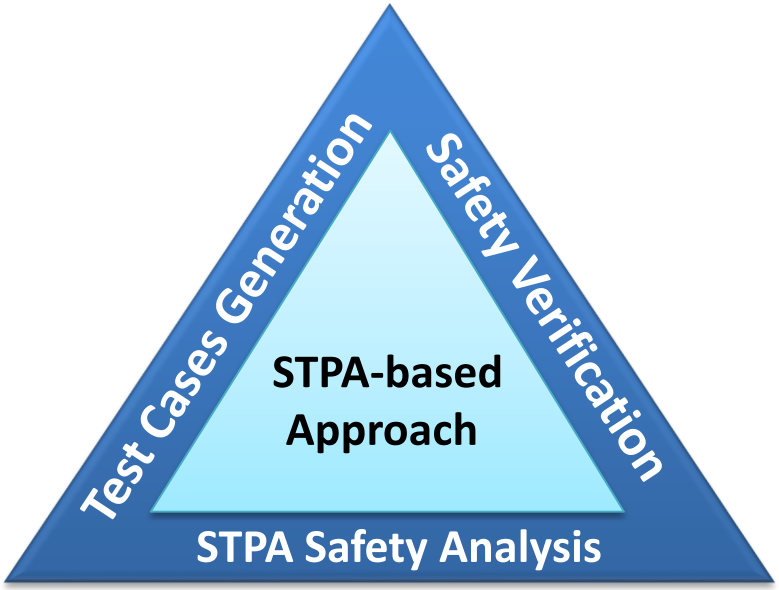
www.uni-stuttgart.de

A Comprehensive Safety Engineering Approach for Software Intensive Systems based on STPA



Asim Abdulkhaleq, Ph.D Candidate
Institute of Software Technology
University of Stuttgart, Germany

Joint with:
Prof. Dr. Stefan Wagner
Prof. Dr. Nancy Leveson

3rd ESW2015, 5th October, Amsterdam, Netherlands

Motivation: Software of Today's Complex Systems

- **Today's safety critical systems are increasingly reliant on software.**
  - Software is the most complex part of modern safety critical embedded systems.
  - E.g. A modern BMW 7 car has something close to 100 million lines of software code in it, running on 70 to 100 microprocessors (Prof. Manfred Broy, TU München)

Electronic Stability Control
Traction Control
Back Camera
Stop & Go Adaptive Cruise Control
Reverse Backup Sensors
Anti-Lock Braking Systems (ABS)
Modern Vehicle
Tire Pressure Monitoring
Electronic Brakeforce Distribution Systems
Adaptive Cruise Control (ACC)
Automatic Braking Systems
Automatic Braking Systems

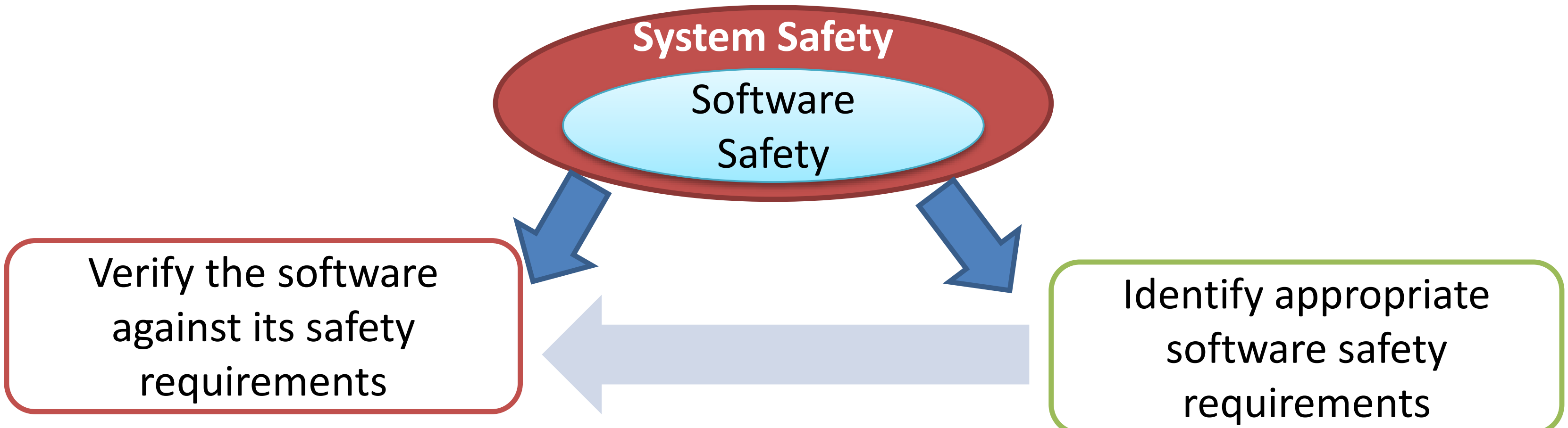
How to develop a safe software (or achieve an acceptable level of safety of software)?

Agenda

- Motivation ✔
- Introduction ◎
  - Problem Statement
  - Research Objectives
  - Contribution
- A Comprehensive Safety Engineering Approach based on STPA
- Illustrative Example: Adaptive Cruise Control System
- Conclusion & Future Work

Problem Statement

- **Problem Statement**
  - Safety is a system property and needs to be analysed in a system context.
  - As software is a part of system, **software safety** must be considered in the context of the system level to ensure the whole system's safety.



- **Software Verification approaches:**
  - **Model checking (SMV, SPIN, .etc.)**
  - **Testing approaches**

❌ Functional correctness of software, however, even perfectly correct software can contribute in an accident.
❌ Not directly concern safety
❌ Some limited in practices
❌ Achieving 100% testing is impossible.

- **Safety Analysis Techniques:**
  - FTA, FMEA, **STPA**

❌ FTA and FMEA have limitations to cope with complex systems. STPA is developed to cope with complex systems, but its subject is system not software
❌ STPA is performed separately
❌ STPA is not Placed into software development process

Research Objectives & Contribution

Research Objectives

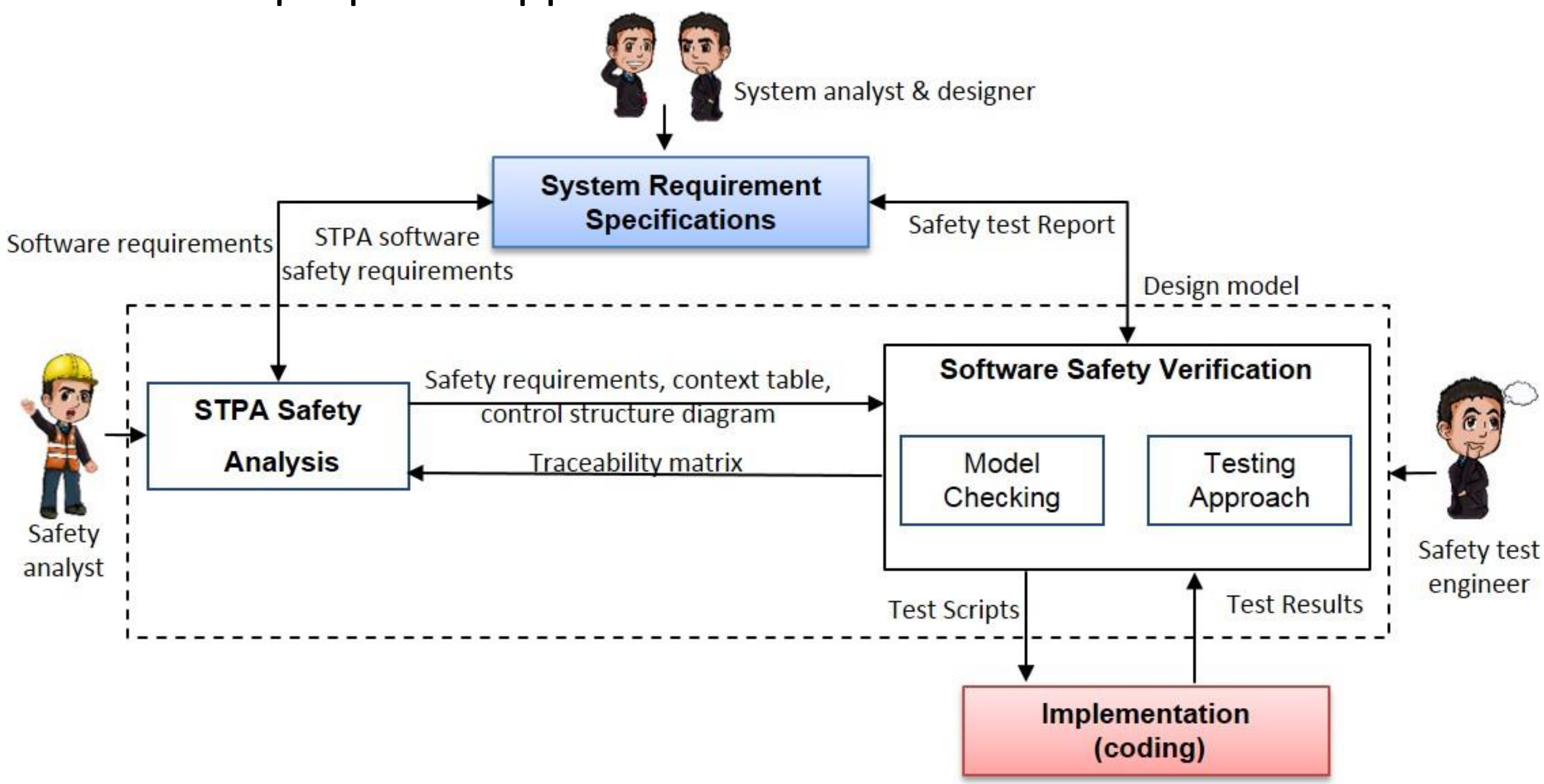
- Integrate STPA safety activities in a software engineering process to allow safety and software engineers a seamless safety analysis and verification.
- This will help them to derive software safety requirements, verify them, generate safety-based test case and execute them to recognize the associated software risks.

Contribution

We contribute a safety engineering approach to derive software safety requirements at the system level and verify them at the design and implementation levels.

A comprehensive Software Engineering based on STPA

- Overview of the proposed approach:

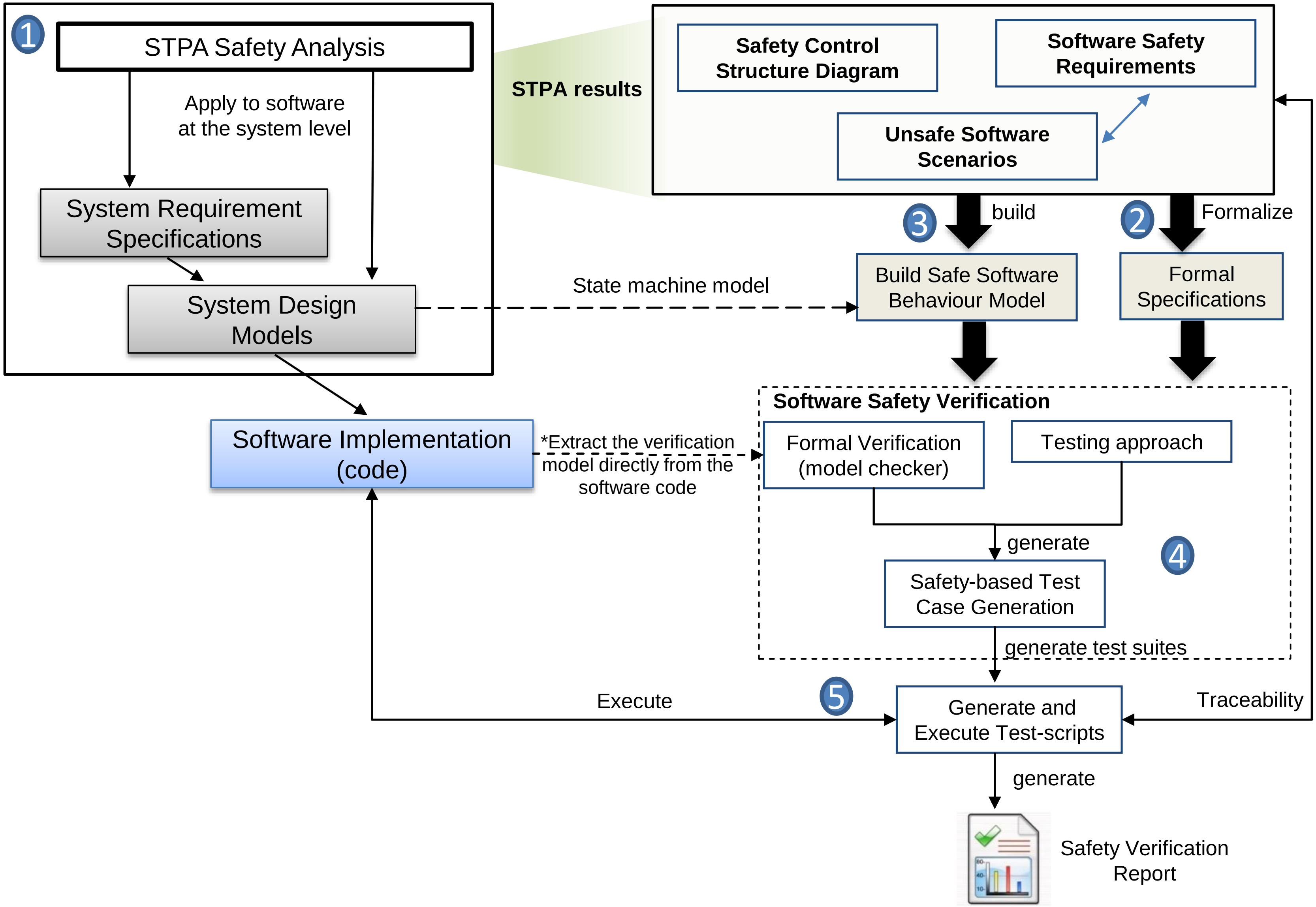


- Four main activities & roles

1. Deriving software safety Requirements at the system level ➡ **Safety Analyst**
2. Constructing the safe behaviour model of the software controller ➡ **Safety Analyst & System Designer**
3. Verifying the safe behaviour model against the STPA results ➡ **Test Engineer**
4. Generating & executing the safety-based test cases based on STPA results ➡ **Safety Analyst & Test Engineer**

Detailed View of the Proposed Approach

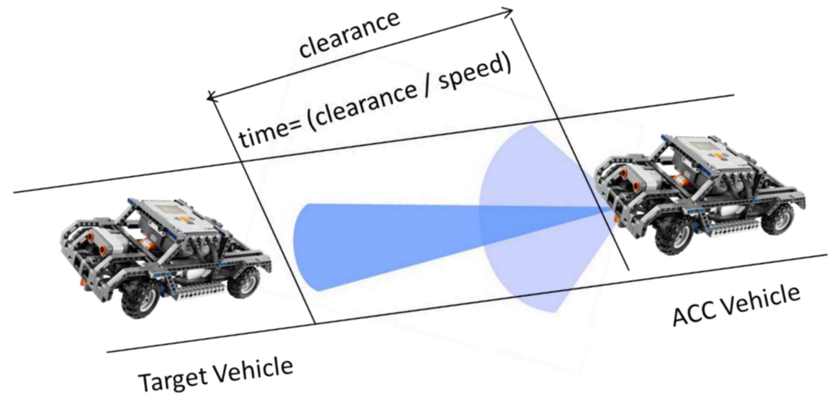
- The proposed approach can be applied during developing a new safe software or on existing software of safety-critical system



Example: Adaptive Cruise Control System

- **Adaptive Cruise Control System:** is a well-known automotive system which has strong safety requirements. ACC adapts the vehicle's speed to traffic environment based on a long range forward-radar sensor which is attached to the front of vehicle.

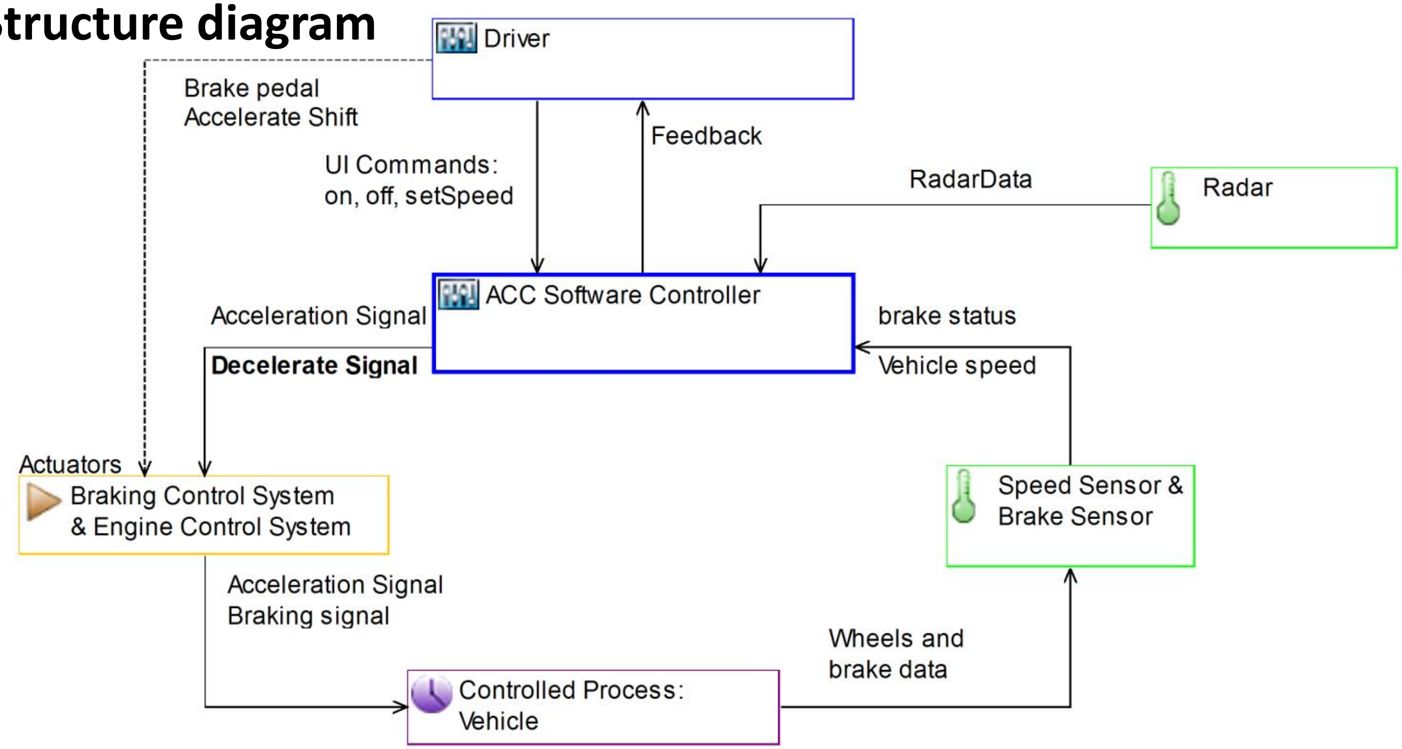
How to derive the safety requirements of ACC software controller at the system level and generate the safety-based test cases?



- **Fundamentals of Analysis**
  - **System-Level Accidents:**
    - ACC-1 : ACC vehicle crashes with front vehicle while ACC status is active.
  - **System-Level Hazards**
    - H-1: ACC software does not maintain safe distance from front vehicle.

Step1.a : Deriving the software Safety Requirements

- **Control Structure diagram**

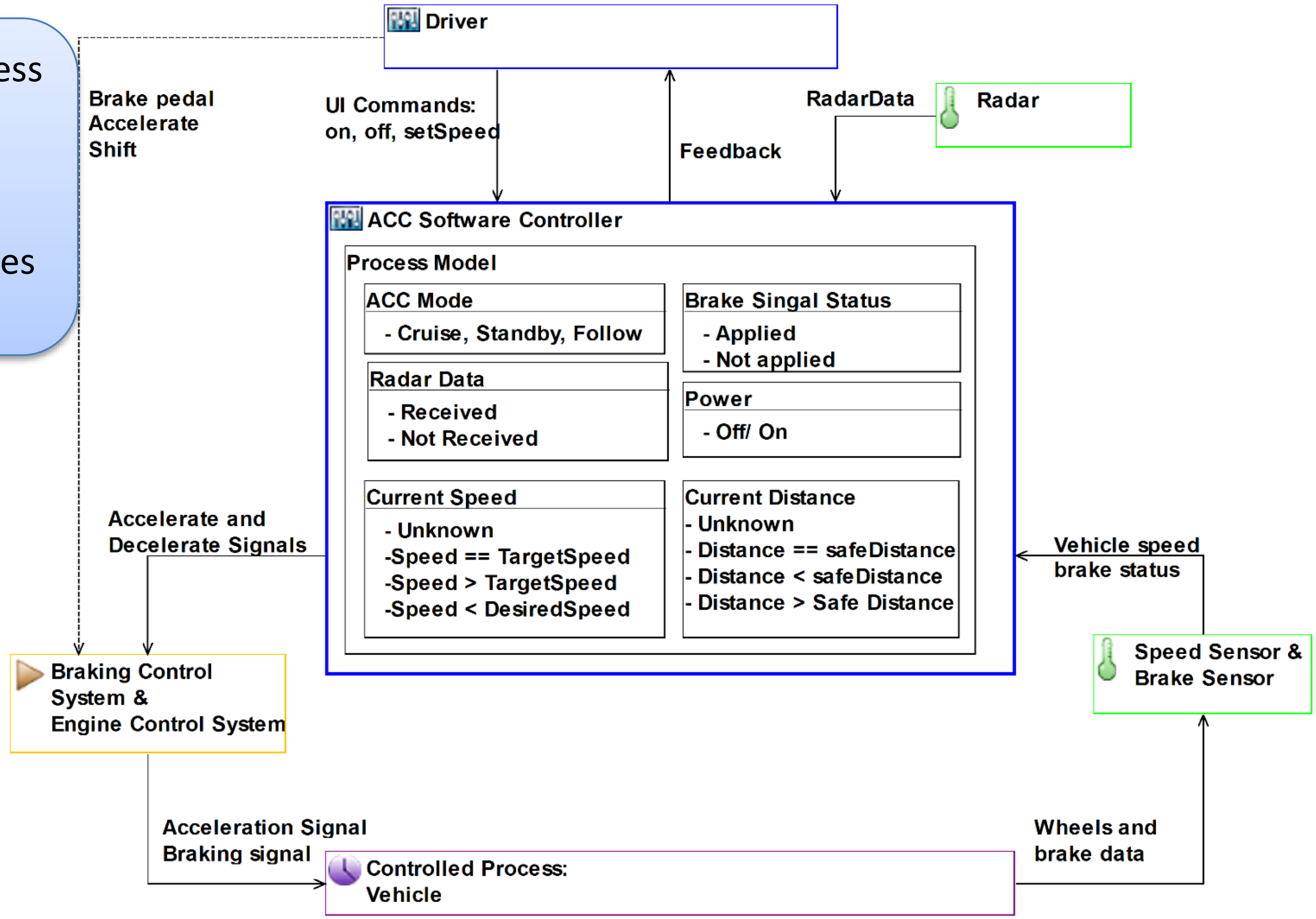


	Software Safety Requirements
SSR1.1	The ACC software controller should provide an acceleration signal when the target vehicle is no longer in the lane
SSR.1.2	The ACC software controller decelerates the speed when the distance to the target vehicle is too close.
SSR1.3	The ACC software controller should not provide the acceleration signal speed when a safe distance is reached

Step 1.b: Identify Unsafe Scenarios of Software

- **Control Structure Diagram & process model :** shows the main interconnecting components of the ACC system at a high level.

Three types of process model variables:
(1) Internal variables
(2) Interaction variables
(3) Environmental variables



The total number of all variables combination: 3 x 2 x 2 x 2 x 4 x 4 = 384.

Extended Approach to STPA

- **Extended Approach to STPA :** John Thomas proposed an extended approach to STPA.
  - It aims to refine the identified unsafe control actions in the STPA Step 1 based on the combination of process model variables.

Control Action	Process Model Variable 1	Process Model Variable 2	Process Model Variable 3	Hazardous?

- **Limitations for complex software controllers:**
  - The difficulty is in defining the combination for large number of values of the process model variables which have affect on the safety of the control actions.
  - Considering all combinations involves more effort and time.

I proposed to use the principle of t-way combinatorial testing algorithm

How to automatically generate the combinations and minimize the number of combination of large complex system ?

CIT is testing technique that requires covering all t-sized tuples of values out of n parameter attributes of a system under test.

Step1 : Automatically Generating Context Tables

- **Apply the combinatorial testing algorithm to reduce the number of combination between the process model variables (Cooperation with Rick Kuhn, National Institute of Standards and Technology, Computer Security Division, US).**

Test Case#	followDistance	cruiseSpeed	BrakePedal	ACCMode
0	current distance < safe distance	current speed ==desired speed	Not applied	Follow
1	current distance < safe distance	current speed < desired speed	applied	Standby
2	current distance < safe distance	current speed > desired speed	Not applied	Cruise
3	current distance < safe distance	Unknown	applied	Follow
4	current distance > safe distance	current speed ==desired speed	Not applied	Standby
5	current distance > safe distance	current speed < desired speed	applied	Cruise
6	current distance > safe distance	current speed > desired speed	applied	Follow
7	current distance > safe distance	Unknown	Not applied	Standby
8	current distance <=safe distance	current speed ==desired speed	applied	Cruise
9	current distance <=safe distance	current speed < desired speed	Not applied	Follow
10	current distance <=safe distance	current speed > desired speed	Not applied	Standby
11	current distance <=safe distance	Unknown	Not applied	Cruise
12	Unknown	current speed ==desired speed	applied	Follow
13	Unknown	current speed < desired speed	Not applied	Standby
14	Unknown	current speed > desired speed	applied	Cruise
15	Unknown	Unknown	Not applied	Standby

- **By combinatorial testing algorithm:**
  - **We can automatically generate the context table.**
  - **We can achieve different combination coverages (e.g. pairwise coverage = 16 combinations, 3-way coverage = 48 combinations)**
  - **We can apply different roles and constraints to the combination to ignore some values**

Examples of the Context Table

◆ **ACC software controller provides a safety critical action: accelerate signal**

Control actions	Process Model variables				Hazardous
	Distance	Speed	Brake	ACC Mode	
Accelerate Signal	< safe distance	== desired speed	Applied	Cruise	No
	< safe distance	>desired speed*	Notapplied	Cruise	Yes (H2, SSR3-4)
	< safe distance	> Desired speed	Notapplied	follow	Yes (H1, SSR1)

Refine the software safety Requirements

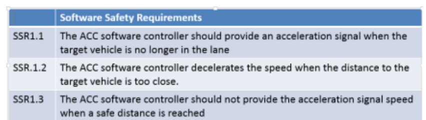
$SSR_{1.3}$: ACC should not provide accelerated signal when the distance is less or equal the safe distance while ACC in cruise mode and brake pedal is not pressed.

Generate LTL formula

$LTL_{1.3}$: G(distance <= safe distance && ACCController == cruise && brakepedal != Pressed) → !(accelerationSignal)

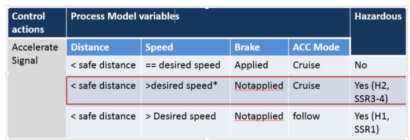
Step 2 : Constructing the safe behaviour model of software controller

- **To verify the design & implementation of software controller against the STPA results and generate the safety-based test cases:**
  - **Each software controller must be modelled in a suitable behaviour model**
  - **The model should be constrained by STPA safety requirements**



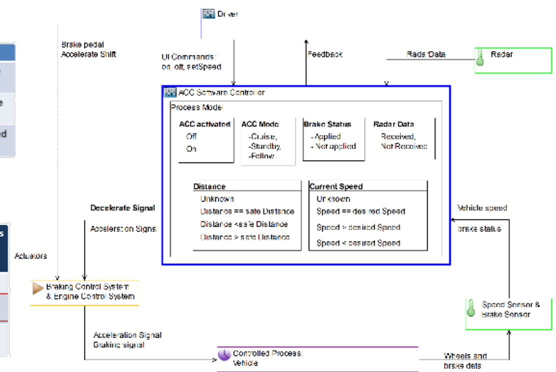
	Software Safety Requirements
SSR1.1	The ACC software controller should provide an acceleration signal when the target vehicle is no longer in the lane
SSR.1.2	The ACC software controller decelerates the speed when the distance to the target vehicle is too close.
SSR1.3	The ACC software controller should not provide the acceleration signal speed when a safe distance is reached

Software safety requirements



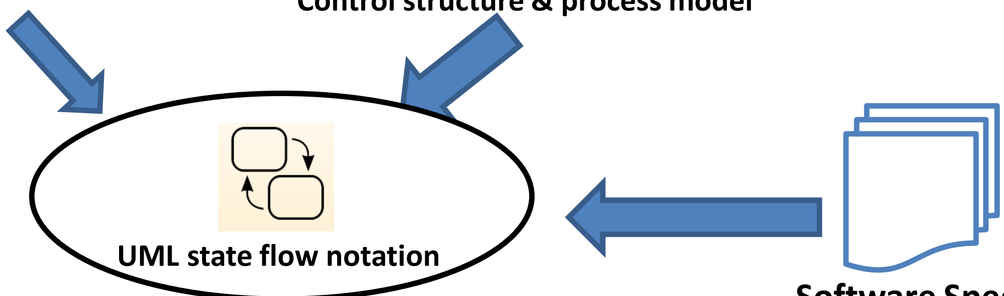
Control actions	Process Model variables				Hazardous
	Distance	Speed	Brake	ACC Mode	
Accelerate Signal	< safe distance	== desired speed	Applied	Cruise	No
	< safe distance	>desired speed*	Notapplied	Cruise	Yes (H2, SSR3-4)
	< safe distance	> Desired speed	Notapplied	follow	Yes (H1, SSR1)

Context tables



Control structure & process model

STPA Results



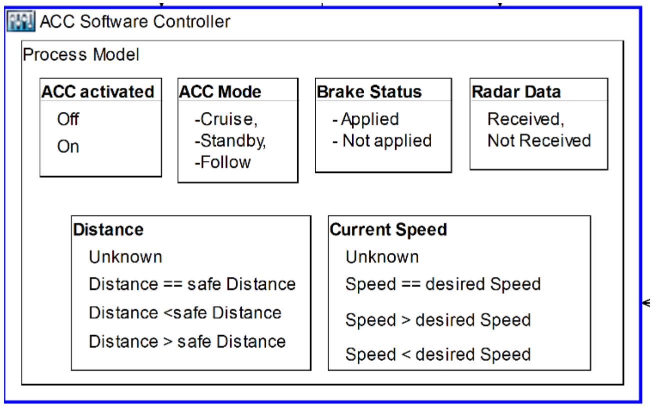
Software Specification

A safe behaviour model of software controller

- **Syntax of each transition of the safe behaiovur model:**

Step 2 : The safe behaviour model of ACC software controller

STPA Results



Software Controller & process model variables

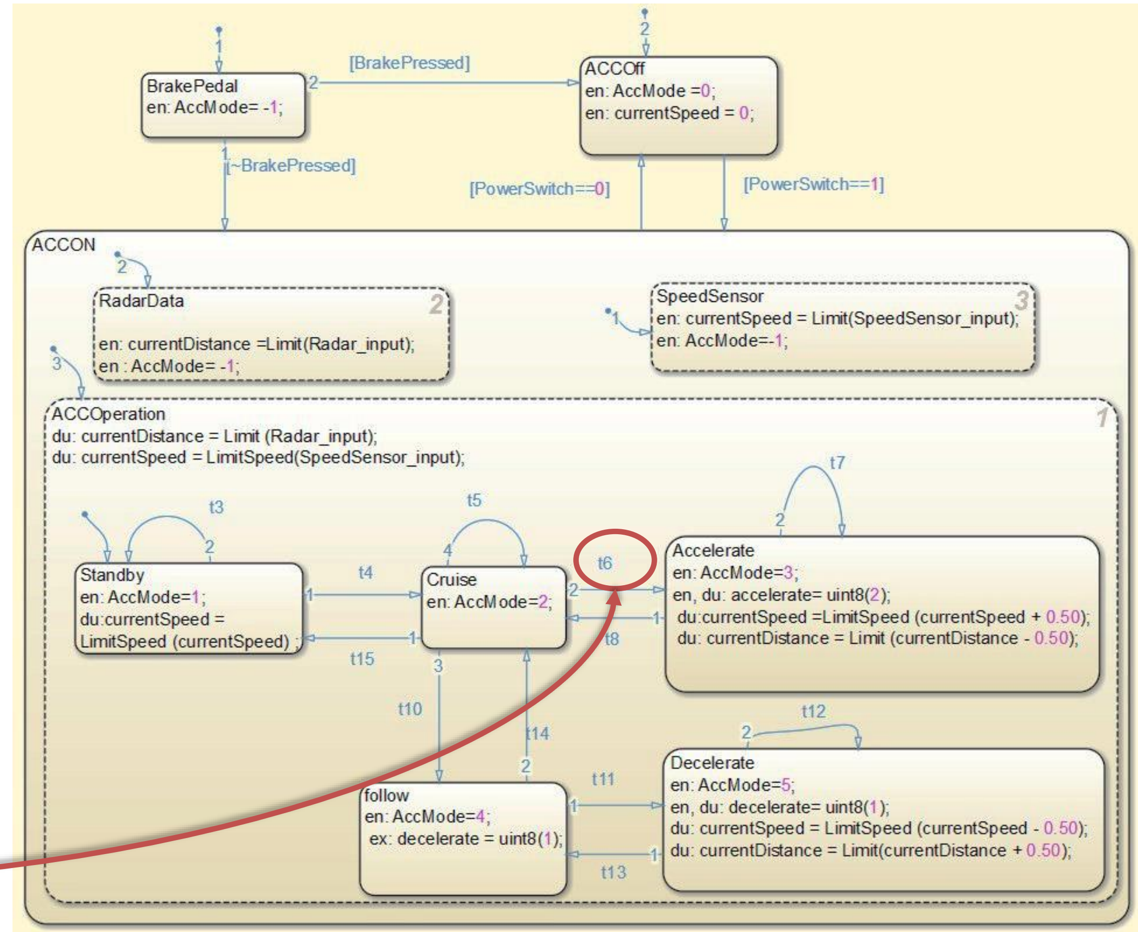
Control actions	Process Model variables				Hazardous
	Distance	Speed	Brake	ACC Mode	
Accelerate Signal	< safe distance	== desired speed	Applied	Cruise	No
	< safe distance	>desired speed*	Notapplied	Cruise	Yes (H2, SSR3-4)
	< safe distance	> Desired speed	Notapplied	follow	Yes (H1, SSR1)

Context Table

Transition t6: (safety requirement)

[currentSpeed < desiredSpeed && currentDistance > safeDistance && ! BrakePressed & ACCMode == Cruise]

A safe behaviour model of ACC software Controller

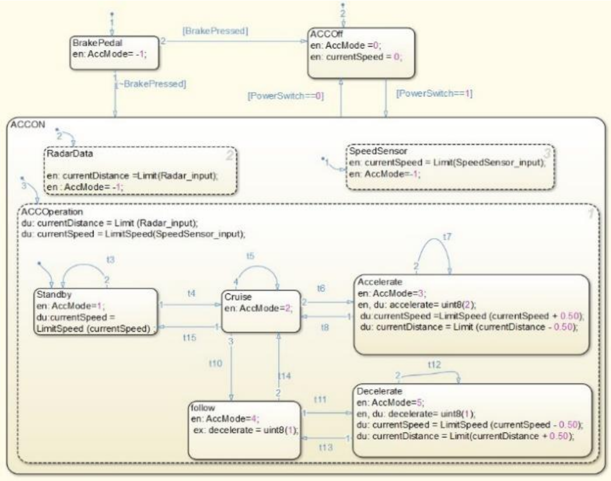


Parallel Process variables

Sequential Process variables

Step 3.1 : Verification of Safe behaviour model

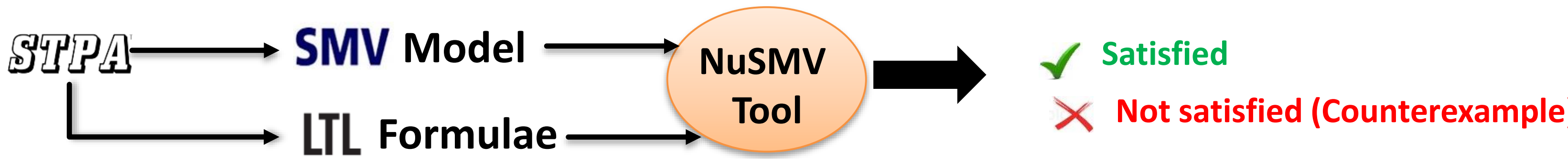
- **To ensure that the safe behaviour model satisfy the STPA safety requirements, We convert the model into a input language of model checker such as SMV (Symbolic Model Verifier) model**




```
MODULE main ()
VAR
RadarData :{unknown, received}
BrakePedal :{notPressed, pressed}
ACC_Activated: {on, off}
ACCMode:{standby, cruise, follow}
Control_actions :{accelerate, decelerate}
ACC_Controller:{radardata, ACCMode,speedData}
controlaction: {toCruise, toaccelerate, todecelerate, tofollow, tosetSpeed}
currentspeed : {0, 25, 45, 65, 100}
dersiredspeed: {25,45, 75, 200}
safedistance: {65}
Ignited  : boolean;
…
init(ACController) := initial;
init (event) :=default;
next(ACController) := case
   ACCController=Off  & (Ignited=off): Off;
   ACCController=Off  & (Ignited=on & BrakePressed=NotApplied): Initial;
   ACCController=Initial  & (Ignited =off | BrakePressed=Applied): Off;
   ACCController=Initial  & (BrakePressed=NotApplied &(CurrentSpeed<25): Standby;
```

Step 3.1 : Verification of Safe behaviour against STPA SSR

- **We ran the NuSMV 2.5.3 model checking tool on a Windows 7 PC, i7 CPU with 2.80 GHZ, 8 GB main memory.**

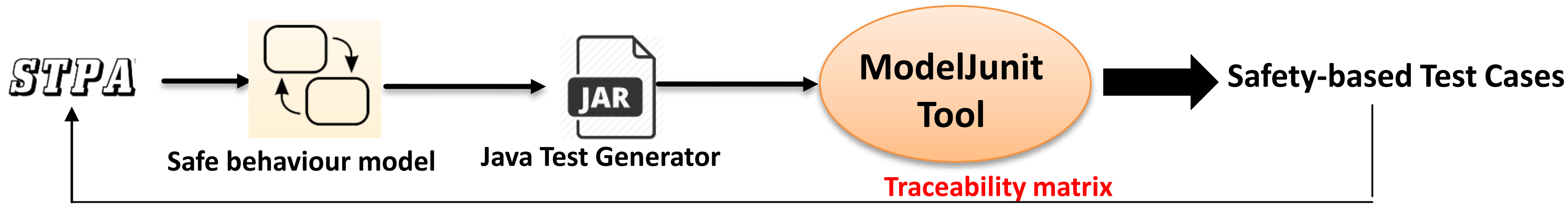


  - The NuSMV tool verified the SMV model against the LTL formulae
  - The SMV model satisfied all the identified STPA software safety requirements and no counterexample generated (itself is built using STPA results)

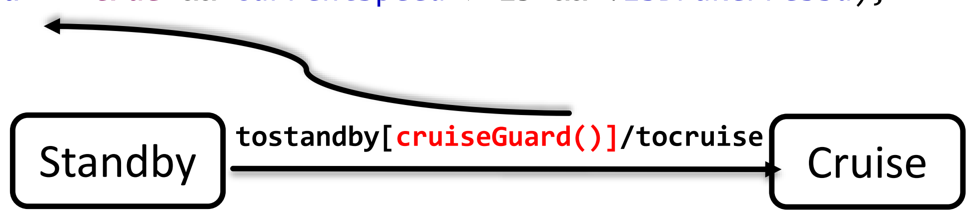
```
WARNING: single-value variable 'TargetSpeed' has been stored as a constant
WARNING: single-value variable 'SafeDistance' has been stored as a constant
-- specification  G (CurrentSpeed > 25 ->  X ACCController = Cruise)  is true
-- specification  G ((CurrentSpeed = TargetSpeed & CurrentSpeed > 25) ->  X ACCC
ontroller = Cruise)  is true
-- specification  G ((((CurrentSpeed < TargetSpeed & CurrentDistance > SafeDista
nce) & CurrentSpeed > 25) & (BrakePressed = NotApplied & ACCController = Cruise)
) ->  X ACCController = Accelerate)  is true
-- specification  G ((((CurrentSpeed < TargetSpeed & CurrentDistance > SafeDista
nce) & CurrentSpeed < 25) & (BrakePressed = NotApplied & ACCController = Cruise)
) -> !(ACCController = Accelerate))  is true
-- specification  G (ACCController = Accelerate ->  X (((CurrentSpeed < TargetSp
eed & CurrentDistance > SafeDistance) & CurrentSpeed > 25) & (BrakePressed = Not
Applied & ACCController = Cruise)))  is true
-- specification  G (ACCController = Decelerate -> (((CurrentSpeed < TargetSpeed
 & CurrentDistance > SafeDistance) & CurrentSpeed > 25) & (BrakePressed = NotApp
lied & ACCController = Follow)))  is true
```

Step 3.2 : Safety-based Test Cases Generating & Execution

- **To generate safety-based test cases based on STPA results,**
  - We build a Java test generator based on the safe behaviour model.
  - We use the Java test generator as input to the model-based testing tool e.g ModelJUnit.

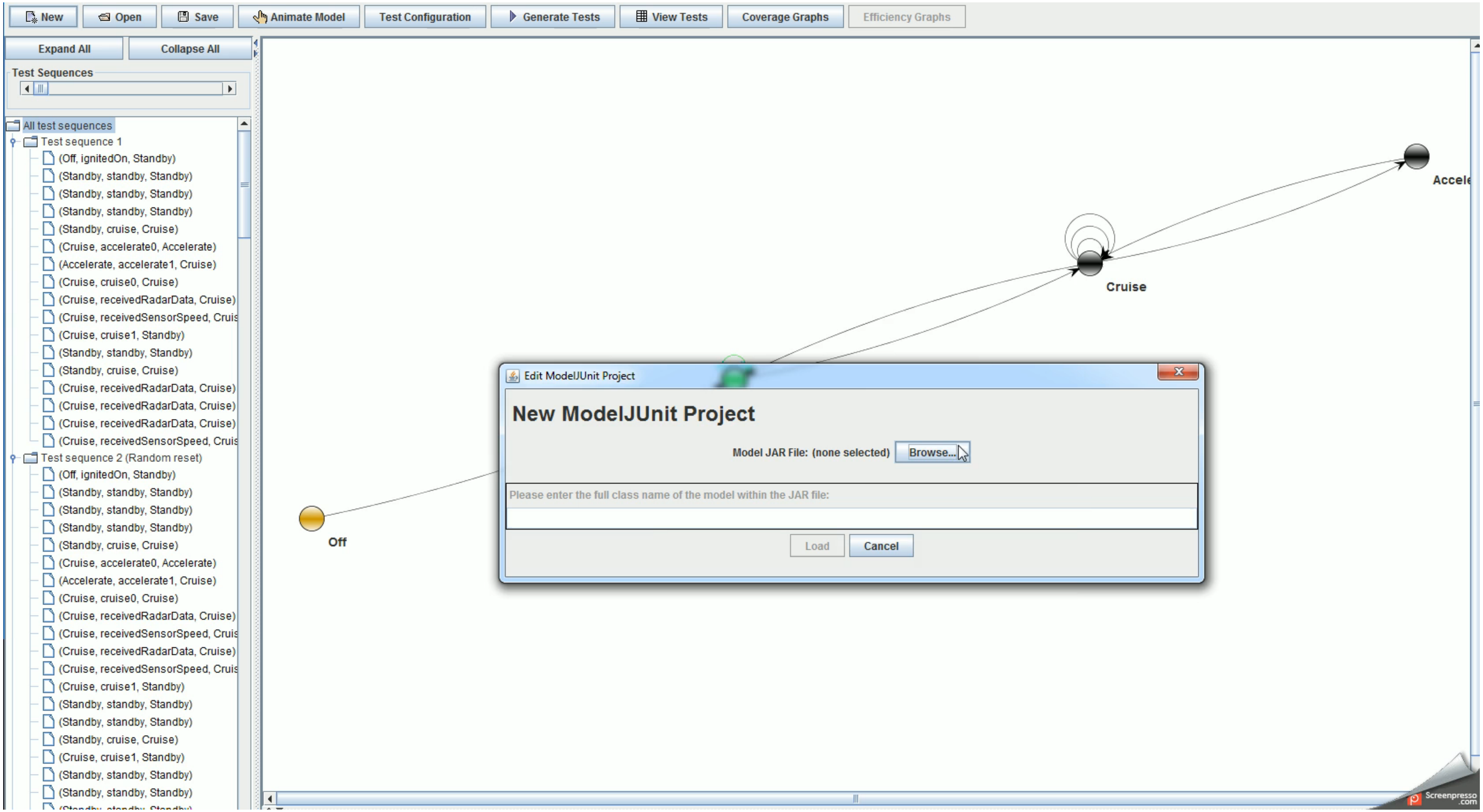



```java
public class ACC_TestCodeGenerator implements FsmModel {
…
public boolean cruiseGuard() {
   return (currentState == State.Standby && ignited == true && currentSpeed > 25 && !isBrakePressd);
}

public @Action void tocruise() {
   printTestInputData();
   currentState = State.Cruise; accelerating();
  if (isBrakePressd)
  isBrakePressd = false;
}

public boolean standbyGuard() {
return (currentState == State.Standby && ignited == true && currentSpeed < 25 && !isBrakePressd);
}

public @Action void tostandby() {
  printTestInputData();
 currentState = State.Standby;
  move();
```



The Results of Test Cases Generating & Demo

- **We generated automatically 487 test cases which cover the safe behaviour of the ACC software controller with the action coverage =15/18, state coverage =6/6, transition coverage =15/15, and the pair transition coverage 36/36.**



- **We created a Java code to generate the traceability matrix between the generated test cases and STPA results and export them as an Excel sheet.**

Syntax of transition = Event[STPA safety requirement]/Control Action

- We calculated the coverage of STPA software safety requirements

$$\text{\#Coverage(SSRs)} = \frac{\#Total\ number\ of\ STPA\ safety\ requiremnts\ covered\ into\ test\ cases}{\#Total\ number\ of\ STPA\ safety\ requriements}$$

$$\text{\#Coverage(SSRs)} = \frac{21}{21} = 100\ \%$$

- We calculated the average of each STPA software safety requirement and each control action of each software controller

$$\text{\#Average(SSR)} = \frac{Total\ number\ of\ test\ cases\ which\ conatin\ SSR}{Total\ number\ of\ test\ cases}$$

$$\text{\#Average(CA)} = \frac{Total\ number\ of\ test\ cases\ which\ conatin\ CA}{Total\ number\ of\ test\ cases}$$

- For example: The average of the software safety requirement (SSR1.3) and control action “providing accelerate signal” are:

$$\text{\#Average(SSR1.3)} = \frac{17}{197} = \sim 10\% \qquad \text{\#Average(CA1)} = \frac{21}{197} = \sim 11\%$$

Conclusion & Future Work

- **Conclusion:**
  - We presented a safety engineering approach based on STPA to develop a safe software. It can be integrated into a software development process or applied directly on existing software.
  - It allows the software and safety engineers to work together during development process of software for safety-critical systems.
- **Limitations**
  - The main steps of approach require manual intervention
  - The difficulty of using formal testing and verification in practice and using formal approaches require some programming knowledge of the software.
- **Future (recent) Work:**
  - We plan to develop a plug-in tool called STPA-verifier which will be integrated with our expansible platform XSTAMPP to enable safety analyst performing STPA and verifying the STPA results with SPIN.
  - We conducted two case studies: the first case study conducted with our industrial partner to investigate the effectiveness of applying the proposed methodology .
  - The second case study conducted during developing a simulator of ACC with LEGO-mindstorm roboter

Thank You

Questions and Feedback are welcome!