Dezyne succeeds ASD

The new upgrade in formal verification based MDSD

SIOUX

SOURCE OF YOUR TECHNOLOGY

60th Systems Architecture Study Group Meeting

June 15, 2017

Author: Michaël van de Ven (Sioux)

About Michaël van de Ven

- Joined Sioux in 2006
- Technology Specialist ASD/Dezyne
- Experience with Verum's technology in >10 projects as of 2009
- Contributing to Dezyne since the start in '14

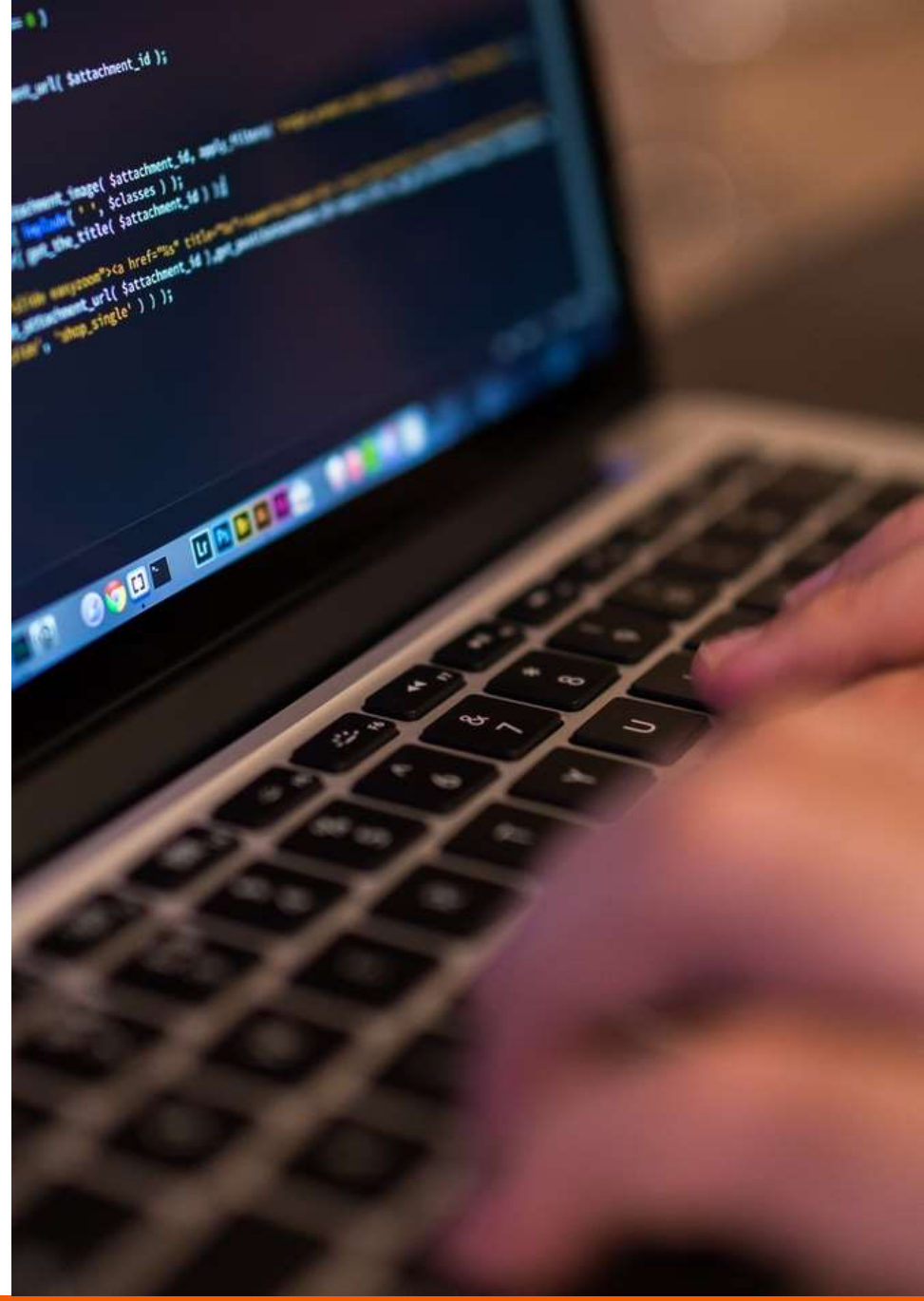
About Sioux

- Founded in 1996 (Eindhoven), grown to 500 employees
- Technical software, mechatronics, electronics, industrial mathematics, remote solutions
- The innovative technology partner for high tech companies:
  - Supports in development and manufacturing of their products
  - Help in shorten the development time by excellent productivity

SOURCE OF YOUR TECHNOLOGY

Sectors

- Semicon & Solar
- Life Science & Health
- Automotive
- Image & Printing
- Consumer Electronics & Telecom
- Traffic, Transport & Infrastructure
- Agro & Food



About Verum

- Founded in 2004 (Waalre)
- Academic background:
  - TU/e, Oxford University, University of Tennessee
- Provides software engineering tools for designing verifiably correct embedded software:
  - **ASD**: Analytical Software Design
  - **Dezyne**: successor of ASD as of 2014

verum®

Facing the challenges

The challenge

In a world of software controlled systems getting more complex, how could engineers master the challenge?

Example of challenges

- System decomposition: initially vs future
- Correctness of components functioning together

and more…

> Concurrency

> Reliability / Safety

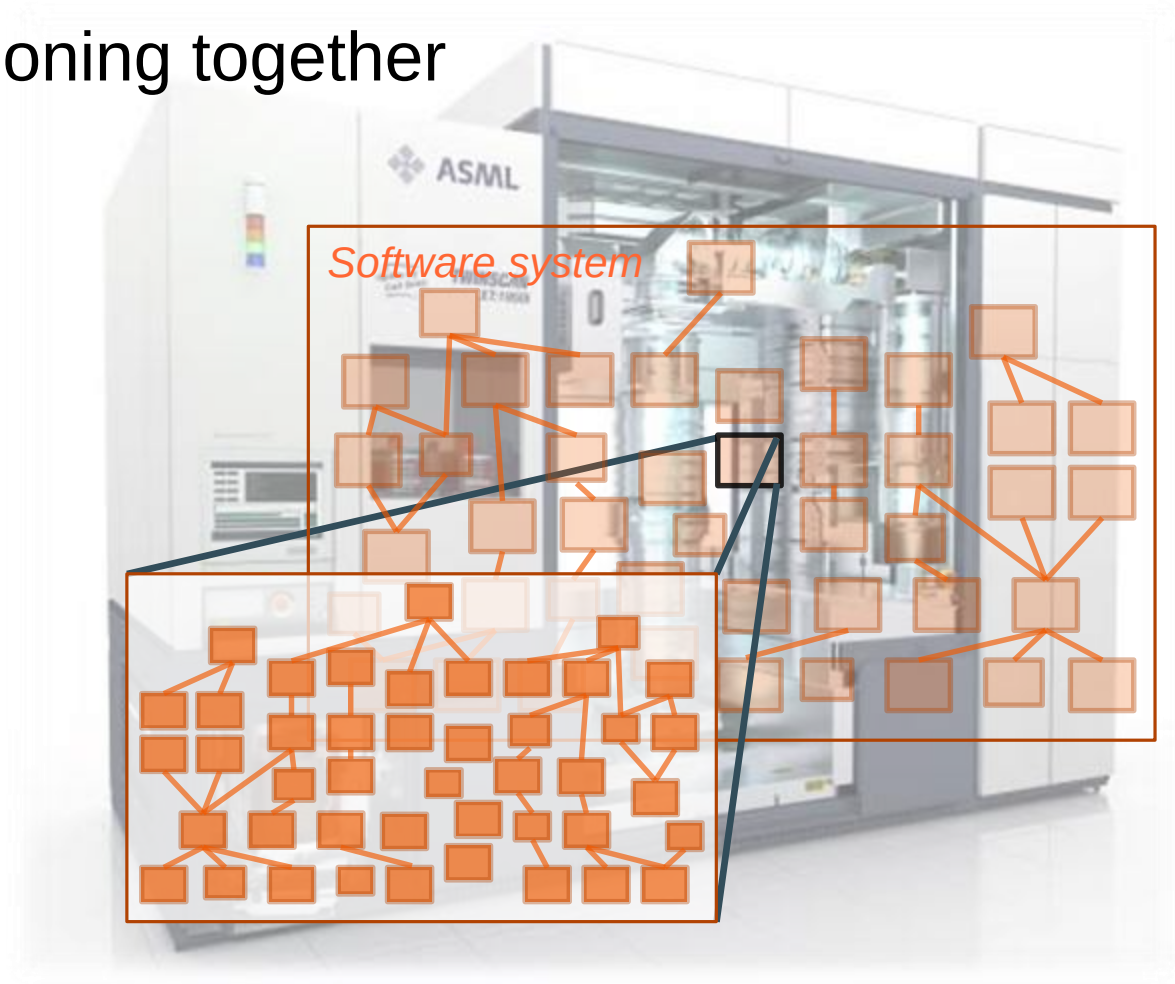
> Scalability

> Time to market

> Increase productivity

> Creating features vs. solving defects

> Burdened with technical debt



Do we still enjoy coding?

When software systems grow further and further

- Keep on handwriting only: **unsustainable**
- Quality assurance: how to proof absence of **defects?**
- Growing automated testsuites contribute to **even more** handwritten code

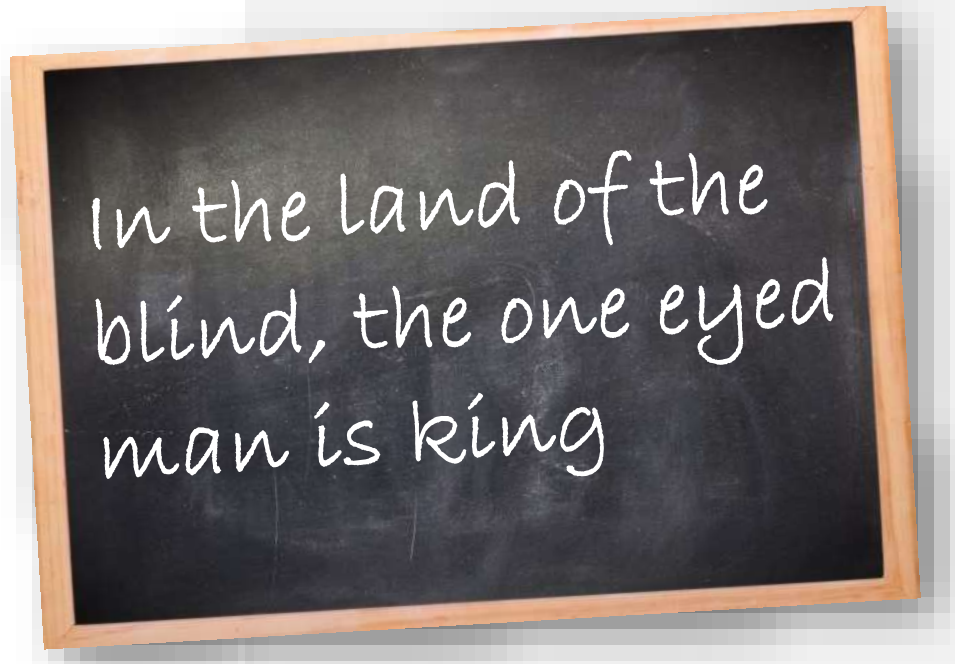
>> Model-driven development seems to be a saviour **<<**

Model-driven, with verification?

- Model-driven methods enable us to specify on a more abstract level, and then generate code. That's a good thing!

- Some methods also promise 'checking'. But behold the assumption that it would imply defect-freeness!

- So what is being checked?
  - > Completeness of specification?
  - > Coverage of all scenarios, not just only the happy flow?
  - > Nasty defects like: race conditions, dead/live-locks?
  - > What is the case: verification and/or validation. Or nil?



ASD/Dezyne to the rescue

ASD & Dezyne

Sioux has knowledge of various model driven engineering tools. The tools from Verum are preferred when control logic is being developed.

> Component based development

modelling of interfaces, components and systems

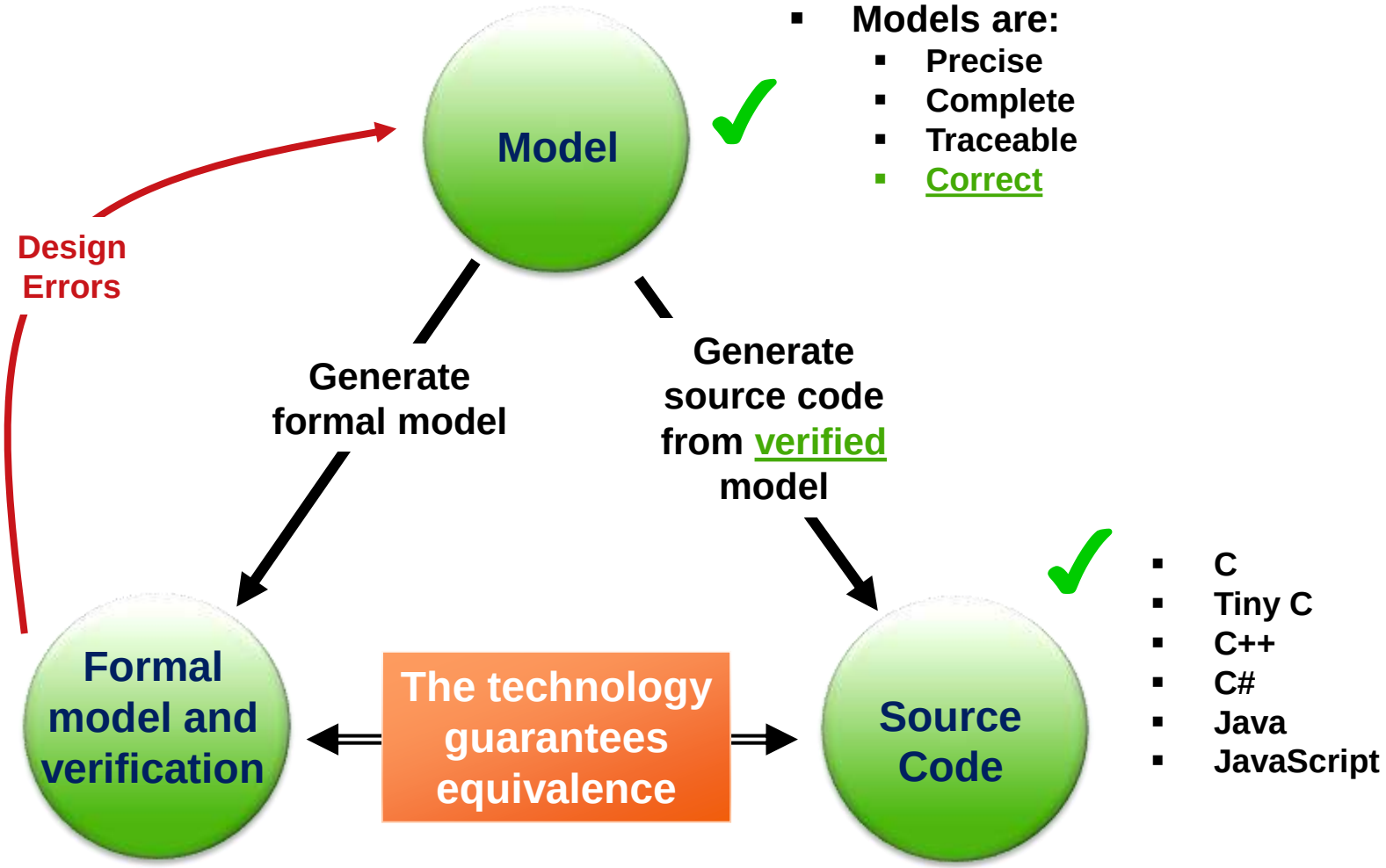
> Mathematical power

verification, simulation and code generation

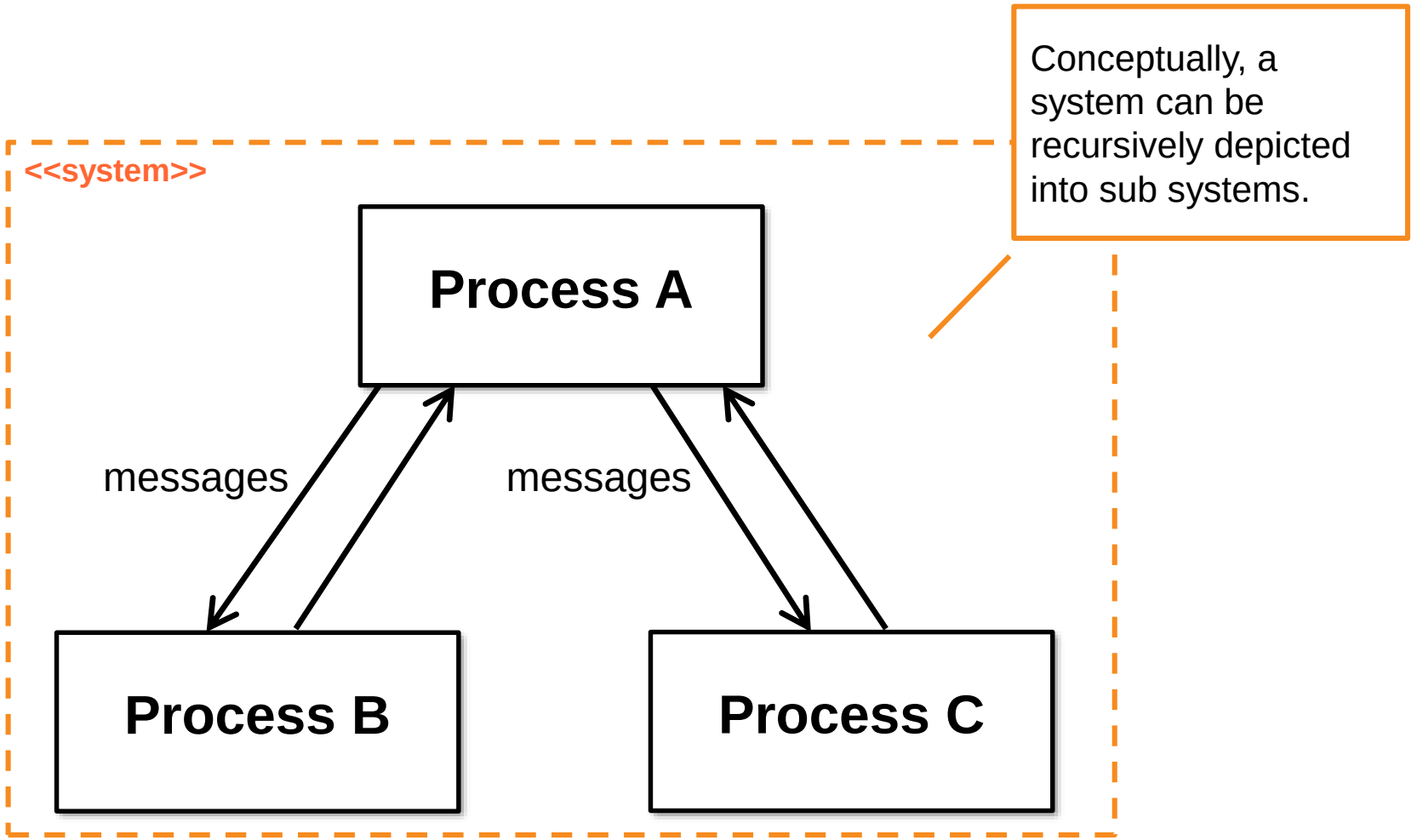
> Dezyne (aka ASD Gen 2) provides for the future

open language, easier to adopt and extendability

The fundament of ASD/Dezyne

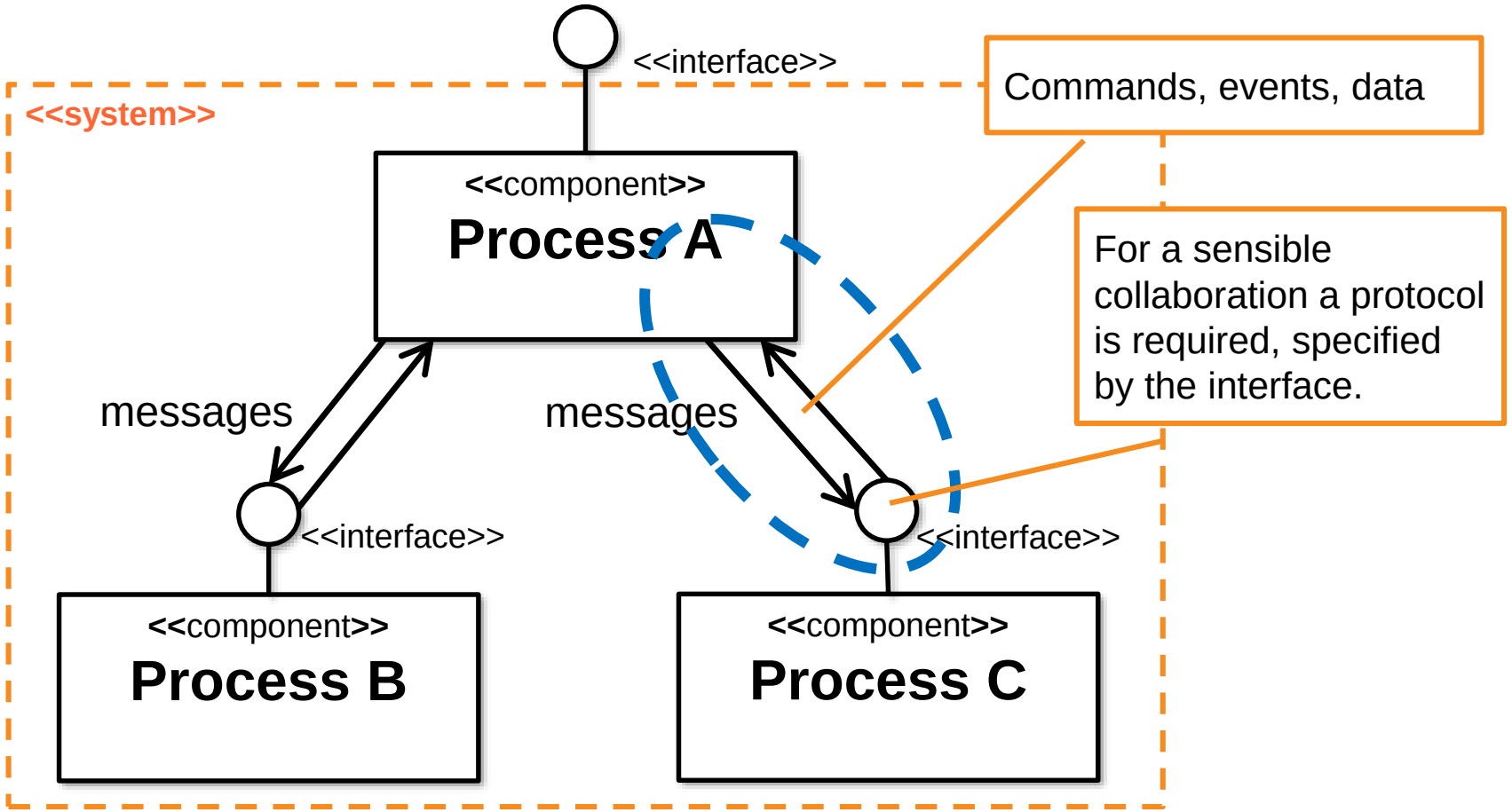


Subdividing and encapsulation



Collaborating processes take a system responsibility.

Collaboration via interfaces



An interface is the contract between two software components.

There’s no interface without a protocol

A **protocol** must define the syntax, semantics, and synchronization of communication. The specified behavior is typically independent of how it is to be implemented.

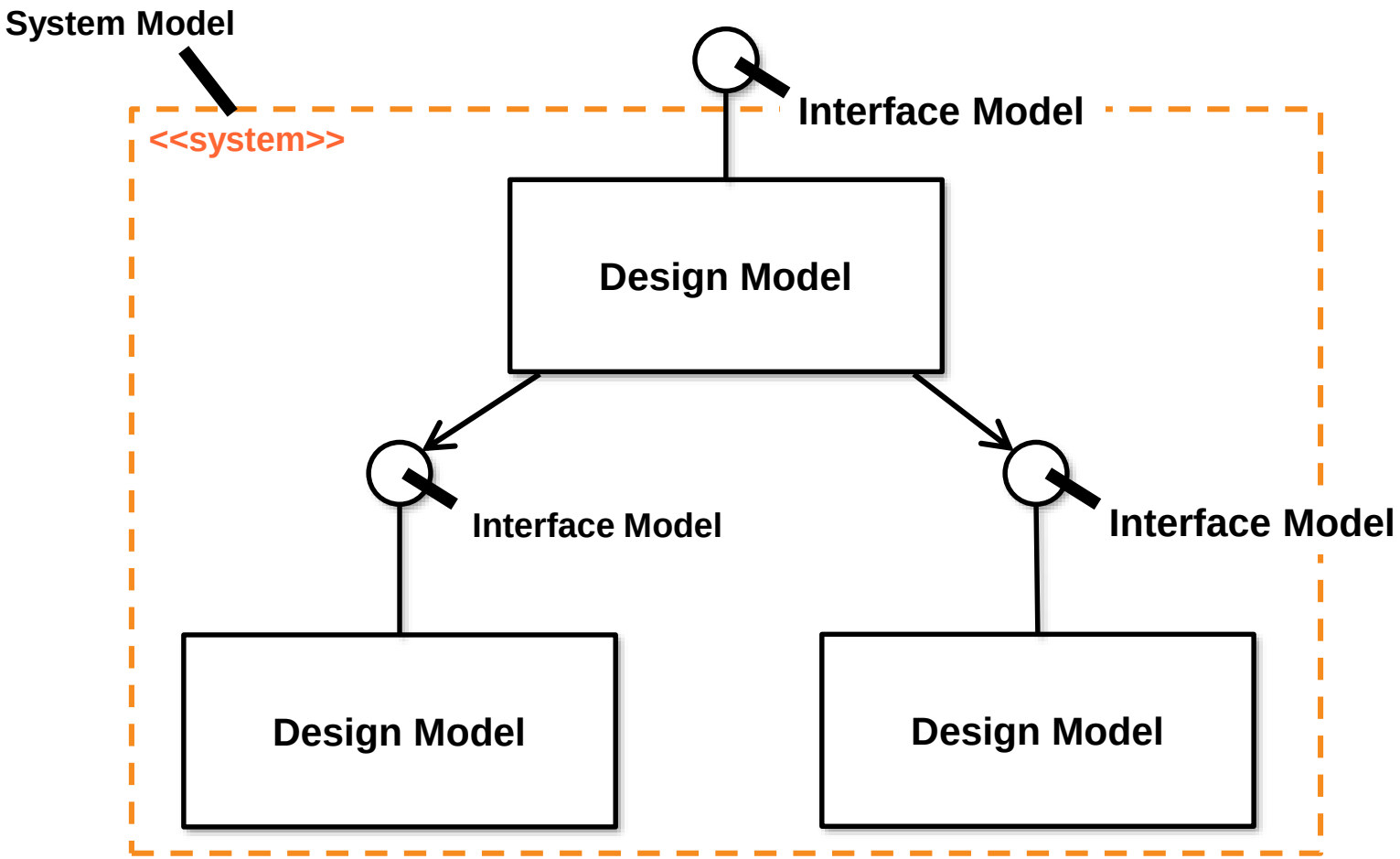
> Syntax: function prototype, the API signature

> Semantics: functional description, meaning

> Synchronization: what/when, functional behavior

Important: multiple realizations can have the same external visible behavior

Three types of models

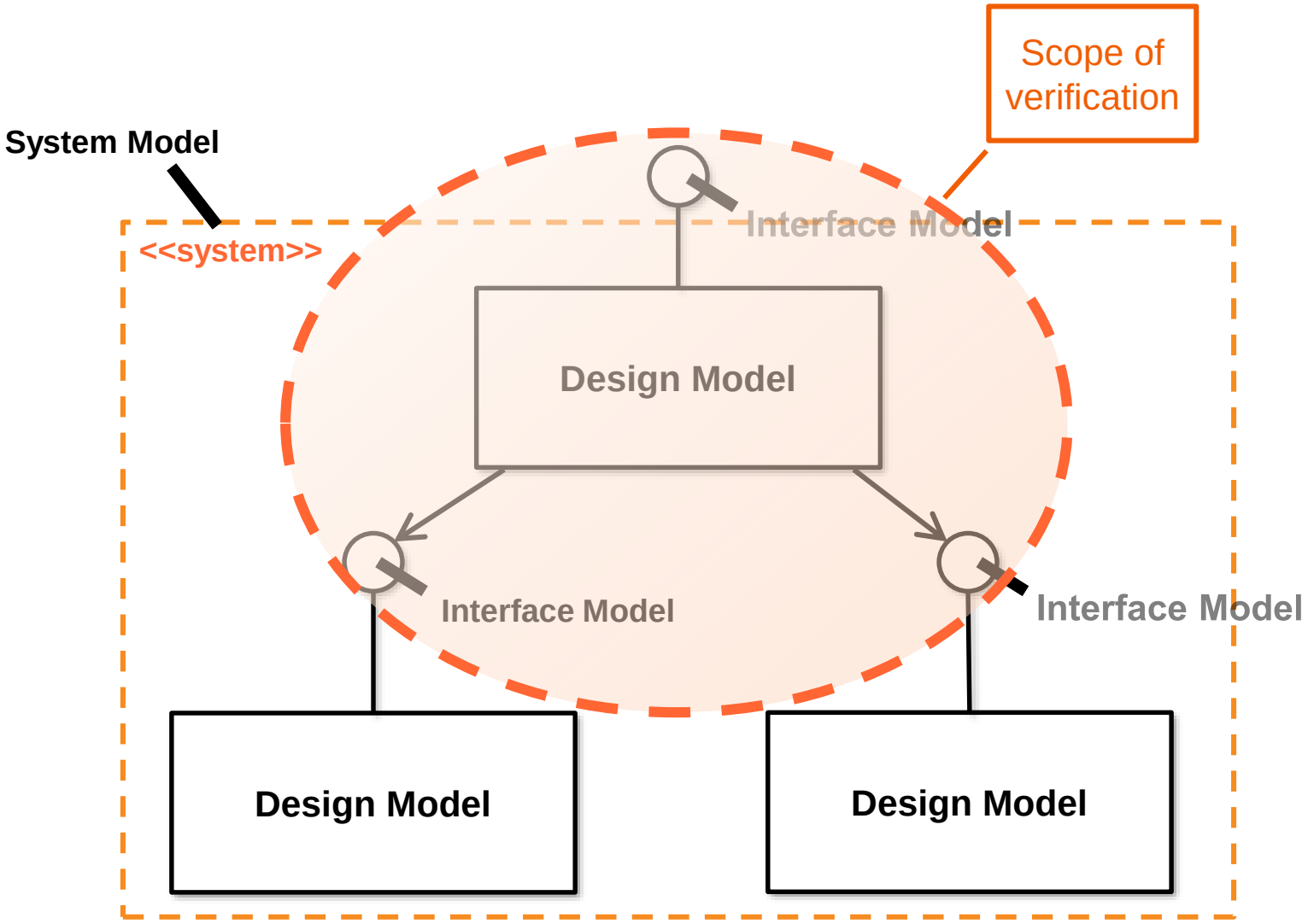



```
interface IMyInterface
{
in void RequestX();
out void SomeNotification();

behaviour { ... }
}
```

```
component MyDesignModel
{
provides IMyInterface api;
requires ISensor sensor;
requires IMotor motor;

behaviour { ... }
}
```

```
component MySystem
{
provides IMyInterface portX;
requires ISensor windowSensorY;

system {
  // instantiate all components
  MyDesignModel example;
  Motor motorZ;
  // interconnect
  portX <=> example.api;
  example.sensor <=> windowsSensorY;
  example.motor <=> motorZ.api;
}}
```

Model verification



The models are checked on:

- *Completeness*
- *Provided interface compliance*
- *Correct usage of used interface*
- *Live/dead-locks and race conditions*
- *Determinism*
- *Check on Illegals*

*By iterating all design models, the **entire system** is regarded modelchecked.*

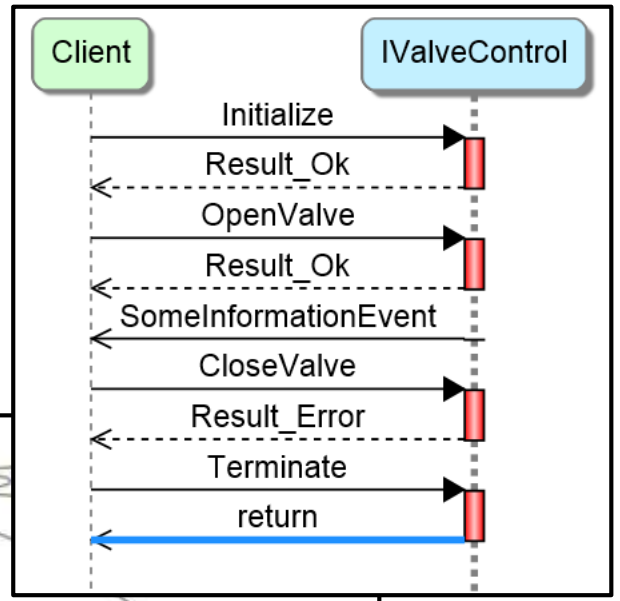
Example interface model in detail

```
interface IValveControl
{
enum Result { Ok, Fail, Error };

in Result  Initialize();
in void    Terminate();
in Result  OpenValve();
in Result  CloseValve();
out void   SomeInformationEvent();

behaviour
{
   enum State { Uninitialized, Idle, Error };
   State state = State.Uninitialized; // Set initial state

   on Terminate: state = State.Uninitialized; // Always allowed

   [state.Uninitialized]
   {
      on Initialize: {reply(Result.Ok); state = State.Idle;}
      on Initialize:  reply(Result.Fail);
      on OpenValve, CloseValve: illegal;
   }
   [state.Idle]
   {
      on Initialize: illegal;
      on OpenValve, CloseValve: reply(Result.Ok);
      on OpenValve, CloseValve: { reply(Result.Error);
                                  state = State.Error; }
      on optional: SomeInformationEvent;
   }
   [state.Error]
   {
      on Initialize, OpenValve, CloseValve: illegal;
   }
}}
```

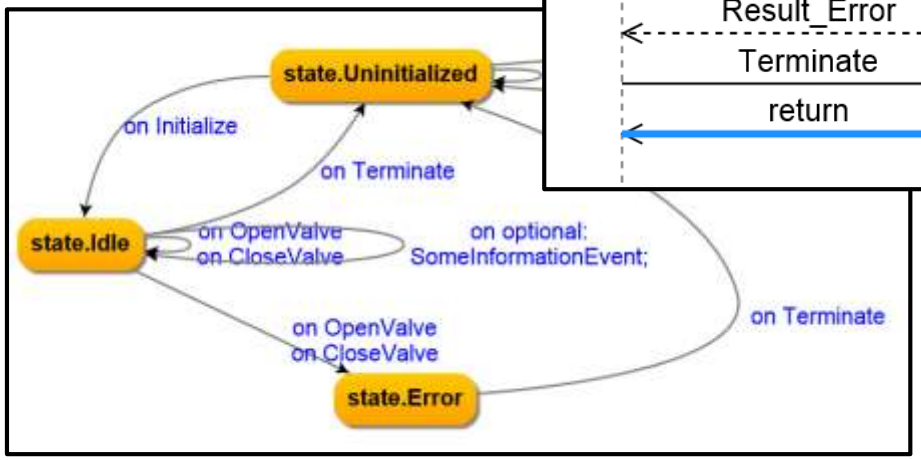
Semantics must be precise and complete. Also specify when events are **not allowed**

(by keyword 'illegal')

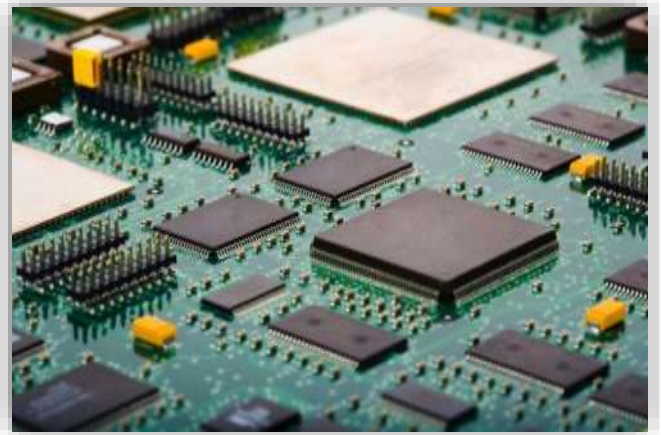
Sequence diagram view while simulating:



State chart view:



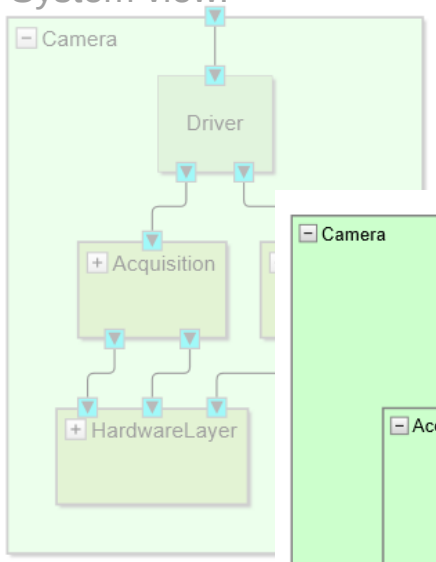
Creating a system (of systems)



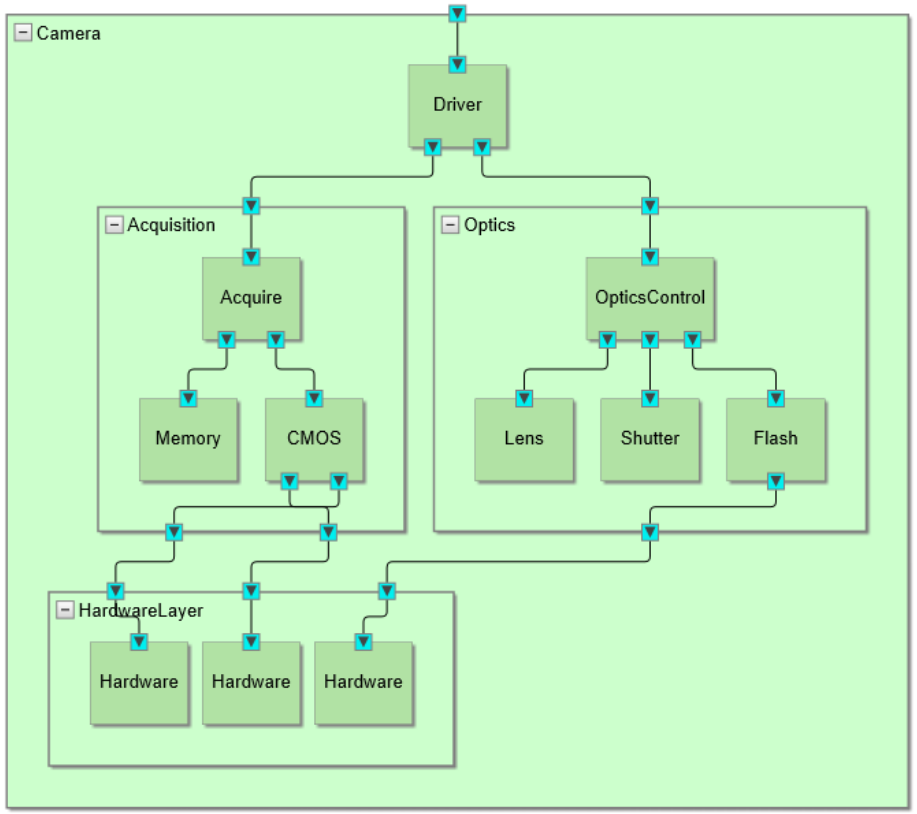
System model specification:

```
component Camera
{
  provides IControl control;

  system
  {
    Driver driver;

    Acquisition acquisition;
    Optics optics;
    HardwareLayer hardware;

    control <=> driver.control;
    driver.acquisition <=> acquisition.port;
    driver.optics <=> optics.port;

    hardware.acquire <=> acquisition.acquire_hardware;
    hardware.contrast <=> acquisition.contrast_hardware;
    hardware.flash <=> optics.flash_hardware;
  }
}
```

System view:



further expanded:



What isn’t ASD/Dezyne

- Not a tool that solves all problems
- Not a tool that does all the thinking for you
- Not suitable for systems that concentrate on data processing
- Not a tool to create a Domain Specific Language
  > *you are actually ‘programming’ from protocol point of view*
- Validation (functional correctness)
  > *although **Dezyne’s** roadmap presents items in collaboration with **TU/e**:*
  2018: Multi-component simulation & verification
  2019: System simulation & verification

The beneficial ASD/Dezyne effect

Measured since 2013

- Using ASD we reduced the **cost** on the total development project with ~35% on average
  - > *Including license costs*
- In maintenance phase the number of defects is extremely low (handful in the 1st maintenance year)
- Increased productivity
- Increased quality

Comparison: Dezyne and ASD

	Feature	Dezyne	ASD
Core Functionality	Automated Formal Verification	☺☺	☺
	Code Generation	☺☺	☺
	Sequence Trace	☺☺	☺
Customer Requirements	Open model description	☺	☹
	SysML Compatibility	☺	☹
Ease of Use	Text based Modelling Language	☺	☹
	IDE Integration	☺	☹
	Component Simulation	☺	☹
Applicability	Easy Legacy Integration	☺	☹
	Unified Threading Model	☺	☹
Value	System Architecture Definition	☺	☹
	(Executable) Trace Replay	☺	☹

My insight so far

> **Fundamentals**: Both ASD and Dezyne are based on the same core strength of formal verification and the effects on a project

> **Opportunity**: Dezyne offers room to extend these benefits

Start new project?

> ideal — maximize the benefits

Migrate ASD? Pick the right strategy:

> inefficient — continue on auto converted models
especially multi-threaded scheme

> effective — rebuild from scratch, side-by-side
backed by existing validation

See how far we already got!

Develop smarter

Traditionally

~ - ✓

no defects left?

~35% total project cost reduction

✓

defect free

~35 + ? % total project cost reduction

✓

defect free

Source of your technology