QTronic
VIRTUAL ECUs

VOLVO

CONTINUOUS INTEGRATION AND TEST

FROM MODULE LEVEL TO VIRTUAL SYSTEM LEVEL

A WAY OF WORKING

- At Volvo Car Corporation, Propulsion division, we develop our own algorithms
- These algorithms control the engine and the gearbox
  - Equations of physical systems are expressed in Python
  - These equations or functions are connected into a network
  - A python package is used to optimize the equation parameters to match a measured reality.
  - When the result is robust we implement these equations in TargetLink and C-code is generated that is then run in the ECU
  - On-line ODE solvers for state estimations (Euler is unstable)are often used
  - IMC feedback (e.g. lambda control) is used in the Air Charge system

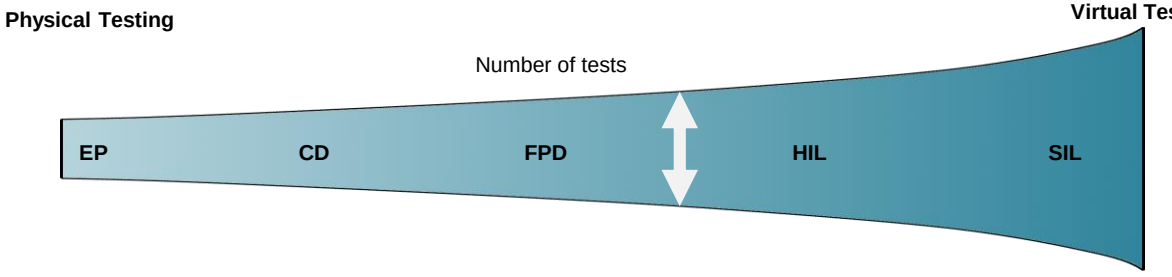
python

git

QTronic
VIRTUAL ECUs

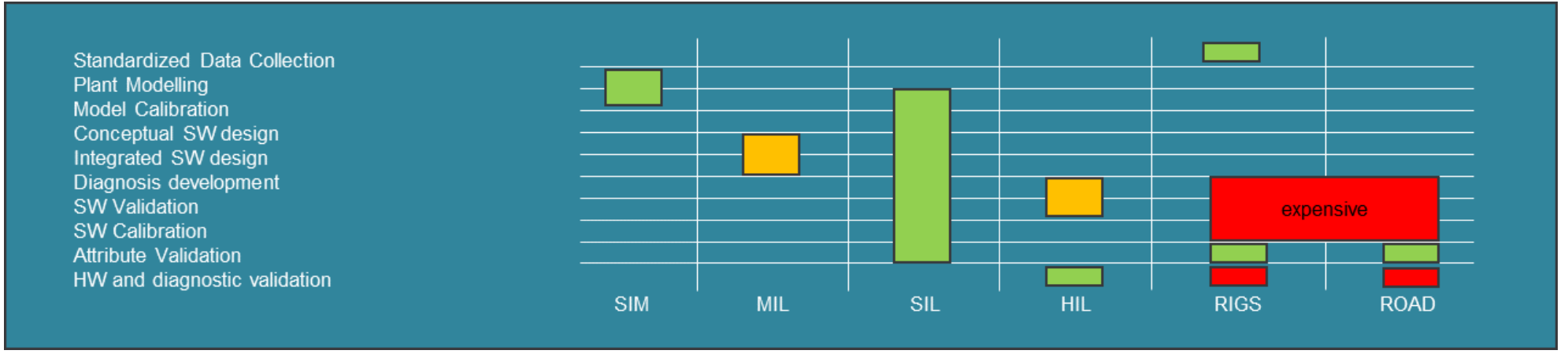
THE BIGGER PICTURE

SIL removes bottlenecks



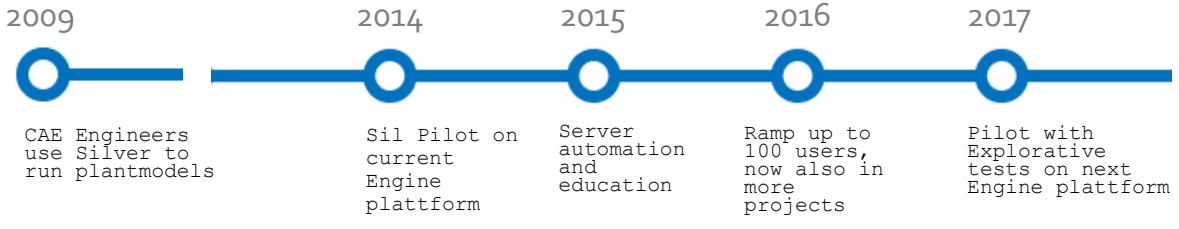
- Physical test cells are limited in numbers due to HW and Facilities
- Virtual test cells are limited by number of SW licenses, only
- SIL offers a vehicle like integration environment
- SIL provides faster execution time than MIL
- SIL can in many cases replace HIL debugging
- SIL serves as a virtual test environment

Unified testbench for SW design

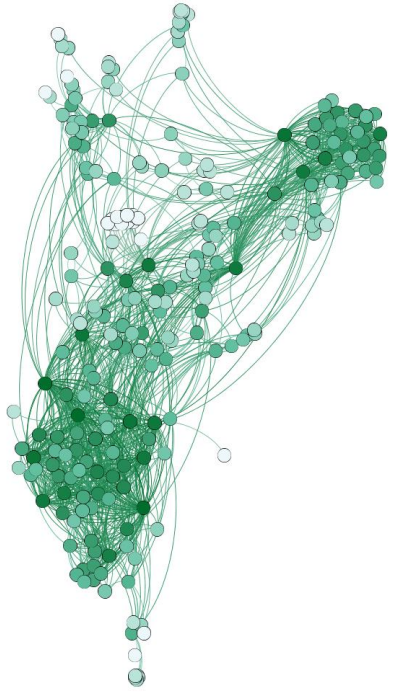


HISTORY

- Prior to the current engine generation, all tests done in car
- During the development of the current engine generation, automatic unit and system tests were introduced.
  - Aftertreatment SW solely developed in Sil platform.
  - One senior SW developer said: *now I know it will work when we test in the car...*

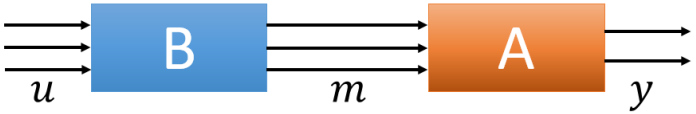


UNIT TESTING AN ENGINE CONTROL MODULE



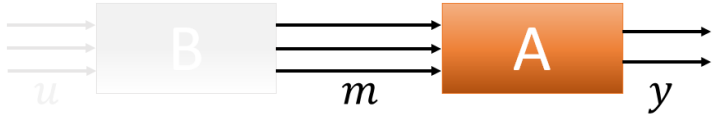
- The Software in current generation ECMs is structured into around 250 modules
- Unit tests are required to ensure software quality and compliance with industry norms (e.g. ISO 26262)

CODE INSTRUMENTATION: MOTIVATION



$$y(t) = A\left(B\left(u(t)\right)\right)$$

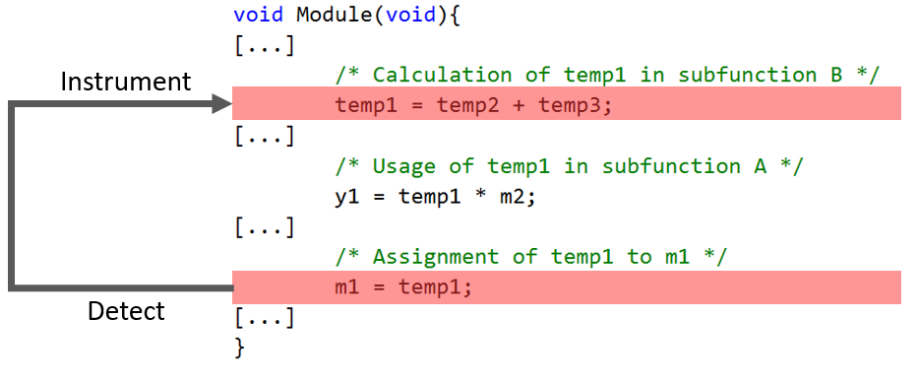
- Functions in the ECU are made up of smaller units during the design phase
- These Subfunctions are not represented in generated code



$$y(t) = A\left(m(t)\right)$$

- Unit tests need to be applicable to subfunctions
- Values need to be injected into the input vector of each subfunction

CODE INSTRUMENTATION: SIGNAL ALIASES



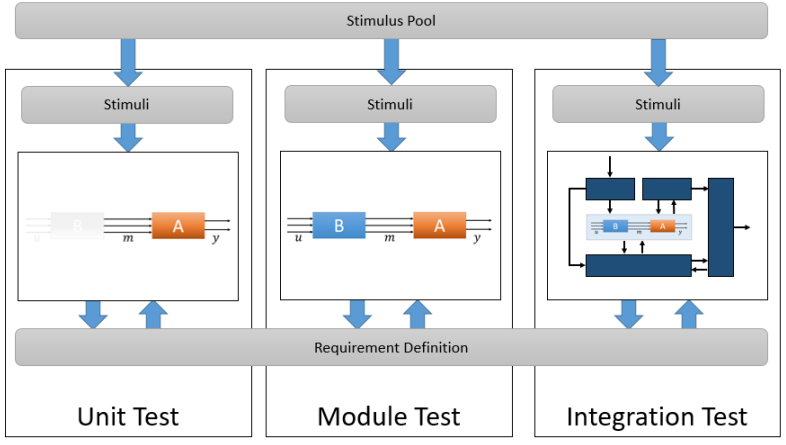
- Code generators tend to use temporary, local variables
- Analysis is needed to determine cases where such a temporary variable is always equal to a measurable signal
- Value injecting needs to take into account all aliases of a given input

REQUIREMENT BASED TEST (MOTIVATION)

- Traditional Unit Tests rely on a small amount of scripted scenarios
- Every requirement is associated with a fixed set of stimuli that should result in the wanted behavior
- In large systems in a continuous environment it is hard to achieve a sufficient test coverage

- To solve this problem, tests are defined as **Requirement Watchers** which are decoupled from the stimulus

REQUIREMENT BASED TEST (WATCHERS)



- Requirement watchers are defined through preconditions and expected behavior

WHENEVER	Pressure Cntrl is Active
EXPECT	Desired Pressure
WITHIN	10s

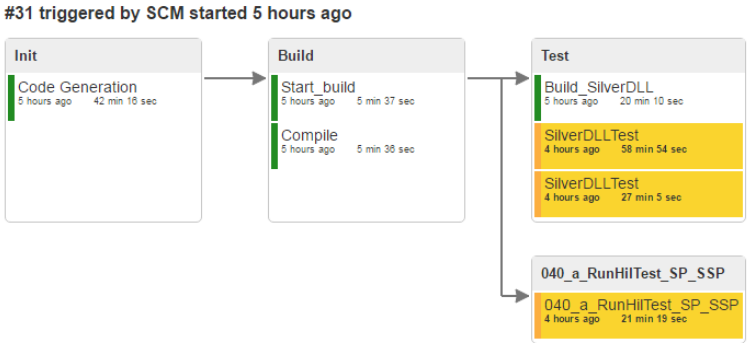
- All requirements can be monitored at all times
- Watchers are independent of the scope of the test

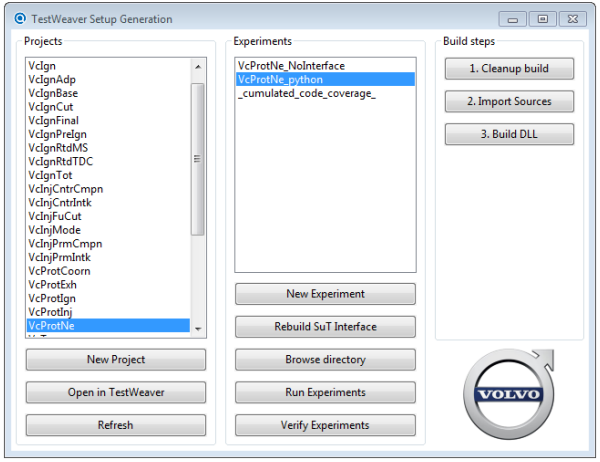
large number of stimuli X large number of watchers

=> ensure high test coverage

SIL SOFTWARE PROCESS

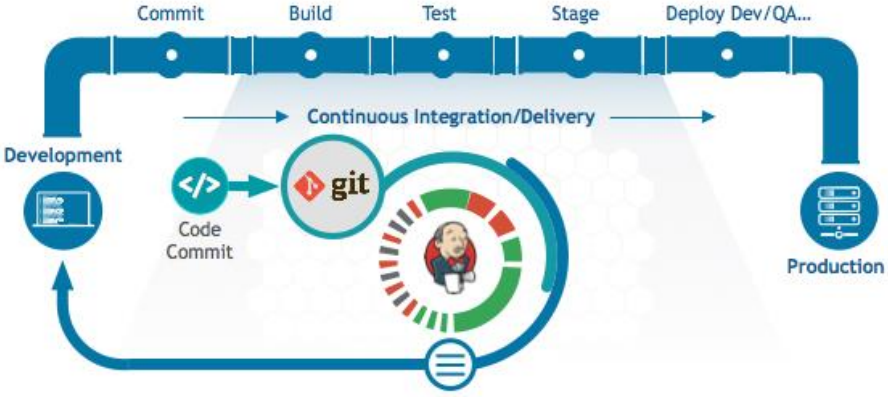
- A DLL of our SW components is created
- A text file specifies the System Under Test, can be the whole SW, a set of function, one function or a subsystem of a model
- Test cases are extracted from the developers Silver database

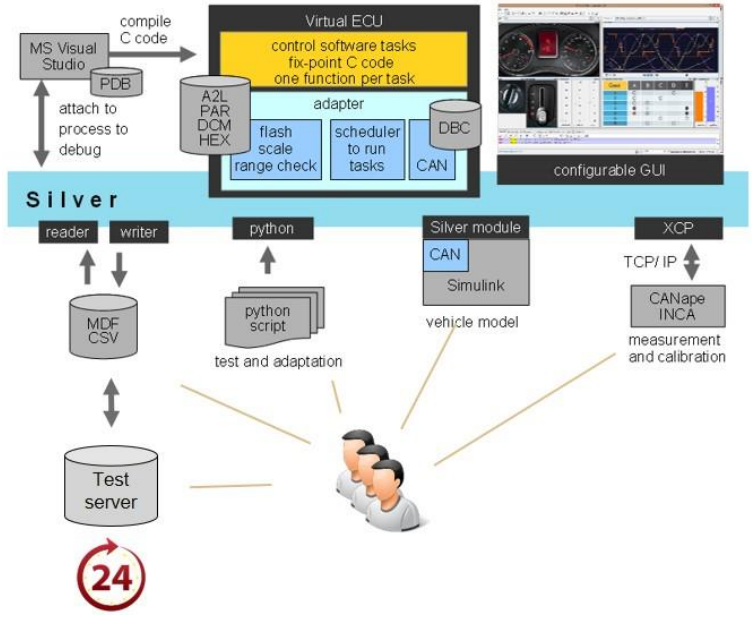




SIL SOFTWARE PROCESS

- Test cases are expressed in Python
- Run TestWeaver Light on a Jenkins test server as a nightly build
- Code coverage is measured by using CTC++ from Verifysoft and reported as HTML





CONCLUSIONS

- The presented method of **code instrumentation** provides a way to design tests and requirements for arbitrarily small units **inside a large system**
- By ensuring reusability and independence of requirement definitions **- watchers,** they can be applied to many test – **stimuli**
- The **cross product** of many watchers and many stimuli ensures **high test coverage**
- A high test coverage is essential in order to **guarantee high quality software**

WAY FORWARD

- We are currently expanding our range of plant models, so that the developers can use closed loop simulations.
- We have started with Explorative Tests, where the stimuli is automatically generated.
- We are adding more Additional ECUs