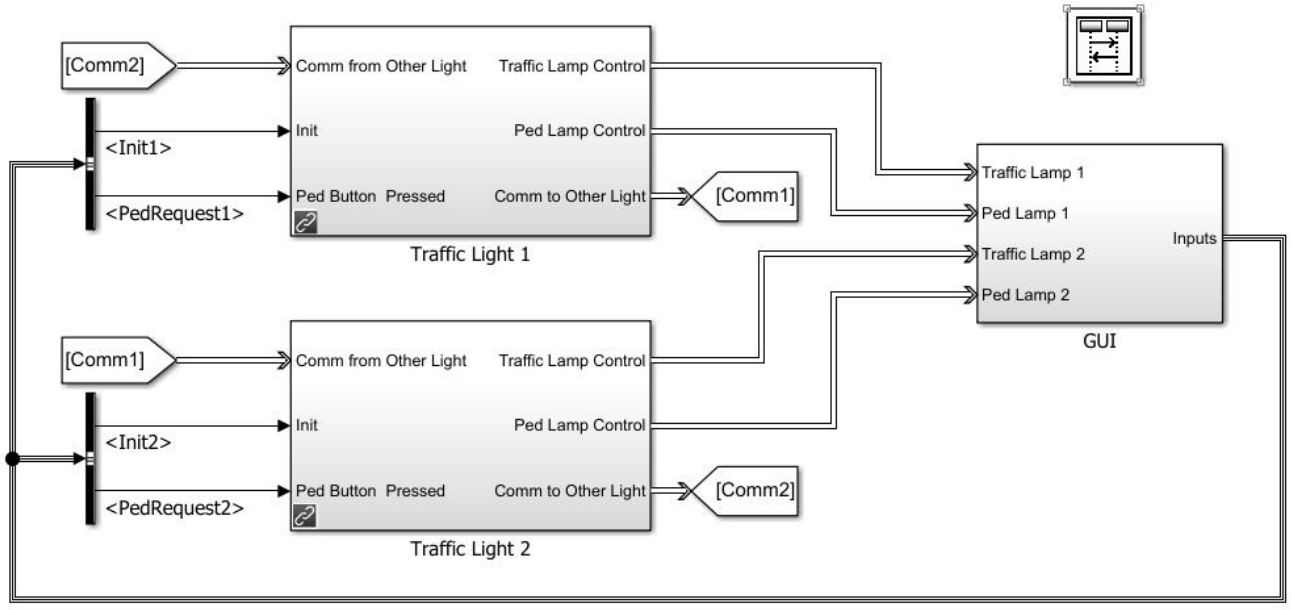
Message

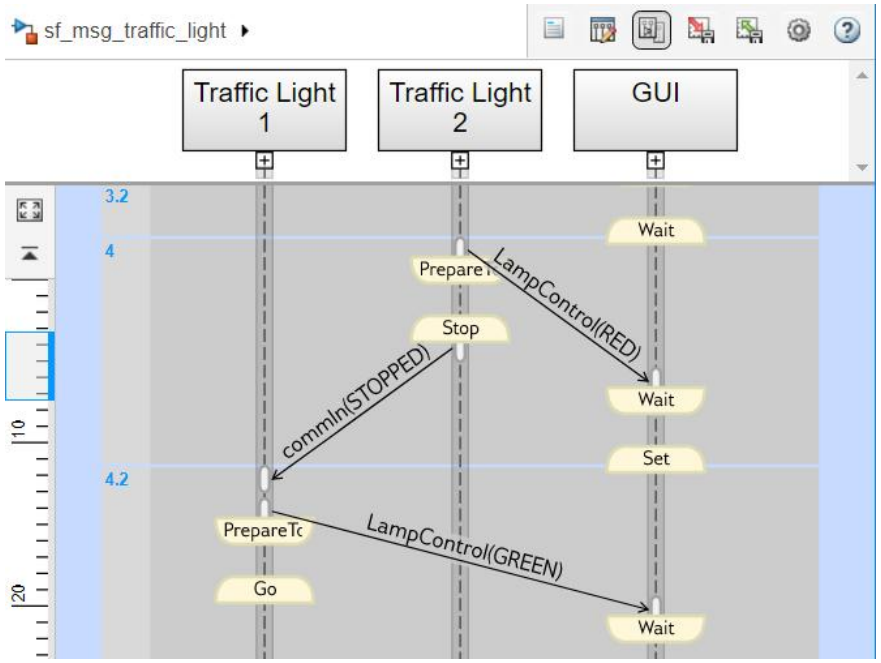
- Contains data: Message_name.data
- Receiver has a queue for each input message
- send(message_name)
- receive(message_name)
- discard(message_name)
- forward(input_message_name, output_message_name)
- isvalid(message_name)
  - if the chart has removed it from the queue and has not forwarded or discard



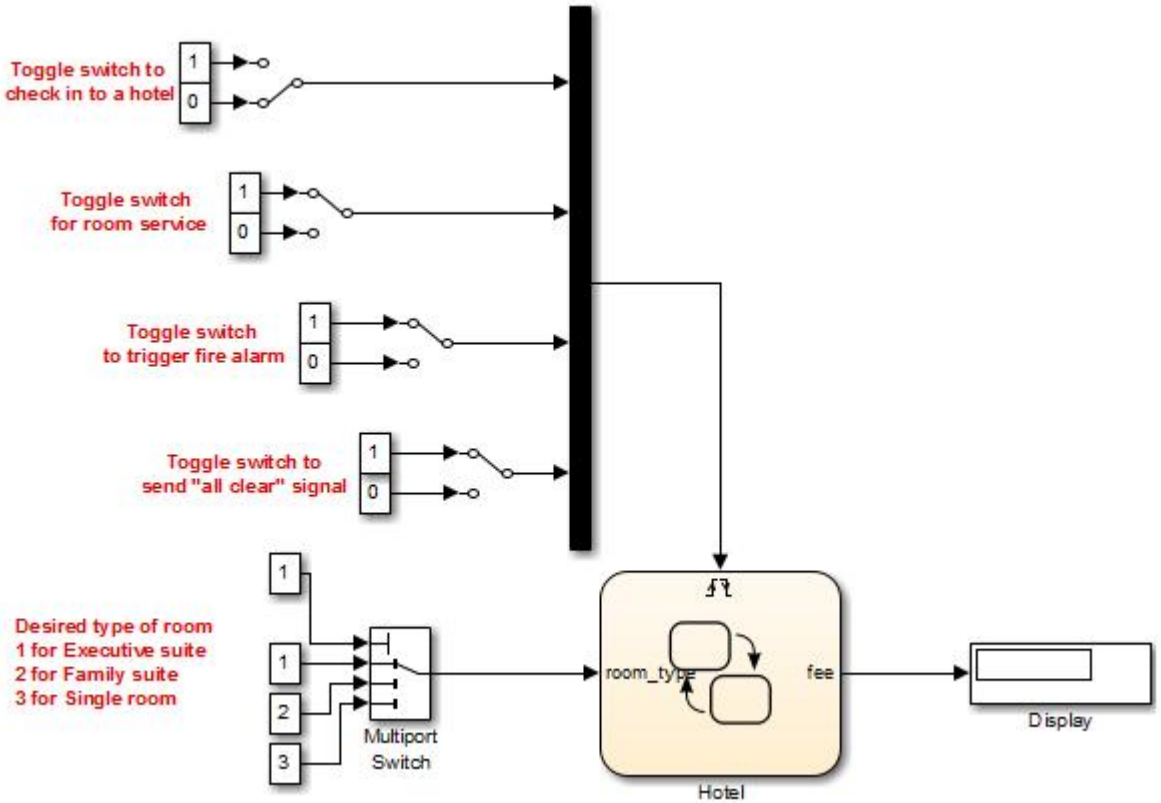


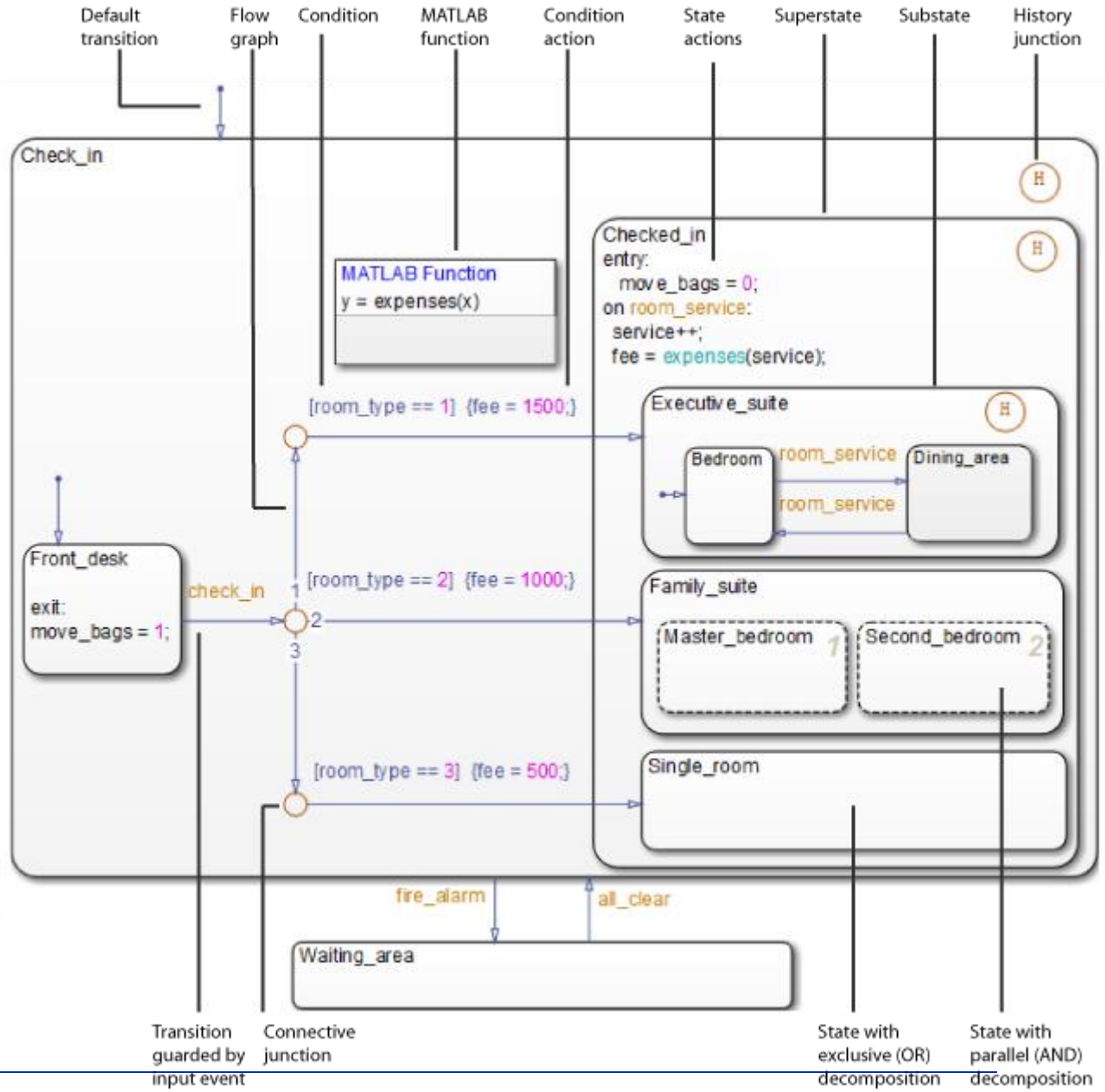
Visualizing messages/events





Example


Toggle switch to check in to a hotel
Toggle switch for room service
Toggle switch to trigger fire alarm
Toggle switch to send "all clear" signal
Desired type of room
1 for Executive suite
2 for Family suite
3 for Single room
Multiport Switch
room_type
fee
Hotel
Display

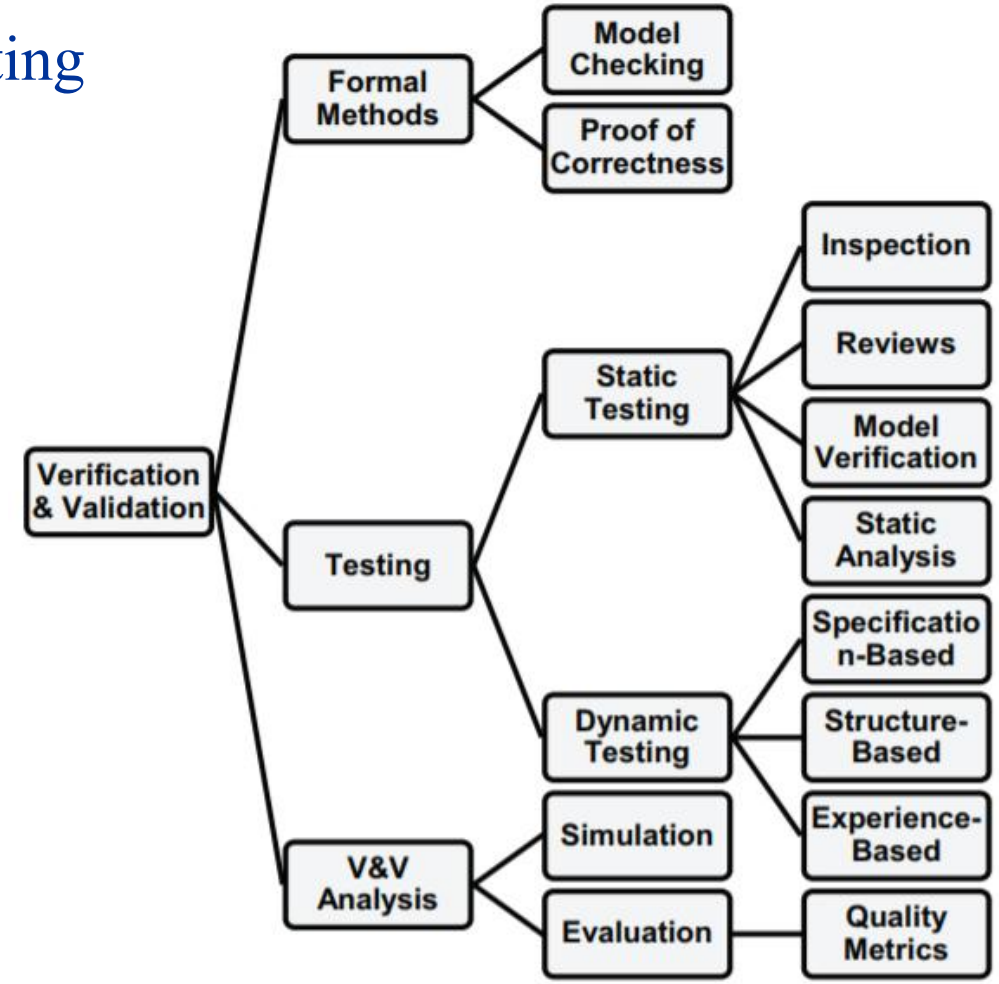

Default transition
Flow graph
Condition
MATLAB function
Condition action
State actions
Superstate
Substate
History junction
Check_in
MATLAB Function
y = expenses(x)
Checked_in
entry:
move_bags = 0;
on room_service:
service++;
fee = expenses(service);
[room_type == 1] {fee = 1500;}
Executive_suite
Bedroom
room_service
Dining_area
room_service
Front_desk
exit:
move_bags = 1;
check_in
[room_type == 2] {fee = 1000;}
Family_suite
Master_bedroom
Second_bedroom
[room_type == 3] {fee = 500;}
Single_room
fire_alarm
all_clear
Waiting_area
Transition guarded by input event
Connective junction
State with exclusive (OR) decomposition
State with parallel (AND) decomposition

Modeling Tips

- Use signals of the same data type for input events
- Use a default transition to mark the first state to become active among exclusive (OR) states
- Use condition actions instead of transition actions whenever possible
- Use explicit ordering to control the testing order of a group of outgoing transitions
- Use MATLAB functions for performing numerical computations in a chart

Testing in V&V

- Both verification and validation involves testing
- Verification
  - Whether the code conform with software specification
  - Conformance testing
- Validation
  - Functional testing
  - Scenario testing
  - Risk based testing



Who and when?

- Testing is performed by *test engineers* after the initial development
  - Test suite should be constructed during development
- Test designer may be different from test engineers
- The test engineers should be able to do it without participating the development process

Terminologies

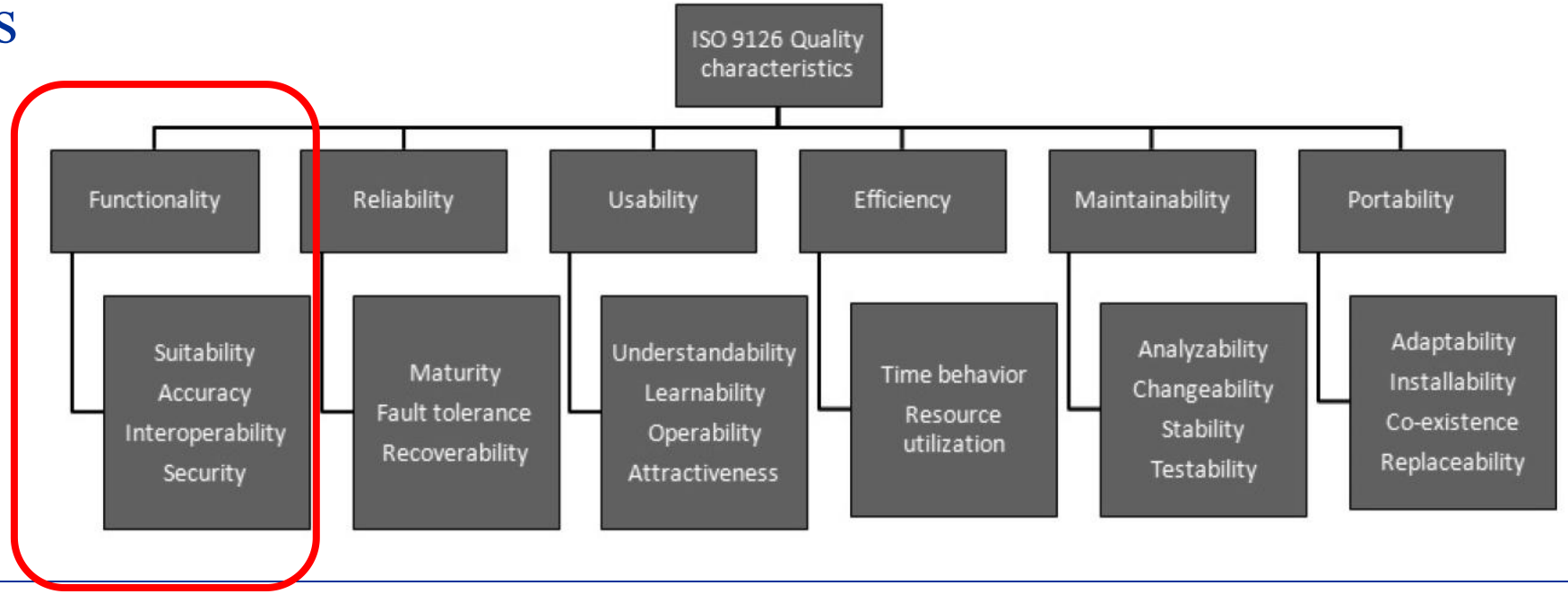
- International Software Testing Qualifications Board (ISTQB)
  - error: human action at the root of a defect;
  - defect: result of a human action (i.e. error), which is present in the test object
  - failure: result from the execution of a defect by a process (whether the process is automated or not).

Objectives of testing

- Software testing focuses on two complementary but distinct aspects:
  - Defect and failure detection
    - Fix the bugs
    - Improve the quality of the product delivered to customers and users
  - Establishing confidence to the software
    - Level of risks associated with the delivery of the software to the market
    - Capabilities of the developer team
  - 信为疑之余，疑为信之本

Software characteristics

- Functional characteristics
- Non-functional characteristics
- Non-functional tests should be performed together with functional tests



Testing vs. Risk management

- There is a tradeoff on how much should be tested
  - Principle similar to risk evaluation process
- Testing can be used to confirm the existence of risk

Integrity levels (IEEE 829-2008)

- level 4: catastrophic:
  - software must execute correctly or grave consequences (loss of life, loss of system, environmental damage, economic or social loss) will occur. No mitigation is possible;
- level 3: critical:
  - software must execute correctly or the intended use (mission) of system/software will not be realized causing serious consequences (permanent injury, major system degradation, environmental damage, economic or social impact). Partial-to-complete mitigation is possible;
- level 2: marginal:
  - software must execute correctly or the intended function will not be realized causing minor consequences. Complete mitigation possible;
- level 1: negligible:
  - software must execute correctly or the intended function will not be realized causing negligible consequences. Mitigation not required.

Testing vs. debugging

- Testing
  - Identify defects in software
- Debugging
  - Fix defects

Early Testing

- The cost of finding (and fixing) a defect
  - In the design phase is “1”.
  - In the coding phase, multiplied by “10”
  - During the test phase (system test or acceptance test), multiplied by “100”.
  - In production (by the customer or by a user), multiplied by “1,000”.

Defect clustering

- If you find a defect, it might be efficient to look for other defects in the same piece of code.
  - A complex section of an algorithm
  - Components designed by the same person
  - A group of components designed in the same period of time

Testing mentality

- A developer will try to find *one* solution to a particular problem, and will design the program based on the solution envisaged;
- A tester will try to identify *all* possible failures that might have been forgotten.
- A developer usually has two hats: the developer's and the tester's
- Our brain has a tendency to hide defects because it is looking for usual patterns.
- Need to have different perspectives

Independence level

- Same person
- Pair programming
- Independent tester from the same team
- From the same company
- Third party

Testing is not a destructive activity

- The defects were not introduced by testers, but developers
- It is not personal, it is just business
  - Point out defects should not look bad for developers
  - need good relational skills

Types of tests

- Component tests
- Component integration tests
- System tests, on the completely integrated system
- Acceptance tests, a prerequisite for delivery to the market or production.

Attributes of testing

- *Test object:* the target of the tests
  - a function, a sub-program, or a program, a software application, or a system made up of different sub-systems;
- *Specific objectives:* the reasons why the tests will be executed
  - To discover certain types of defects, to ensure correct operation, or provide any other type of information (such as coverage);
- *Entry and exit criteria:* when a task can start (the pre-requisites) and when it can be considered as finished.

Component level test/Unit test/Class test

- *Test object*:
  - components, program modules, functions, programs,
- *Objective*:
  - detect failures in the components,
- *Entry criteria*:
  - the component is available, compiled, and executable in the test environment;
  - the specifications are available and stable.

- *Exit criteria*:
  - the required coverage level has been reached;
  - defects found have been corrected;
  - the corrections have been verified;
  - regression tests have been executed on the last version and do not identify any regression;
  - traceability from requirements to test execution is maintained

Integration level testing

- *Test object*:
  - components, infrastructure, interfaces, database systems, and file systems.
- *Objective*:
  - detect failures in the interfaces and exchanges between components.
- *Entry criteria*:
  - at least two components that must exchange data are available, and have passed component test successfully.
- *Exit criteria*:
  - all components have been integrated and all message types (sent or received) have been exchanged without any defect for each existing interface;
  - statistics (such as *defect density*) are available;
  - the *defects* requiring correction have been corrected and checked as correctly fixed;
  - the impact of not-to-fix defects have been evaluated as not important.

Types of integration

- Integration of software components
  - typically executed during component integration tests
- Integration of software components with hardware components
- Integration of sub-systems
  - typically executed after the system tests for each of these sub-systems and before the systems tests of the higher-level system.

Integration approaches

- Big bang integration
  - Con: difficult to isolate defects found in the next level
- Bottom-up integration
  - Con: need drivers to execute
- Top-down integration
  - Con: need mock objects (stubs) to execute
- Sandwich integration
  - Combinations of bottom up and top down approach
- Integration by functionalities
  - Intuitive
- Neighborhood integration

System Test

- *Test object*:
  - the complete software or system, its documentation, the software configuration and all the components that are linked to it
- *Objective*:
  - detect failures in the software, to ensure that it corresponds to the requirements and specifications, and that it can be accepted by the users.
- *Entry criteria*:
  - all components have been correctly integrated, all components are available.
- *Exit criteria*:
  - the level of coverage has been reached;
  - must-fix defects have been corrected and their fixes have been verified;
  - regression tests have been executed on the last version of the software and do not show any regression;
  - bi-directional traceability from requirements to test execution is maintained;
  - statistical data are available, the number of non-important defects is not large;
  - the summary test report has been written and approved.

Acceptance Test

- *Test object*:
  - the complete software or system,
- *Objective*:
  - obtain customer or user acceptance of the software.
- *Entry criteria*:
  - all components have been correctly tested at system test level and are available,
  - the last fixes have been implemented and tested without identification of failures,
  - the software is considered sufficiently mature for delivery.
- *Exit criteria*:
  - the expected coverage level has been reached;
  - must-fix defects have been corrected and the fixes have been verified;
  - regression tests have been executed on the latest version of the software and do not show regression;
  - traceability from requirements to test execution has been maintained;
  - user and customer representatives who participated in the acceptance test accept that the software or system can be delivered in production.

Acceptance Test

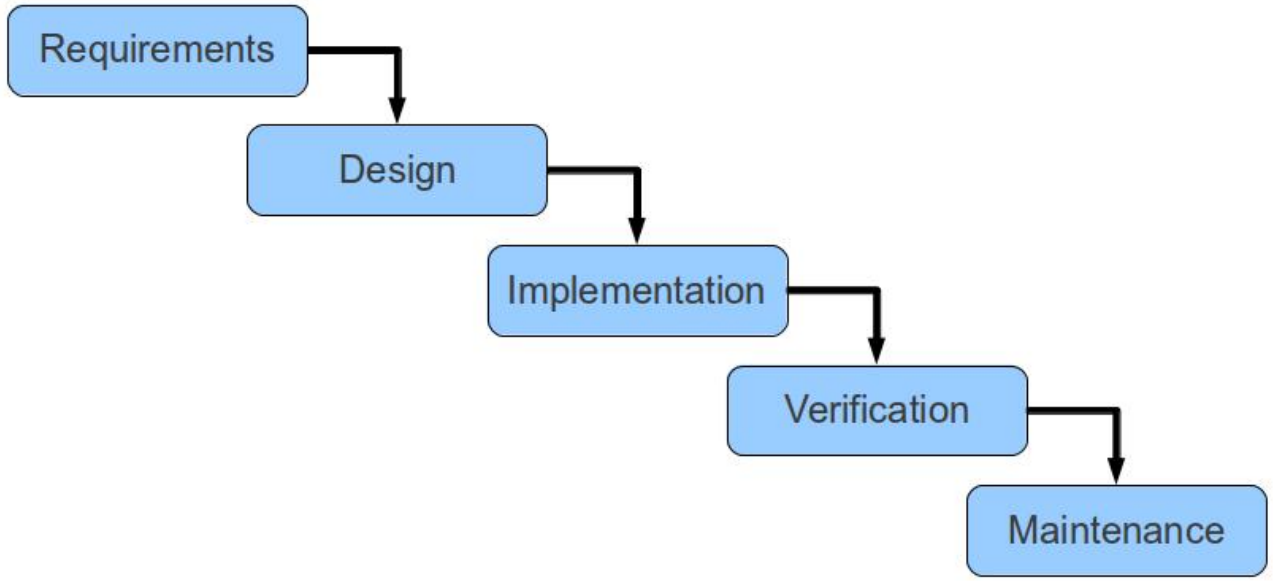
- user acceptance
- operational acceptance
- contractual acceptance
- regulatory acceptance
- alpha tests
- beta tests
- pilot phase

Model-independent testing principles

- Each design activity, and each deliverable, must have a corresponding test activity that will search for defects introduced by this activity or in this deliverable;
- Each test level has its own specific objectives, associated with that test level, so as to avoid testing the same characteristic twice;
- Analysis and design of tests for a given level start at the same time as the design activity for that level, thus saving as much time as possible;
- Testers are involved in document review as soon as drafts are available, whichever the development model selected.

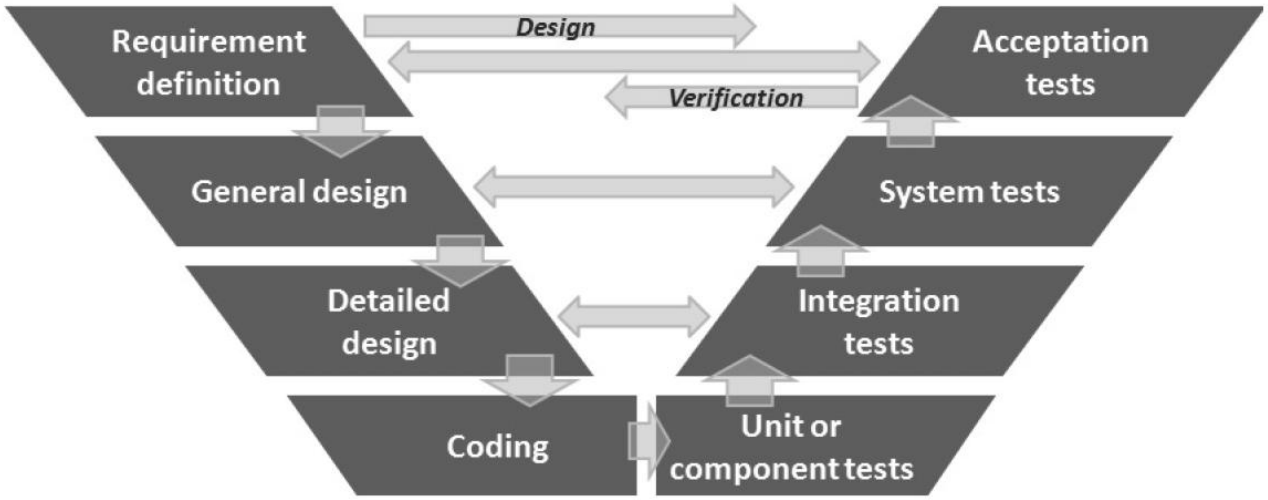
Waterfall software development model

- Applicable to applications with confirmed requirements at early stage
- Cons: Cost is huge once validation fails at later development phase



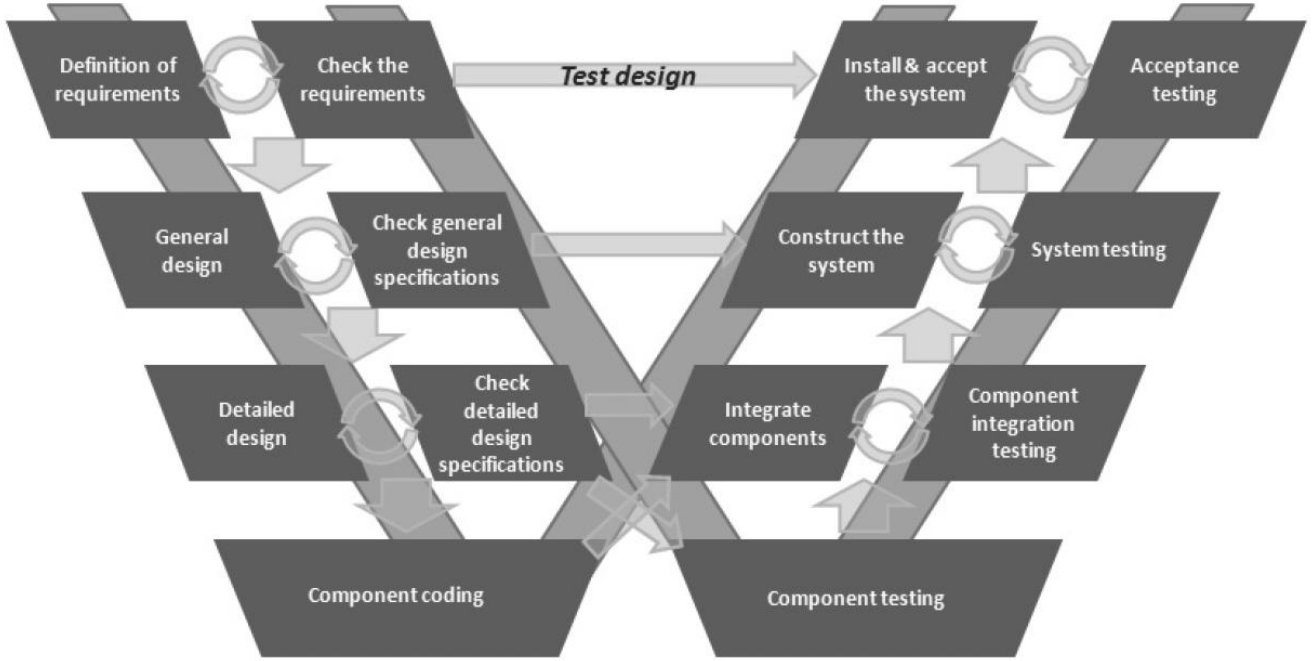
The V model

- Testing for each sequential stages
- Test design at the same time with the development stage



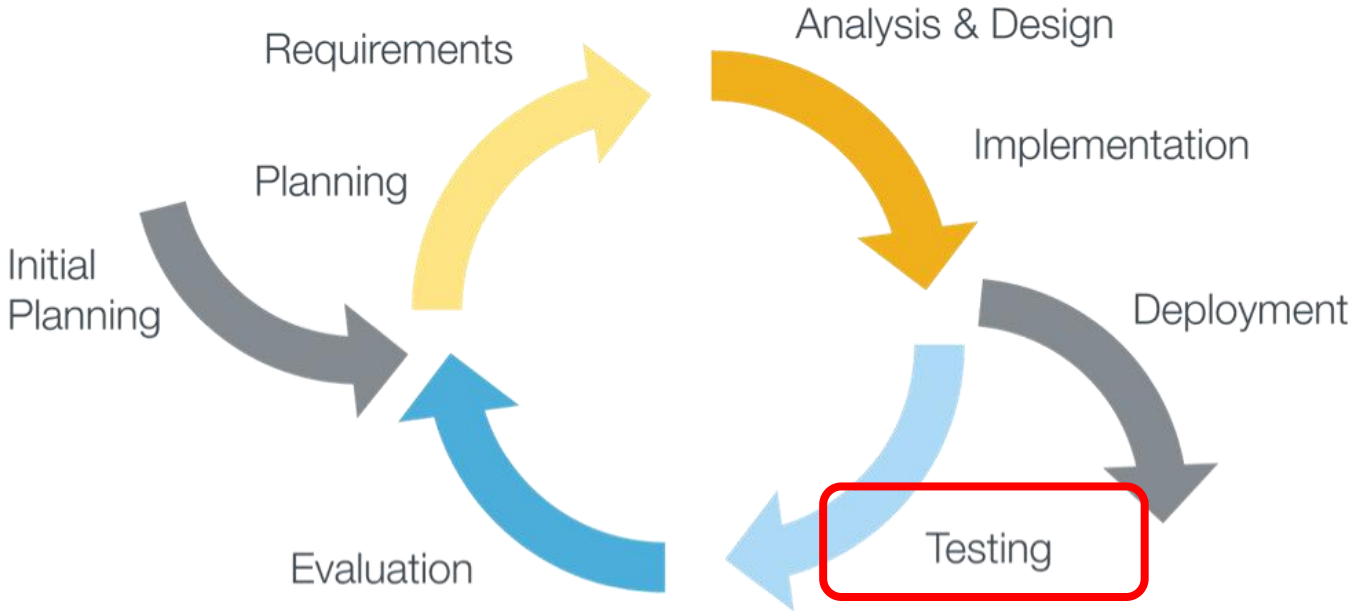
The W/VVV model

- Incorporating static testing techniques
- Early V&V



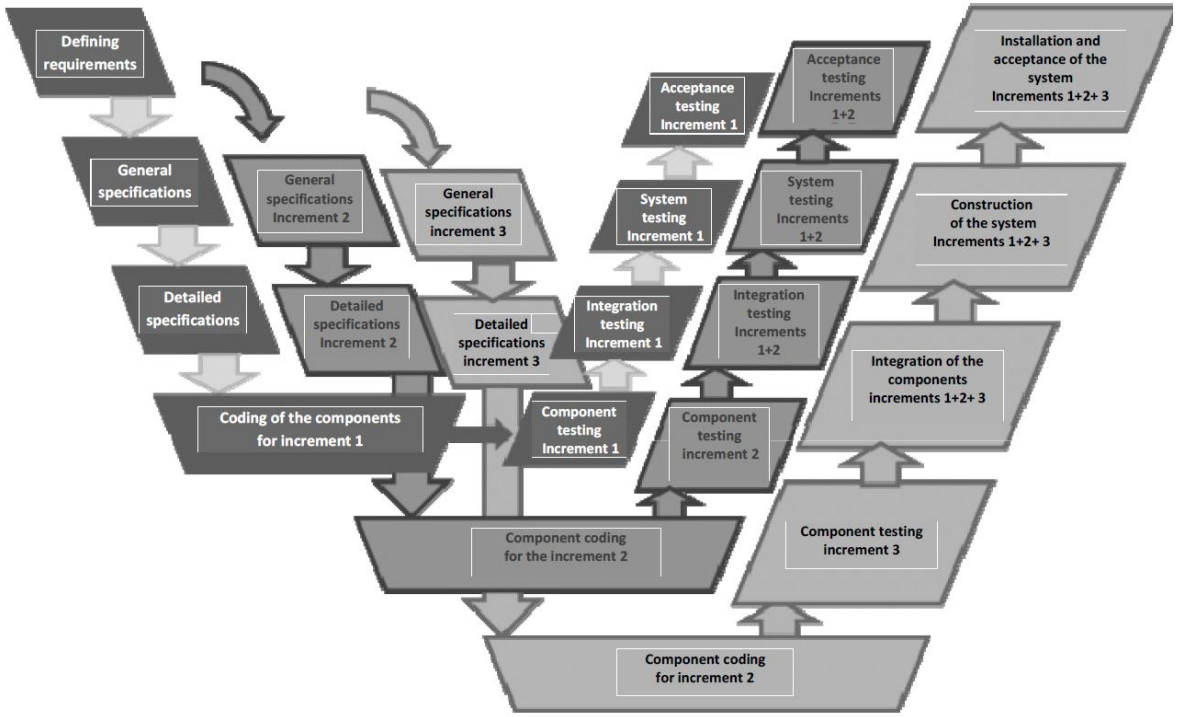
Iterative software development model

- Develop core functionalities first
- Improve/refine software in later iterations
- Problem: later iterations may damage previous iterations



Incremental model

- Increments expected before design
- Increments defined during design
- Cons: spend too much time developing an increment if system is not divided properly



After a defect has been corrected

- Confirmation tests or retests
  - Verifying that the defect has been corrected and the software operates as expected
- Regression tests
  - Make sure that the correction did not introduce any side effects (regression) on the rest of the software

Regression test example

- Changes in node 22
- Direct impact: 1, 23, 24, 25, 26, and 27
- Indirect impact: 14, 15, 6, 8, 2, 3, 4, and 13