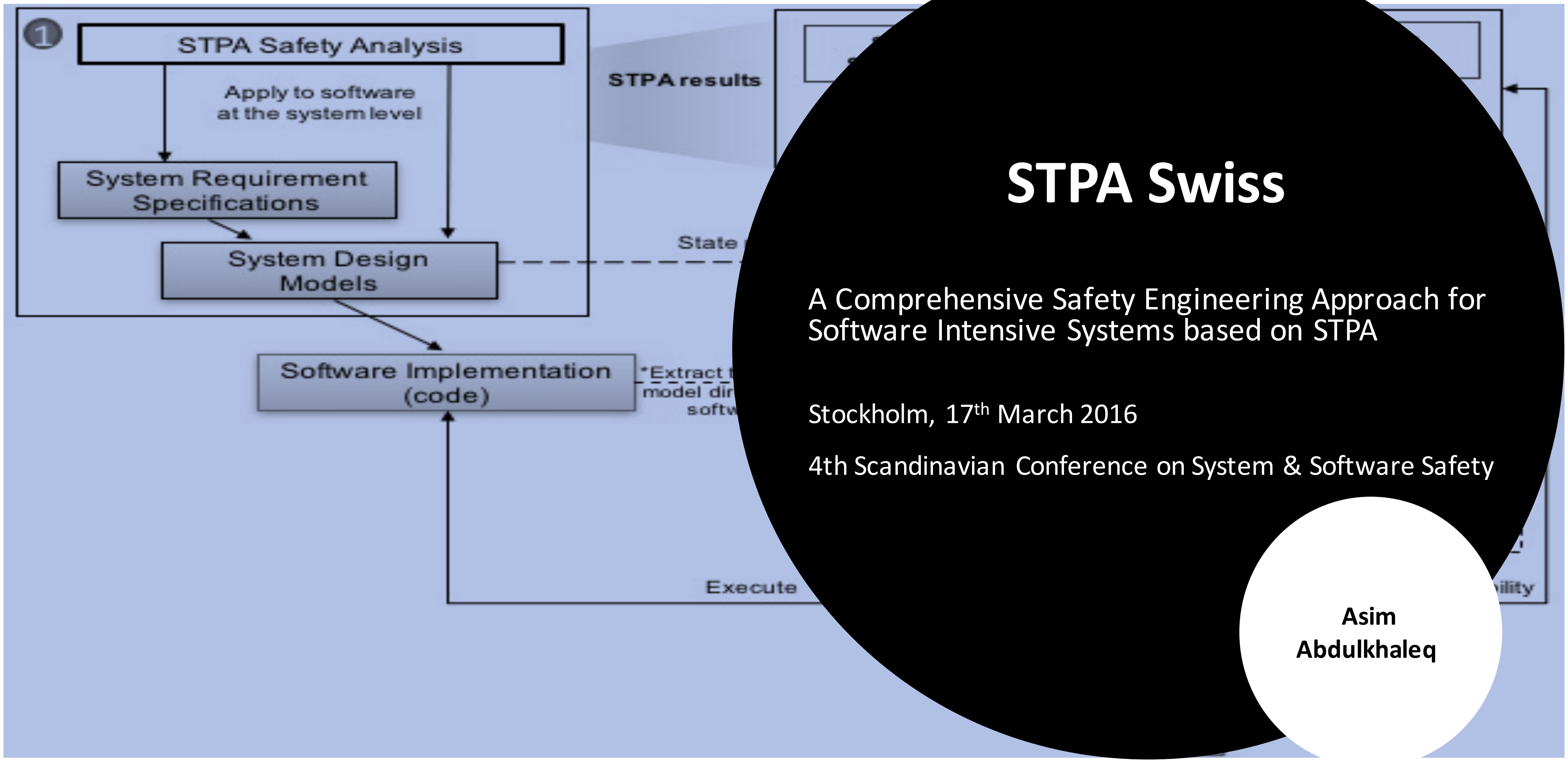
University of Stuttgart
Germany



STPA Swiss

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

Stockholm, 17th March 2016

4th Scandinavian Conference on System & Software Safety

Asim Abdulkhaleq

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 car has something close 100 million lines of software code in it, running on 70 to 100 microprocessors.

Electronic Stability Control

Traction Control

Back Camera

Stop & Go Adaptive Cruise Control

Reverse Backup Sensors

Vehicle

Anti-Lock Braking Systems (ABS)

Tire Pressure Monitoring

Electronic Brakeforce Distribution Systems

Adaptive Cruise Control (ACC)

Automatic Braking Systems

Automatic Braking Systems

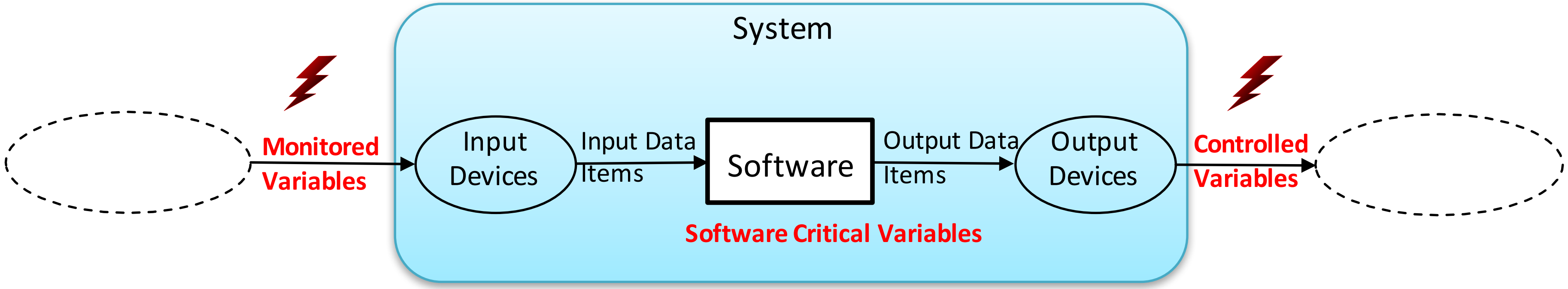
How to recognize the software risks in modern systems and reduce them to a low level?

Agenda

- Motivation ✔
- Problem Statement & Research Objectives ◎
- Background
  - Safety Analysis Techniques
  - STAMP and STPA Approach
- STPA Swiss Approach
- XSTAMPP: Tool support for STPA Swiss Approach
- Illustrative Example: Adaptive Cruise Control System
- Conclusion & Future Work

Software Safety Challenges

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



Verify the software against its safety requirements

- **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

❌ Test all software behaviours is impossible

STPA-Swiss

Identify appropriate software safety requirements

- **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.

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
  - transform them safety into formal specification in LTL/CTL
  - verify them at the design and implementation levels and
  - generate test cases from the information derived during STPA safety analysis.
- We develop a tool support called XSTAMPP to automate the proposed approach.

Background: Safety Analysis Techniques

- There are over 100 different safety analysis techniques.

- **There are some limitations with traditional safety analysis techniques:**
  - They assume that accidents are caused by component failures.
  - They are not adequate to address new accidents caused by **component interactions, human errors, management and organizational errors and software errors [Leveson 2011].**

Systems Approach to Safety Engineering(STAMP)

STAMP Model

STAMP (Systems-Theoretic Accident Model and Processes) is an accident causality model based on systems theory and systems thinking

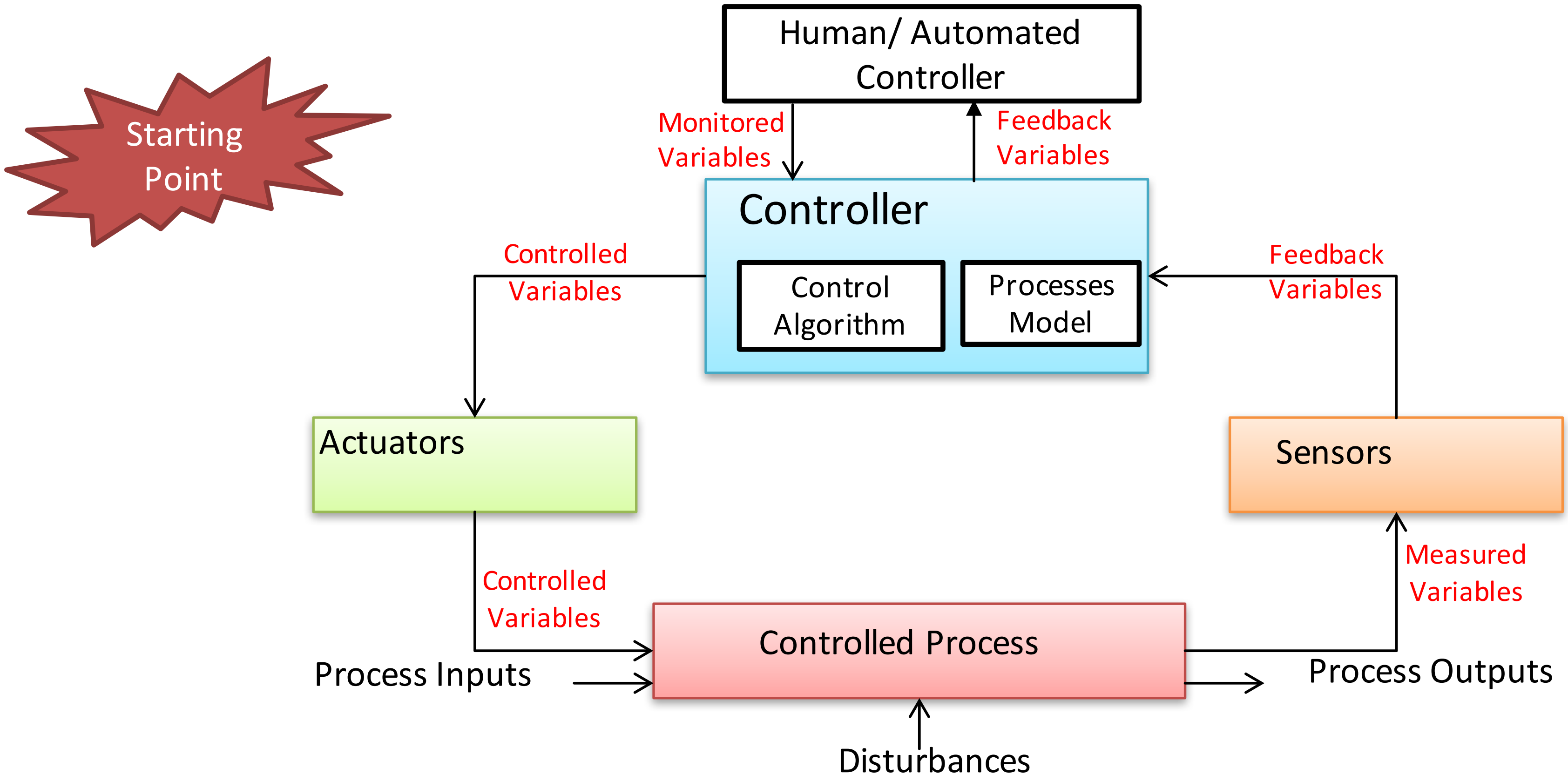
- Accidents are more than a chain of events, they involve complex dynamic processes.
- Treat accidents as a control problem, not a failure problem.
- Prevent accidents by enforcing constraints on component behaviour and interactions.
- Captures more causes of accidents:
  - Component failure accidents
  - Unsafe interactions among components
  - Complex human, software behaviour
  - Design errors
  - Flawed requirements

esp. software-related accidents.

Leveson (2003); Leveson (2011)

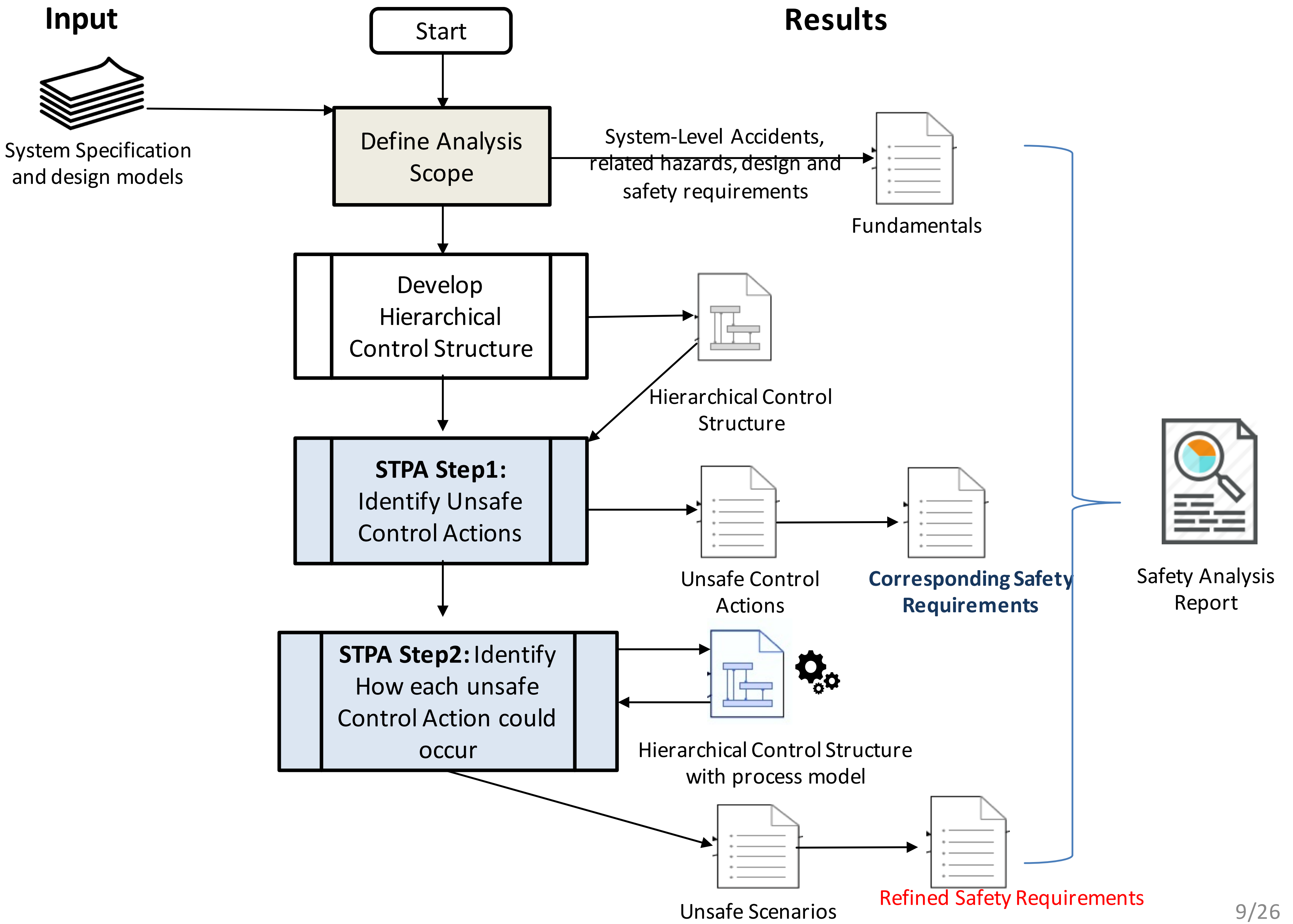
STPA Safety Analysis Technique

- **STPA (System-Theoretic Process Analysis)**
  - Developed by Prof. Leveson at MIT, USA, 2004
  - Built on STAMP model based on system and control theory rather than reliability.
  - Treats safety as dynamic control problem rather than failure problem



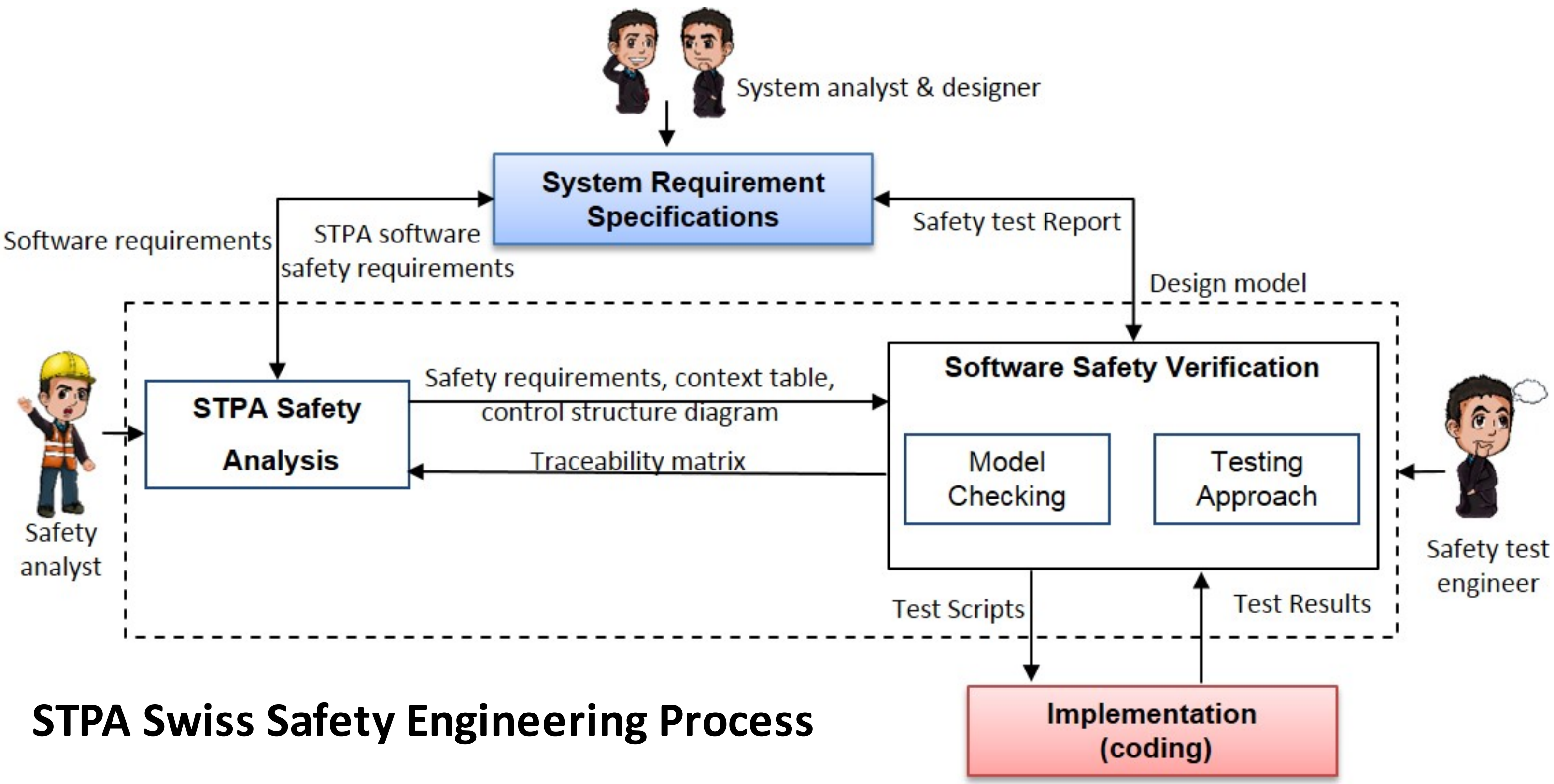
A generic control loop of system

STPA Approach Process



STPA Swiss: A Software Safety Engineering Approach

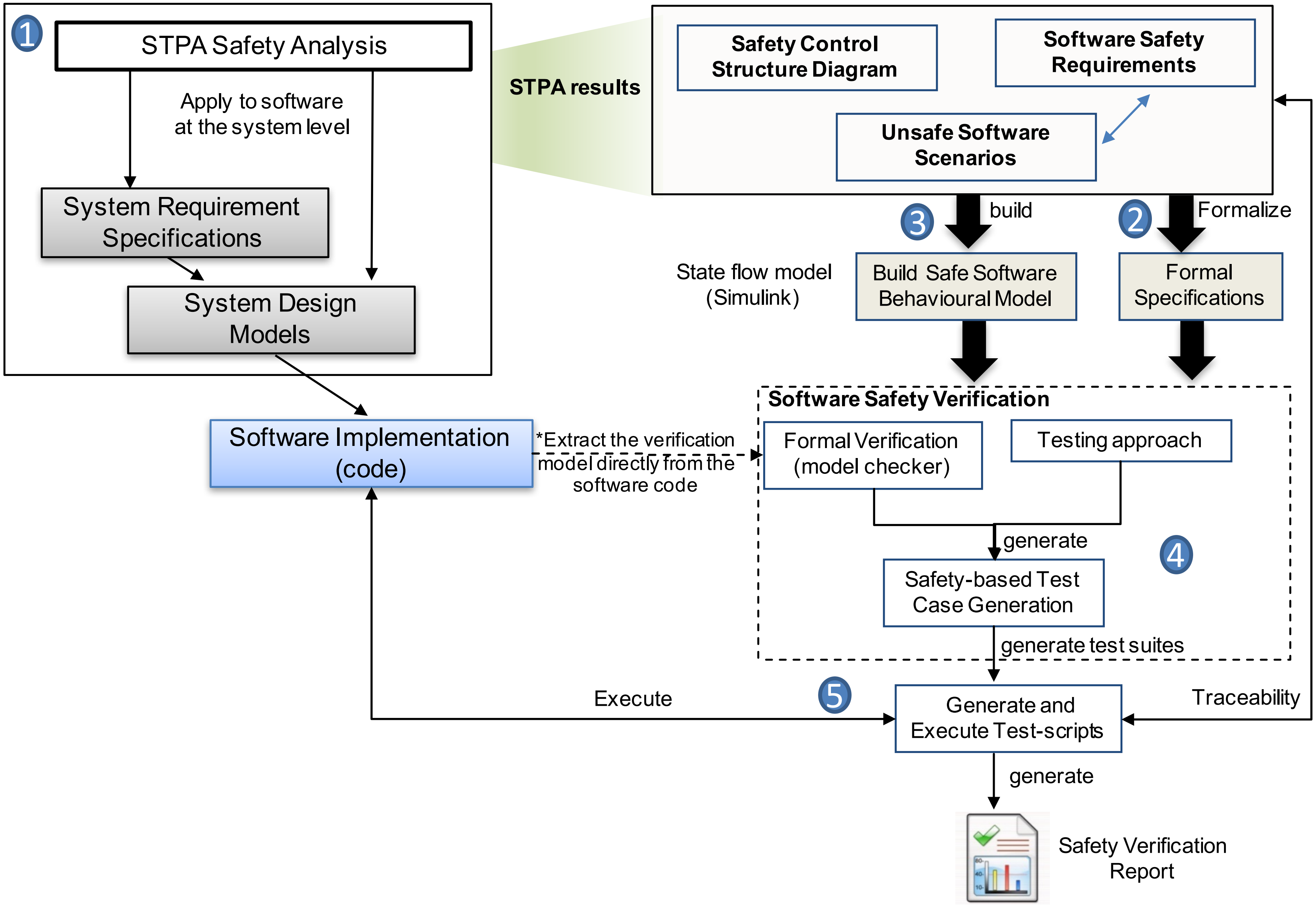
- Major issues of using STPA in software development process:
  - STPA is performed separately and has not been yet placed in software engineering process.
  - The STPA-generated software safety requirements are written in natural language, which we can not directly use them in the verification and testing activities.
  - Identify the unsafe scenarios of complex software based on the combinations of process model variable values manually is time and effort consuming.
  - STPA does not provide any kind of model to visualize the relationship between the critical process variables of controller which have an affect of the safety of control actions.



STPA Swiss Safety Engineering Process

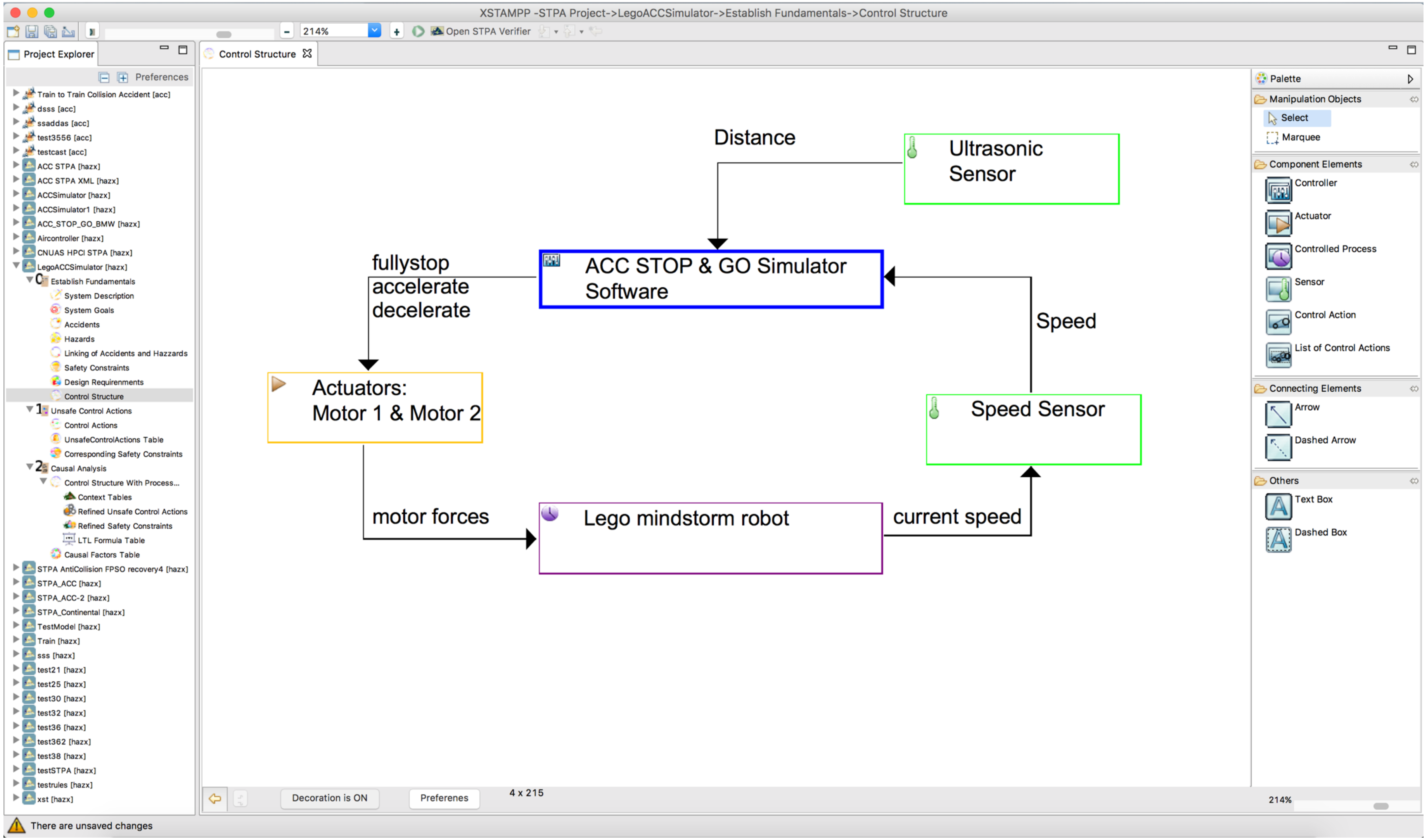
Detailed View of the STPA Swiss Approach

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



Automated STPA Swiss Approach: XSTAMP Platform

- We developed an extensible platform tool support for STAMP safety engineering called **XSTAMPP** as open source platform.

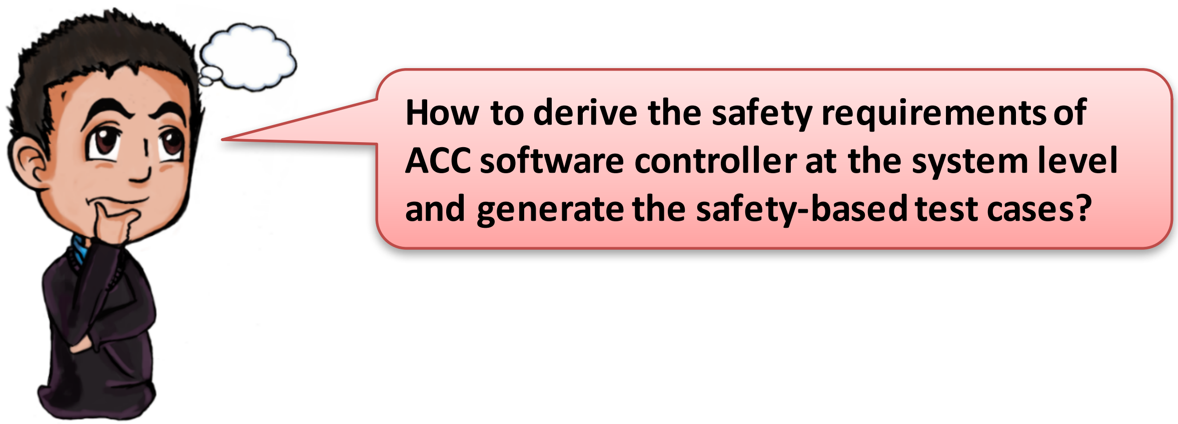


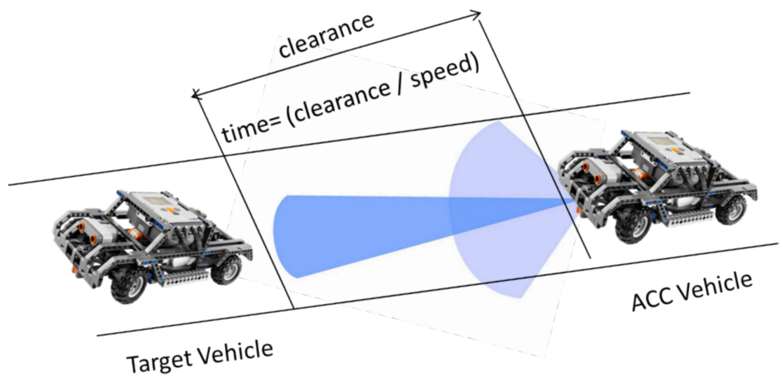
The XSTAMPP main window

www.XSTAMPP.de

Example: Applying STPA to ACC Simulator

- **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.



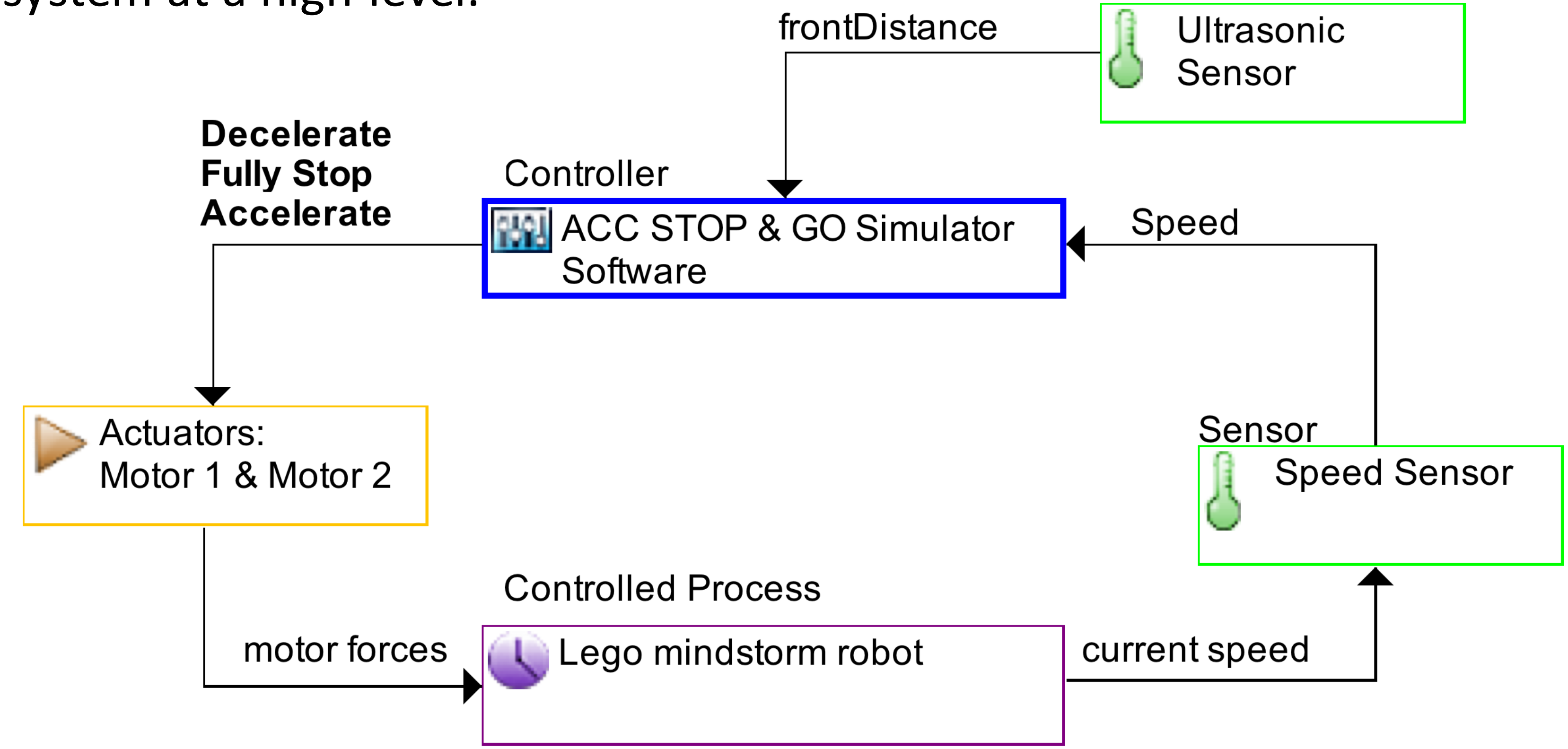


- **Fundamentals of Analysis**
  - **System-Level Accidents:**
    - ACC-1 : ACC vehicle crashes with a vehicle in front.
  - **System-Level Hazards**
    - H-1: ACC software controller does not maintain safe distance from front vehicle.
    - H-2: The ACC software does not stop the vehicle when the front vehicle is fully stopped

http://www.iste.uni-stuttgart.de/en/se/forschung/werkzeuge/acc-simulator.html

Step1.a : Construct The Control Structure Diagram

- **Control Structure diagram** shows the main interconnecting components of the ACC system at a high level.



	Design and Safety Requirements of System
SSR0.1	The ACC simulator should keep a safe distance between the vehicle and a vehicle ahead
SSR0.2	The ACC simulator should stop the vehicle when there is a stopped vehicle in the front.

Step1.b : Identify Unsafe Control Actions

◆ Unsafe Control Actions

Control Action	Not providing causes hazard	Providing causes hazard	Wrong timing or order causes hazard	Stopped too soon or applied too long
Fully Stop	UCA1.1 The ACC software does not bring the robot to fully stop at standstill when the robot vehicle ahead is fully stopped [H-1,H-3]	The ACC software stops the robot suddenly when distance to the robot ahead is too close [Not Hazardous]	The ACC software does not accelerate the speed after the robot vehicle ahead is starting move again. [Not Hazardous]	
Accelerate	The ACC software does not accelerate the speed when the robot vehicle ahead is so far in the lane. [Not Hazardous]	UCA1.2 The ACC software accelerates the speed of robot unintendedly when the time gap to the robot vehicle ahead is smaller than desired time gap [H-1,H-2]	UCA1.3 The ACC software accelerates the speed before the robot vehicle ahead is starting move again. [H-1,H-2]	UCA1.4 The ACC software accelerate the speed too long so that it exceeds the desired speed of the robot [H-2]
Decelerate	UCA1.5 The ACC software does not decelerate the speed when the robot vehicle ahead is too close in the lane. [H-1]	UCA1.6 The ACC software decelerate the speed of robot unintendedly when the time gap to the robot vehicle is approaching too fast. desired time gap. [H-4]	The ACC software decelerate the speed when the robot vehicle ahead is starting move again. [Not Hazardous]	UCA1.7 The ACC software decelerate the speed too short so that it can not bring the robot to fully stop when the robot ahead is stopped. [H-3]

Each unsafe control action is then translated into a system-level safety constraint

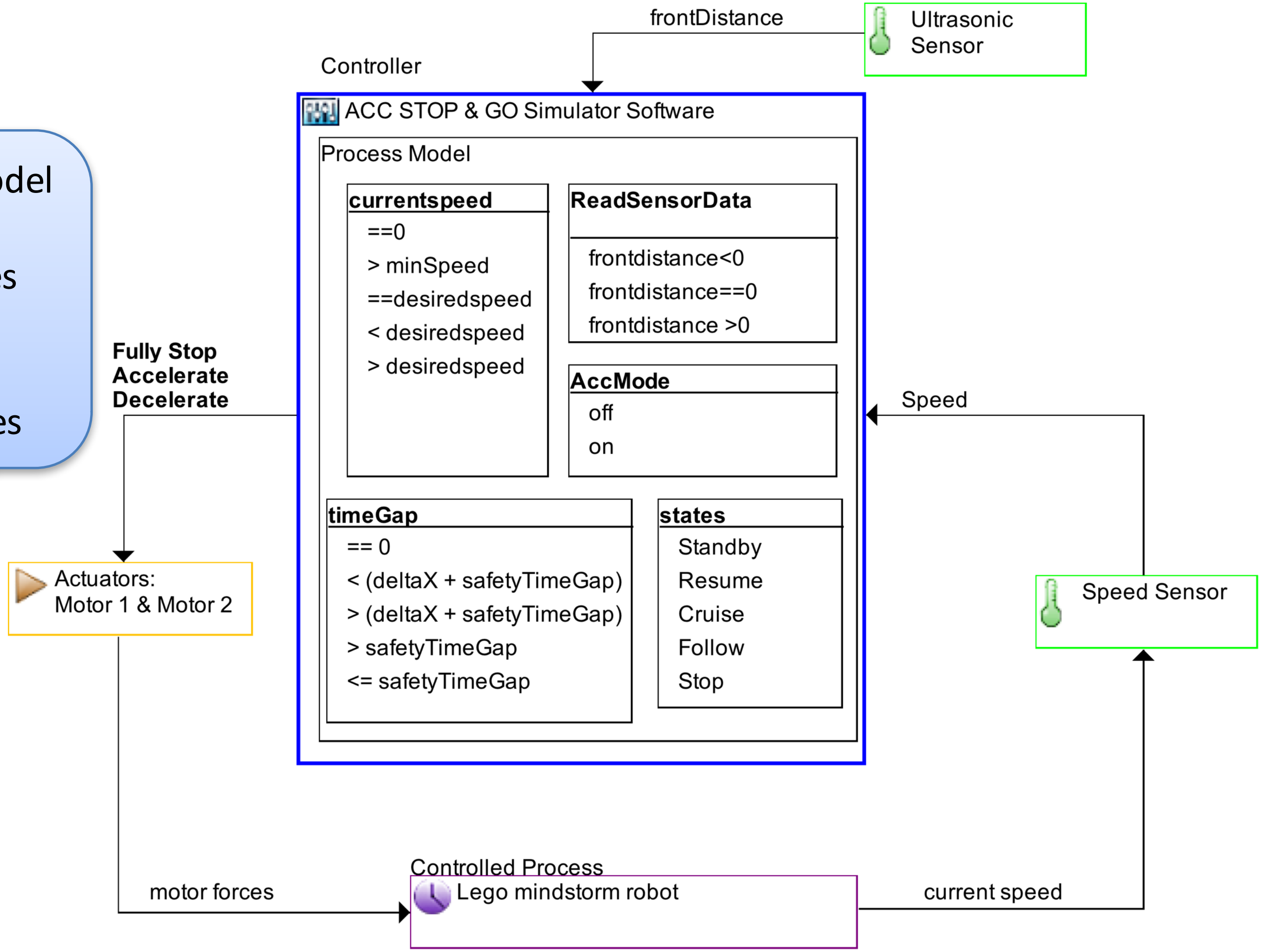
Example: The corresponding safety constraint of UCA1.1 is

SR1.1 The ACC software should bring the robot to fully stop at standstill when the robot vehicle ahead is fully stopped.

Step 1.b: Understand how each UCA could occur

- **Process model** shows the critical variables which have an effect on safety of the control actions.

Four types of process model variables:
(1) Internal states variables
(2) Internal variables
(3) Interaction variables
(4) Environmental variables



- Based on the concept of context tables of each safety-critical actions (John Thomas 2013), we generate the combination sets between process model values

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).**

Context Table of control action Decelerate in context not provided

Control Actions	timeGap	states	currentspeed	RadarSensorData	Hazardous
Decelerate	Standby	== 0	> minSpeed	Frontdistance>0	no
	Follow	< (deltaX + safetyTimeGap)	==desiredspeed	Frontdistance>0	yes
	Standby	> (deltaX + safetyTimeGap)	< desiredspeed	Frontdistance>0	no
	Standby	> safetyTimeGap	> desiredspeed	Frontdistance>0	no
	Standby	<= safetyTimeGap	==0	Frontdistance>0	no
	Resume	== 0	==desiredspeed	Frontdistance>0	no
	Resume	< (deltaX + safetyTimeGap)	< desiredspeed	Frontdistance>0	yes
	Resume	> (deltaX + safetyTimeGap)	> desiredspeed	Frontdistance>0	no
	Resume	> safetyTimeGap	==0	Frontdistance>0	no
	Resume	<= safetyTimeGap	> minSpeed	Frontdistance>0	no
	Stop	== 0	==0	Frontdistance>0	no
	Stop	< (deltaX + safetyTimeGap)	> minSpeed	Frontdistance>0	no
	Stop	<= safetyTimeGap	> desiredspeed	Frontdistance>0	no

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

Automatically Generate LTL formulae

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

Control actions	Process Model variables				Hazardous
	timeGap	CurrentSpeed	RadarData	states	
Accelerate Signal	< (deltaX + safeTimeGap)	== desired speed	Frontdistance>0	Cruise	Yes
	> safeTimeGap	<desired speed	Frontdistance>0	Cruise	No
	< safeTimeGap	> Desired speed	Frontdistance>0	follow	Yes (H1, SSR1)

Refine the software safety Requirements

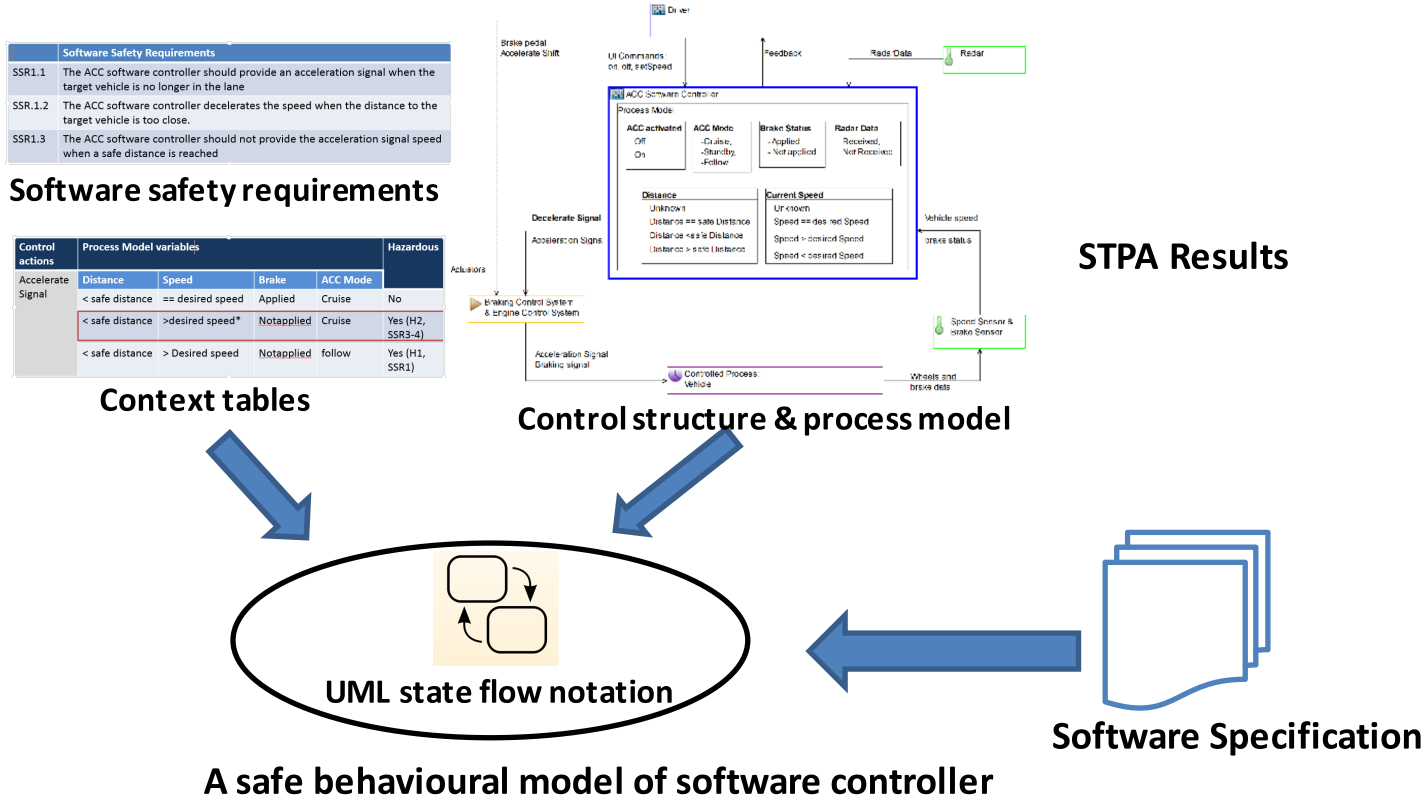
$SSR_{1.3}$: ACC should not provide accelerated signal when the TimGap is less or equal the safeTimeGap while ACC in follow mode current speed is greater than desired speed.

Generate LTL formula

$LTL_{1.3}$ `G ((states=follow)&(timeGap<safeTimeGap)&(currentspeed>DesiredSpeed)& frontdistance >0 )->! ((controlAction=Accelerate))`

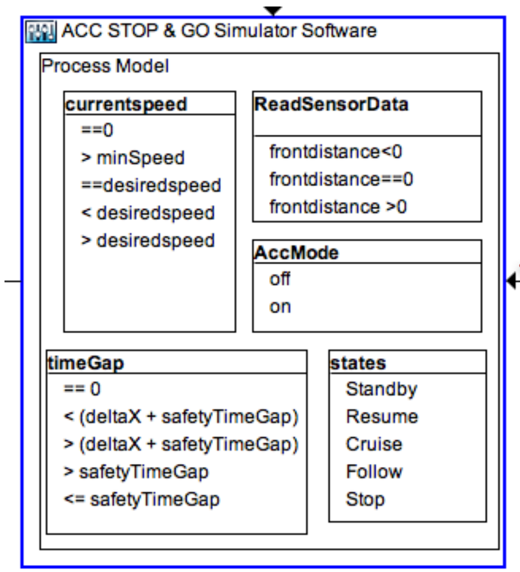
Step 2 : Constructing the safe behavioural model of software controller

- **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 behavioural model**
  - **The model should be constrained by STPA safety requirements**

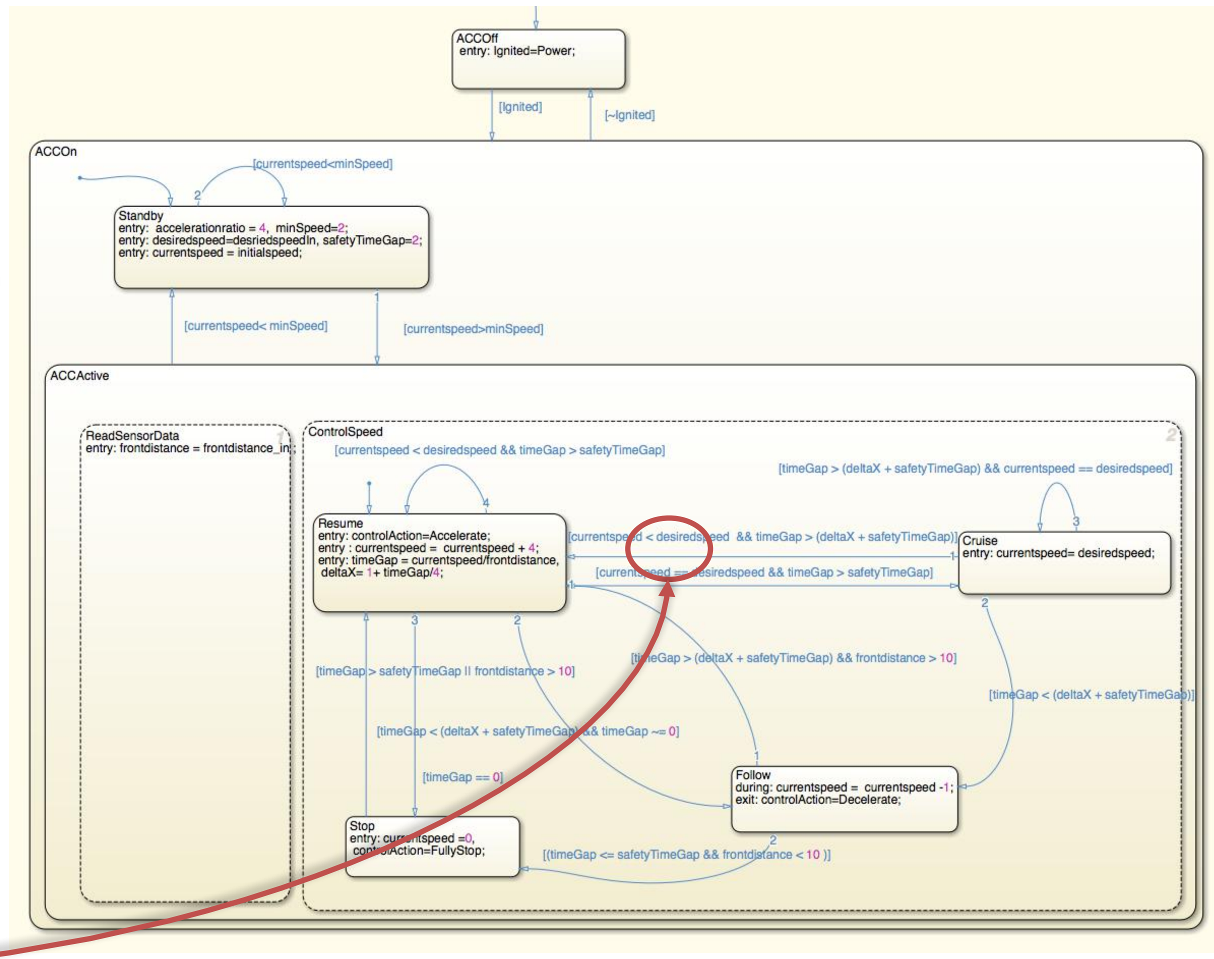


- **Syntax of each transition of the safe behaiovural model:**

Step 2 : The safe behavioural model of ACC software controller



A safe behavioural model of ACC software Controller



Software Controller & process model variables

Control actions	Process Model variables				Hazardous
	timeGap	CurrentSpeed	RadarData	ACC Mode	
Accelerate Signal	< (deltaX + safeTimeGap)	== desired speed	Frontdistance>0	Cruise	Yes
	> safeTimeGap	<desired speed	Frontdistance>0	Cruise	No
	< safeTimeGap	> Desired speed	Frontdistance>0	follow	Yes (H1, SSR1)

Context Table

Transition : (safety requirement)

[currentSpeed == desiredSpeed && timGap> (deltaX+safeTimeGap) && ACCMode == Cruise]

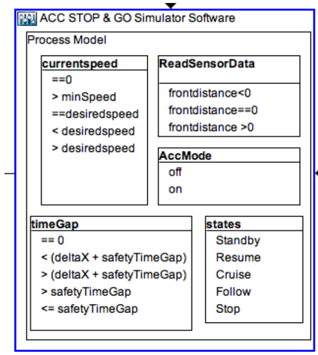
Parallel Process variables

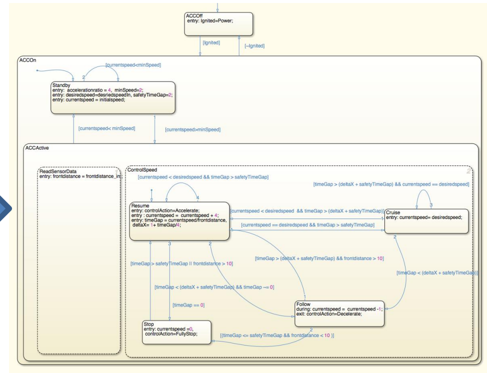
Sequential Process variables

Step 3.1 : Automatically generate Verification Model of SBM

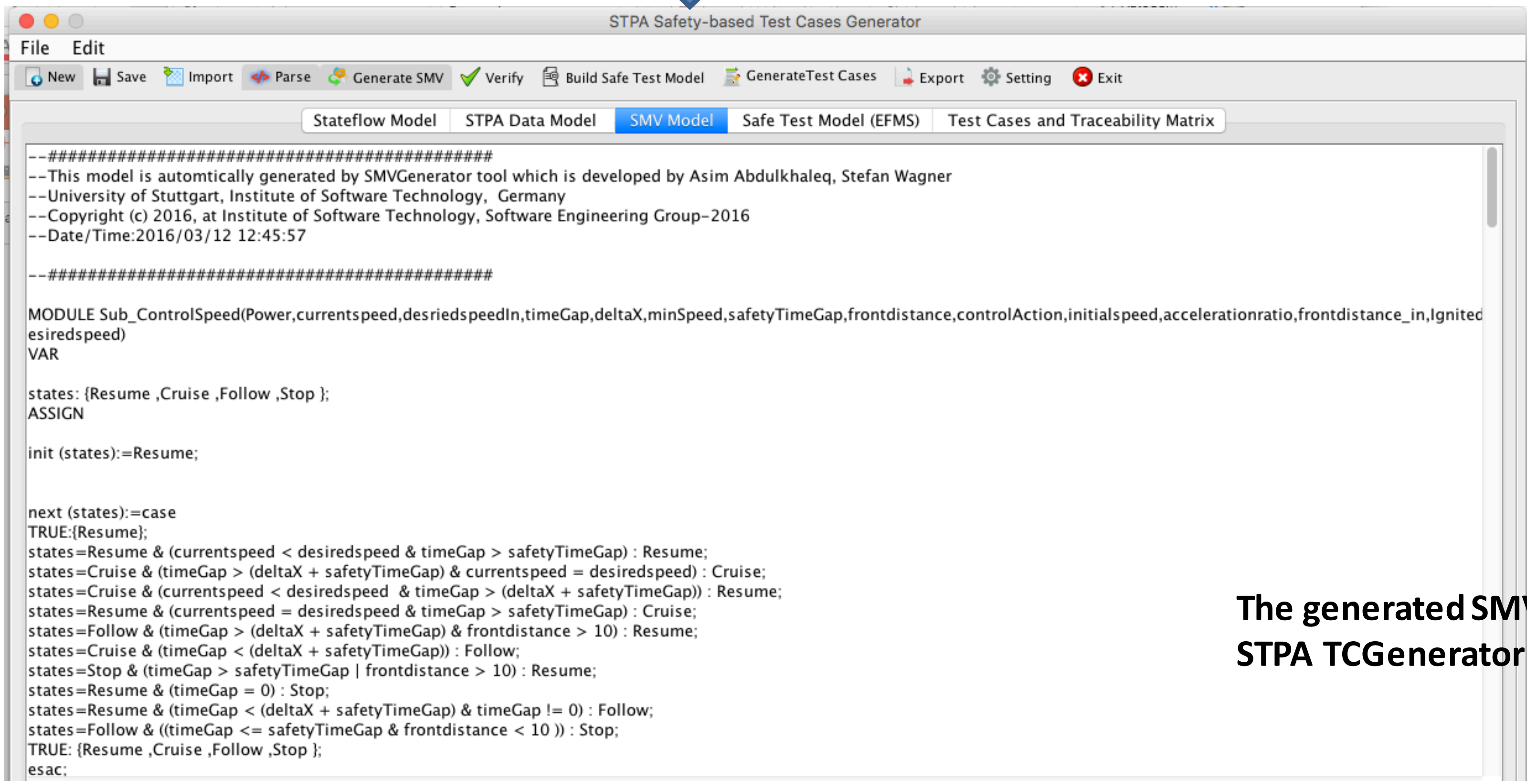
- **To check whether the safe behavioural model satisfy the STPA safety requirements, we developed a tool called STPA TCGenerator which automatically converts the safe behavioural model into a input language of model checker such as SMV (Symbolic Model Verifier) model**

STPA Process Model (static)





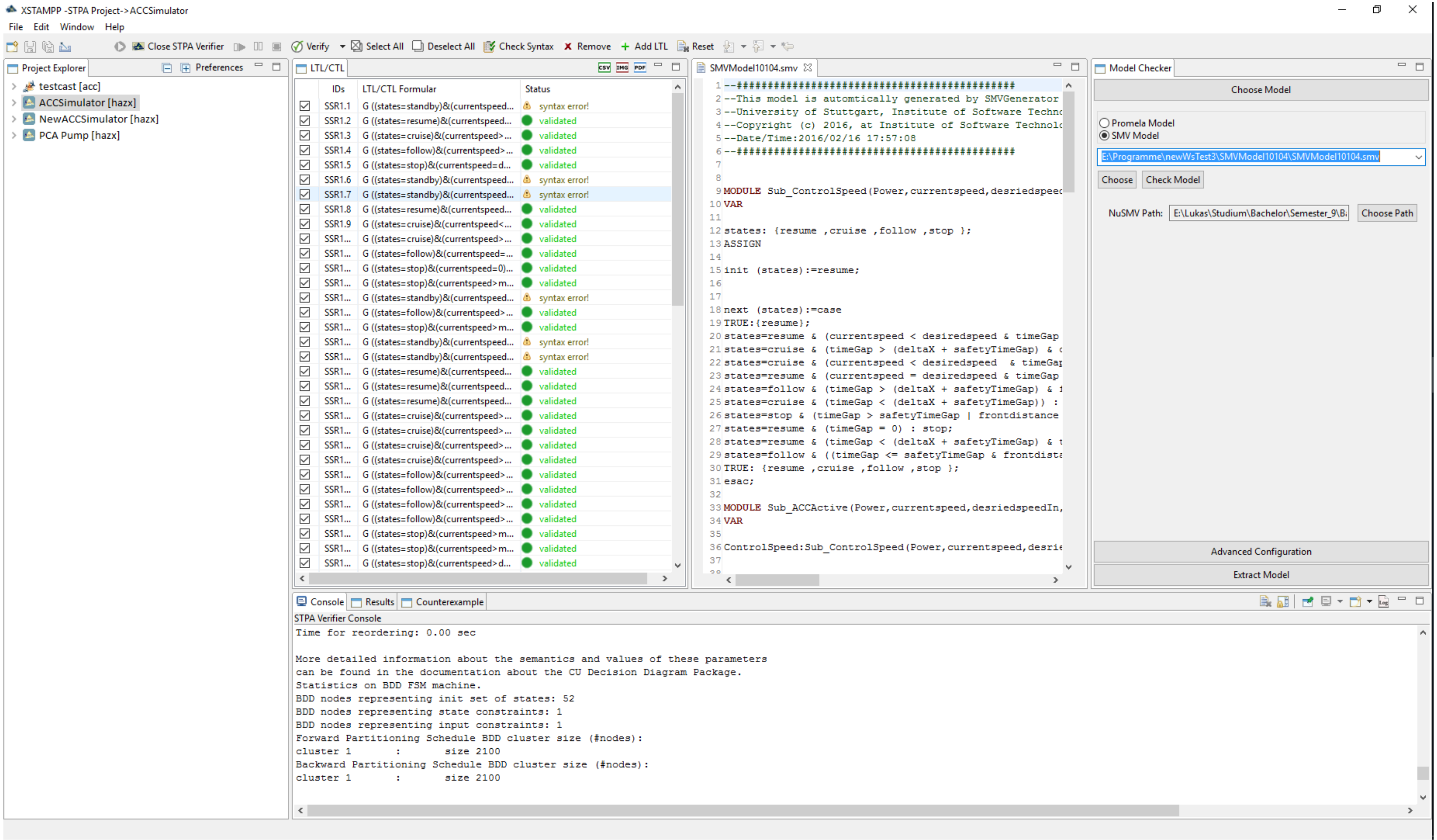
Simulink Stateflow (dynamic)



The generated SMV model by STPA TCGenerator tool

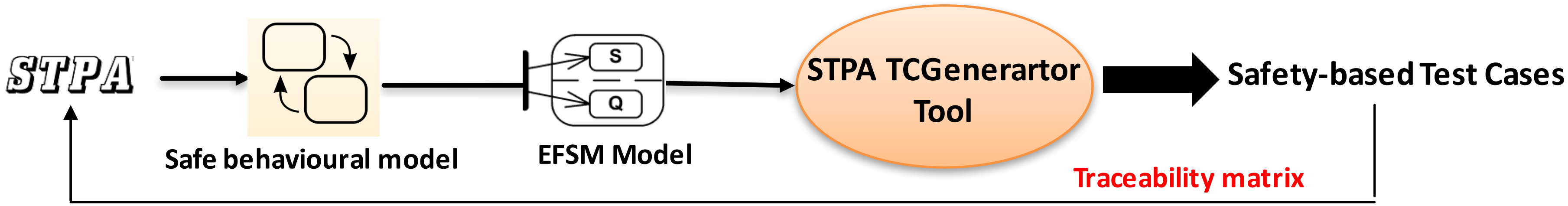
Step 3.1 : Check Correctness of Safe Behavioural Model of SW Controller

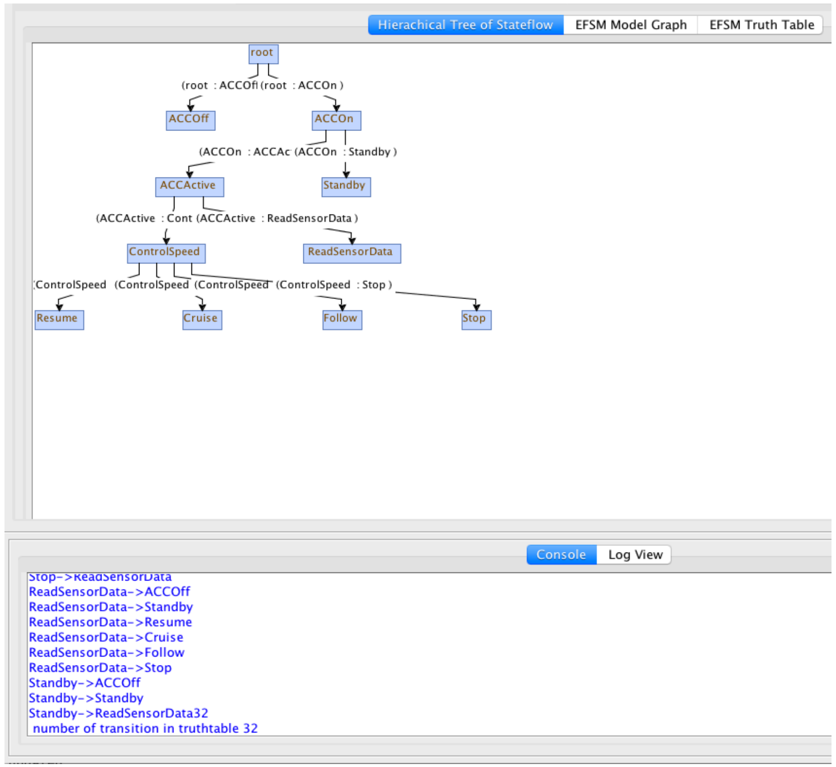
- **Second, we developed a plug-in based on XSTAMPP called STPA verifier to verify the LTL formulae with NuSMV model checker tool**



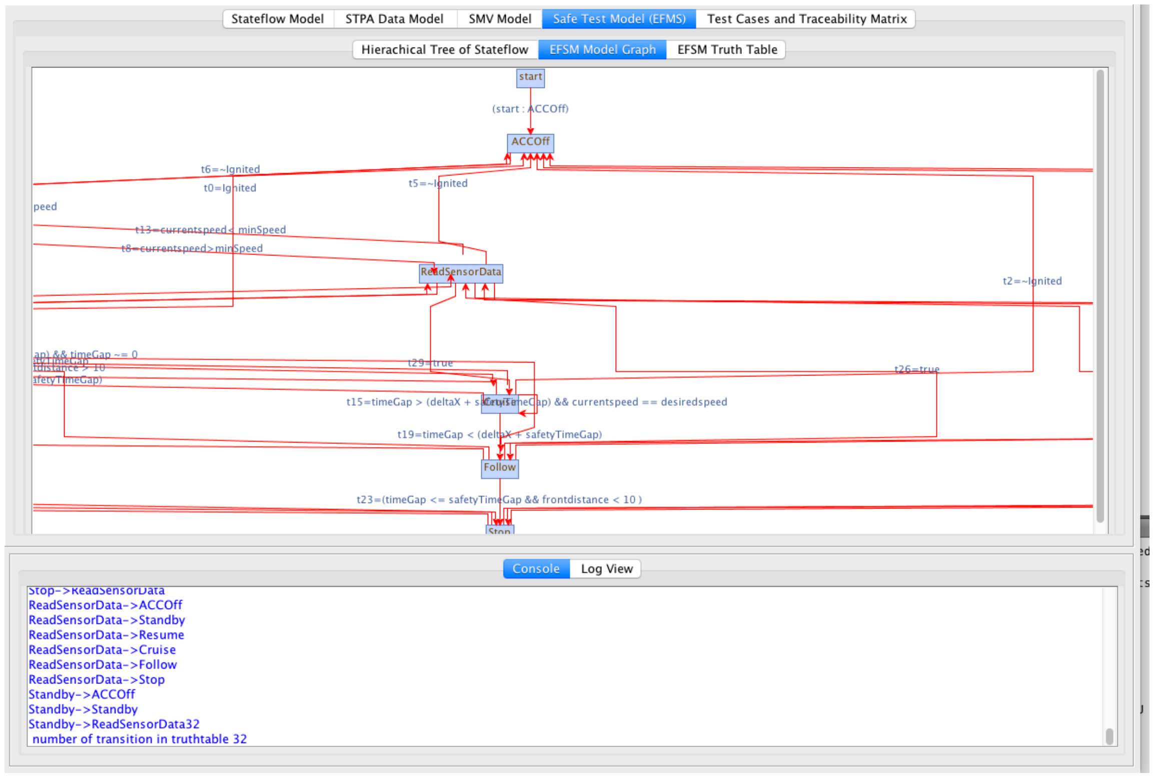
Step 3.2 : Safety-based Test Cases Generating

- **To generate safety-based test cases based on STPA results,**
  - We automatically convert safe behavioural model into extended finite state machine.
  - We use EFSM as input to the **STPA TCGenerator** to generate test cases for each STPA SSR.





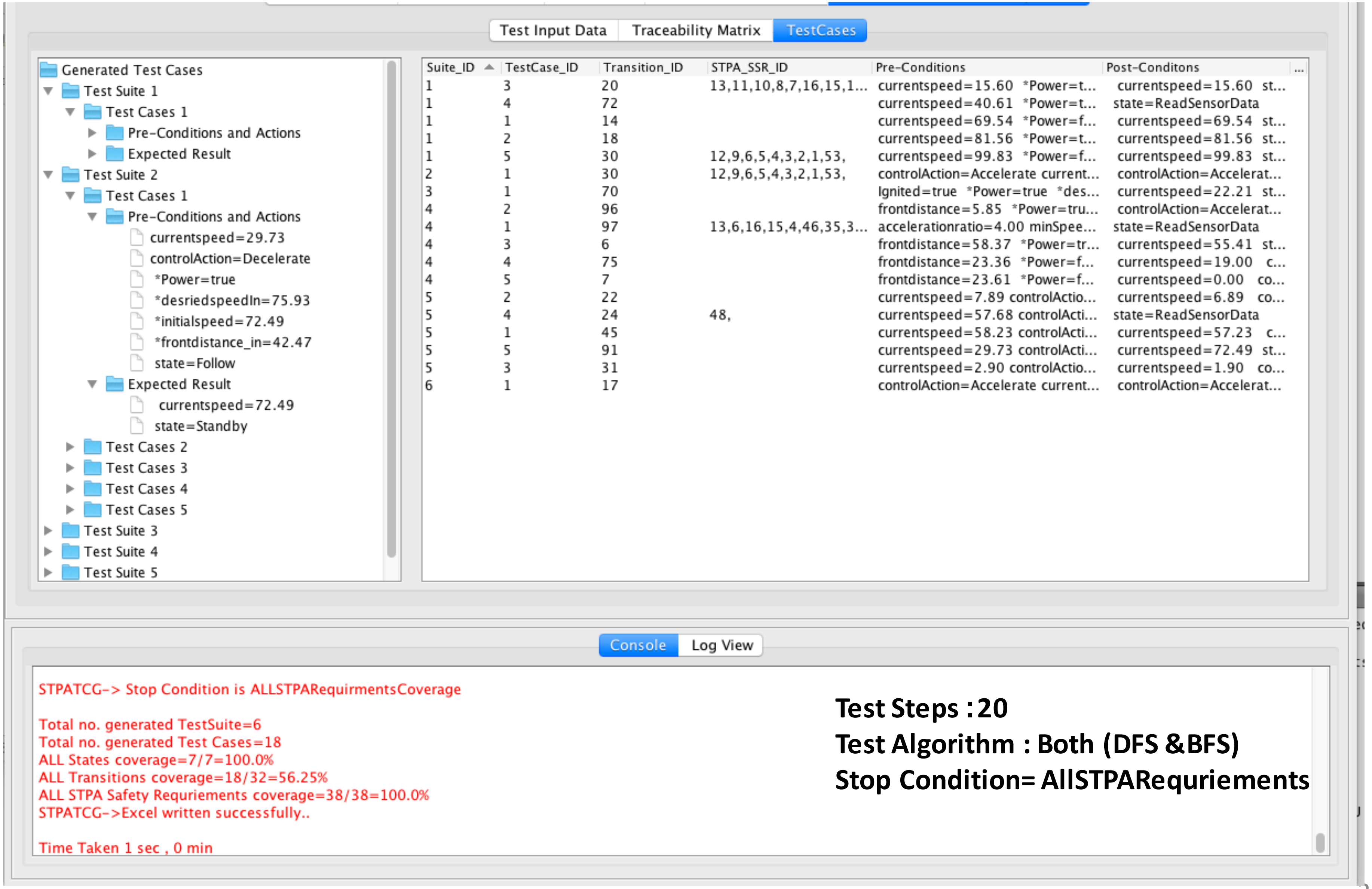
Stateflow Tree of SBM



Extended finite state machine diagram of SBM

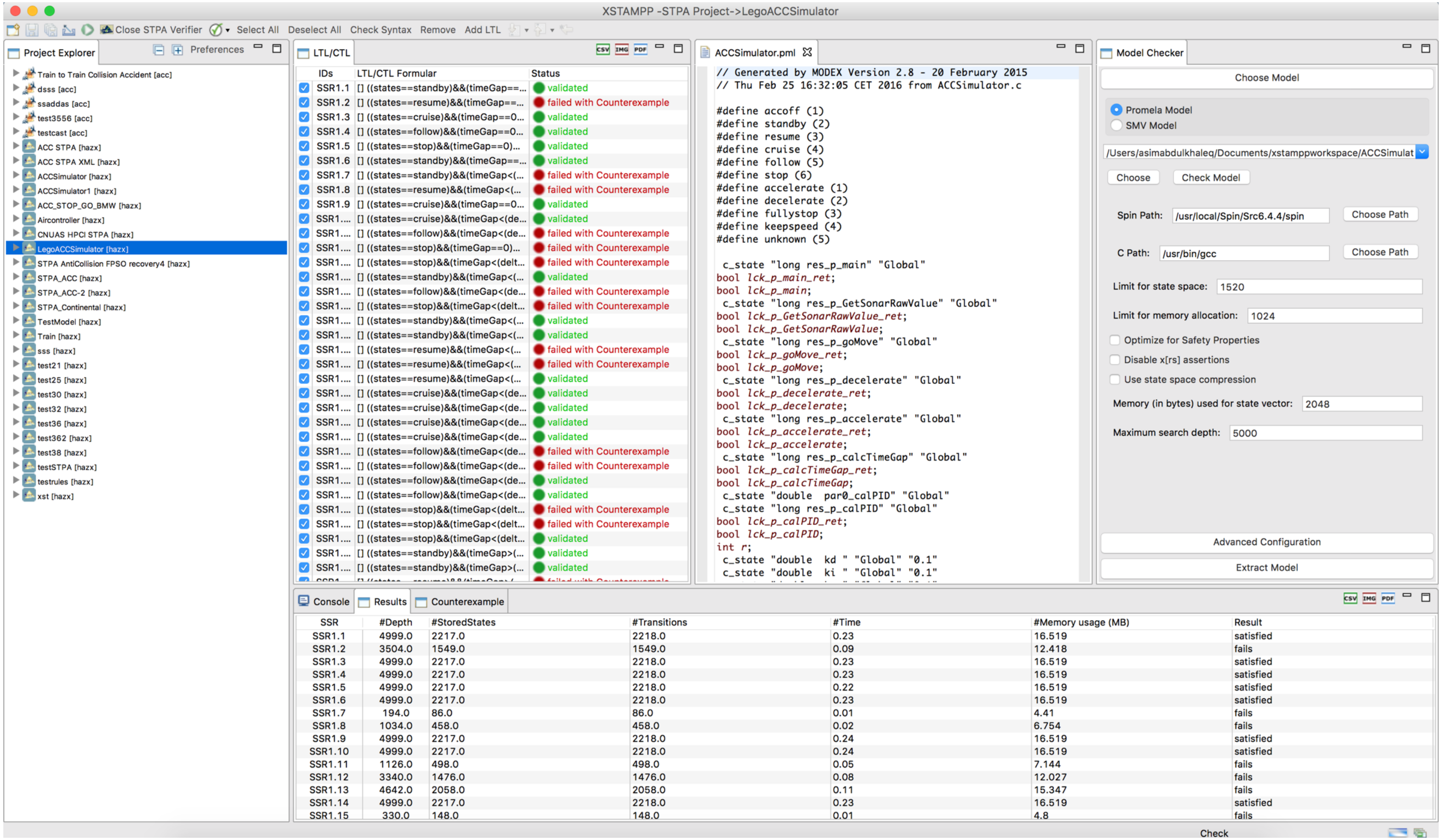
The Results of Test Cases Generating

- **We generated automatically 18 test cases which cover the safe behavioural of the ACC software controller with the state coverage =7/7, transition coverage =18/32, and the STPA Safety Requirements coverage 38/38.**



Verifying STPA Safety Requirements at the implementation level

- **We use STPA verifier to verify the LTL formulae with SPIN model checker tool based on the verification model which is extracted directly from C source code of ACC by Modex tool**



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.
  - We conducted a case study to evaluate STPA Swiss during developing a simulator of ACC with LEGO-mindstorm roboter at our institute.
- **Future (recent) Work:**
  - We conducted a case study with our industrial partner to investigate the effectiveness of applying the STPA Swiss approach to a real system.
  - We plan to position the STPA Swiss approach into an automotive development process of our industrial partner.

University of Stuttgart
Germany

Joint Work with:
Prof. Dr. Stefan Wagner

Thank you!

SE

Asim Abdulkhaleq, M.Sc.

e-mail Asim.Abdulkhaleq@informatik.uni-stuttgart.de
phone +49 (0) 711 685- 88 458
fax +49 (0) 711 685- 88 389

Universität Stuttgart
Institute of Software Technology

www.xstampp.de