Critical software

Model-Based Fault and Safety Analyses of Complex Systems

Managing complexity using modular and digitalised processes

criticalsoftware.com
info@criticalsoftware.com

Setting the scene for model-based fault analysis

More components, more functions, more interactions – how can you keep increasing system complexity under control?

Now more than ever, the development of multi-layered critical systems requires a sound overview of all its components and a good understanding of the correlations, even beyond the boundaries of the product itself. For decades, modelling and simulations have helped to map and evaluate complex relationships. In the last decade, the term 'model-based system engineering' (MBSE) has become increasingly popular in system and function development, often combined with the SysML (Systems Modelling Language).

Nevertheless, text-based documents are still widely used by many development departments to describe their systems, enriched with flat graphics and managed in local files and folders. Often created on the basis of classic office software tools, these solutions reach their limits during the development of 'nominal' functions, their verification and unique traceability.

The situation becomes particularly critical when non-nominal behaviour is considered in analyses aiming to investigate and prove the functional safety and reliability of a system, especially in situations where human lives may ultimately be at stake.

"All models are wrong but some are useful."
George E.P. Box

There is a complex variety of operating states which must be captured and evaluated during risk analysis. These include numerous component types with individual fault modes, each used x-amount of times; irregular interactions, restructuring and reconfiguring topologies; the difficult interplay of software and hardware; strong linkage and interaction with other systems; the environment and human actors; and so-called 'hidden links'.

As the person responsible for the system, how do you ensure that - using such a variety of text reports written by many different authors - your safety conclusions adequately reflect the real hazard potential? How can you make sure this is done completely, without contradiction and, above all, in a way that is self-explanatory and easy to understand, even after the author of the text has moved on? Is valuable know-how on defects and their effects, once acquired, used in such a way that it brings advantages to the system operator, for example in terms of integrated maintenance concepts, high reliability, and good troubleshooting and diagnosis?

In this white paper, we will explore the role of 'classical' or analogue modelling in safety analyses, and then investigate some of the tool-based methods, including digital approaches to model-based safety analysis (MBSA).

Defining key safety analysis terms

Before we explore the model-based approach to fault analysis, let us first clarify a few terms:

Fault model: A mental, physical, mathematical, logical or other description of how a defined system element or component is likely to behave in the event of a fault. For each element, one or several fault cases (fault modes) are conceivable, each with its own behavioural description. It's important to note that the effect description in the system group is not part of the fault model.

Fault: An origin defect in a defined module or component within a system (red in Figure 1). If the module or component is repaired or replaced, this defect should disappear.

Error: A discrepancy between the intended behaviour of a system element and its actual behaviour within the system boundary (yellow in Figure 1). This is caused by other elements, not by the component itself.

Failure: A system function exhibits behaviour contrary to its specification (orange in Figure 1). An internal fault does not necessarily lead to a function failure. If, for example, fault tolerance techniques or redundancies are active, the overall functionality at the system boundary may well comply with the specification.

Symptom: A behavioural observation perceived at a higher system level or at the system boundary, which gives reason to draw conclusions about certain connections and states within the system (orange in Figure 1). The observation can be both in accordance with the specification and in conflict with it.

Root-cause: A possible system-internal explanation for a symptom, in particular a possible internal fault state (from one or a combination of several faults), the occurrence of which can provide a plausible reason for the observed behaviour at a higher level or the symptom pattern.

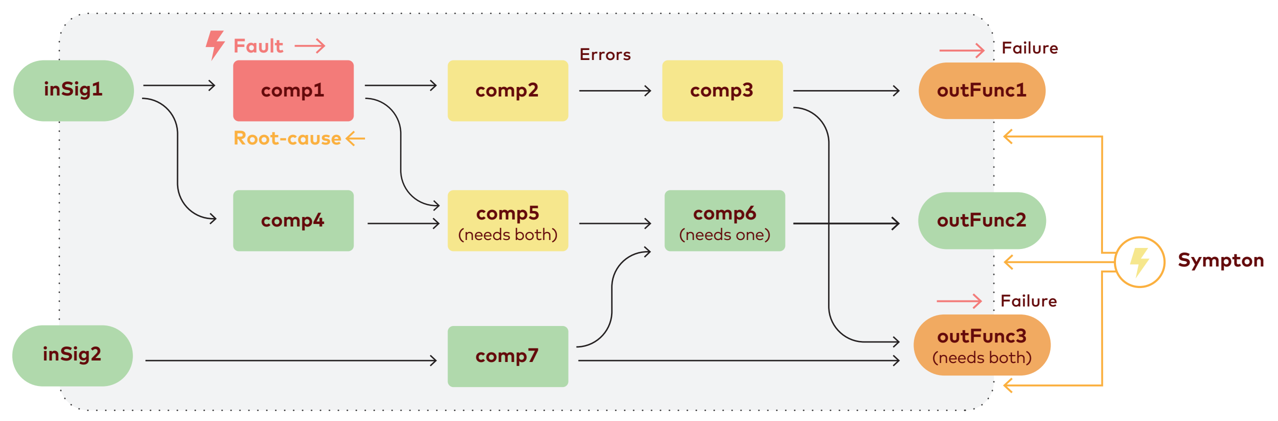


Figure 1 Exemplary networked system with errors and effects on functions

The 'classical' analogue way

Terms such as 'modelling' or 'model-based development' often trigger thoughts of advanced, computer-based virtualisation methods leveraging state-of-the-art software tools, which tend to be used to develop complex embedded systems.

In fact, the term can be understood much more broadly, and incorporates some classical analogue means to model safety-critical systems. Here are a few examples:

Figure 2 British wooden horse simulator to test procedures and process faults in combat, taken before 1915
https://commons.wikimedia.org/wiki/File:Horse_simulator_WWI.jpg

A) PHYSICAL MODELLING

Probably one of the oldest modelling methods is to reproduce real correlations by specific imitation on a physical replica. This replica can be used - often on an adapted scale or in some other simplified form - to carry out experiments in a controlled environment and in a way that is not feasible in real life. Architectural models of buildings or laboratory set-ups of technical plants are typical examples of physical modelling used to investigate static or dynamic behaviour and to draw conclusions about the real-life counterpart that may still need to be built, as well as determining possible optimisations.

In principle, this can also be used to test the effects of non-nominal behaviour by 'injecting' certain faults into the replica. For this purpose, the type of fault must be known with regard to its basic behaviour and its possible effects. For example, in a test setup focusing on the electrical supply of an aircraft, faults such as wire cuts and their effects should be relatively easy to test. More difficult in this example would be the deliberate provocation of short circuits between different wires, which could lead to the destruction of model parts (for example, real subsystem modules or boards) or even result in the loss of the entire setup.

Although using a physical model makes certain experiments possible, the creation of the model is time-consuming and costly, notwithstanding the fact that tests must be extremely well-planned.

B) MENTAL MODELLING

But even without building a physical prototype of the product to be realised, we are much closer to the idea of modelling than we often think we are. The reason is that even the systematic mental capturing of dependencies, with the goal of obtaining a better understanding of the system in question, is a form of 'modelling'. Within an assumed boundary, by means of known information or a priori knowledge, we picture certain elements, internal links, processes and connections from which we can conclude the behaviour of the whole system. We simulate 'mentally' to draw conclusions and benefit from this increase in savoir-faire during system development.

Indeed, this variant of the 'model-based methodology' can be found in many projects, often as an interlinked process. On the basis of preceding documents, the expert forms a model of the system to be developed in their head and, from the current level of maturity, analyses and develops it a little more using their own knowledge. Finally, the results of their work are formalised in a document.

A common form of safety analysis based on mental models is the failure mode and effect analysis, or FMEA. By hypothesising a given functionality within a complex system network, possible effects or failure scenarios at the highest system level are inferred in a structured way for one assumed internal fault after the other, as part of a multi-disciplinary team approach. The analysis direction is inductive, i.e. according to the causality from cause to effect. The analysis is also carried out from the bottom up in the system hierarchy, i.e. from the components up to the system level.

The opposite way of thinking – from the observation of failure or irregular symptoms at the system boundary down to possible root causes, is frequently found in fault analysis practice. This way of thinking can be seen in the diagnosis of technical systems, for example by experienced mechanics in the workshop when identifying a component needing to be replaced. In doing so, they may use previously elaborated and defined troubleshooting procedures in the form of decision trees (DT).

No matter how easy or 'tool-free' the mental modelling method used is, this advantage can also become a major disadvantage: the know-how of experts is inherently linked to the individual and its transmission to others in the organisation, and its effectiveness when said expert is no longer present, depends on the quality of text-based documents and charts.

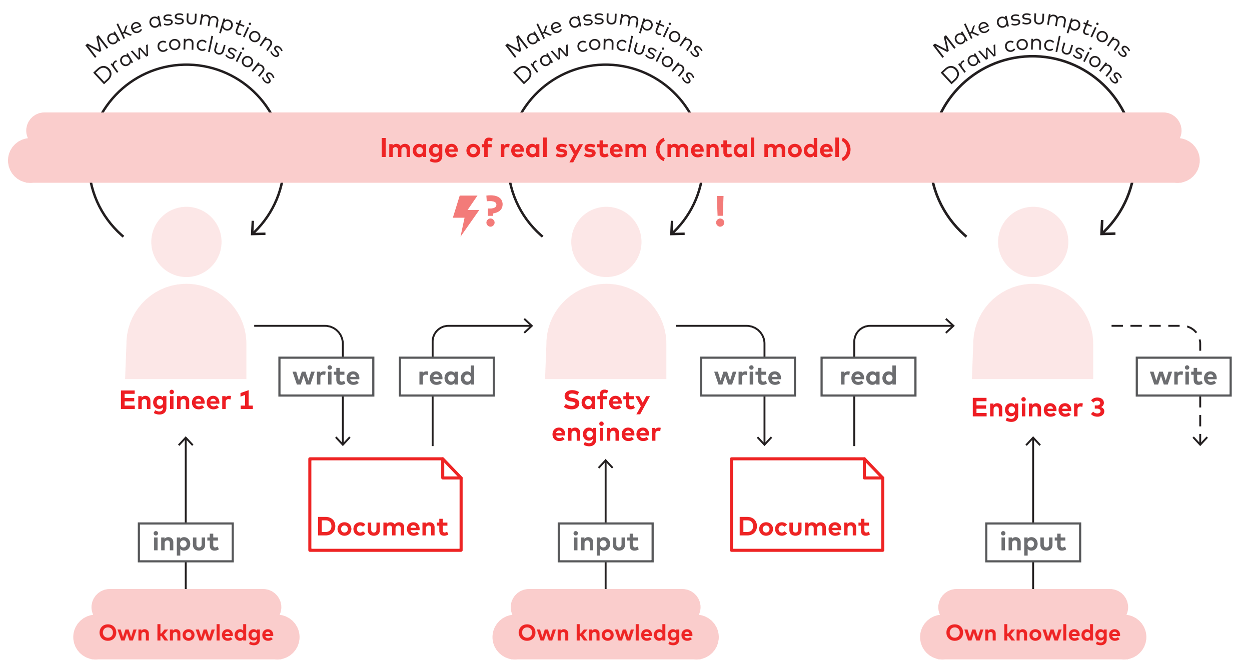


Figure 3 Chain of mental modelling steps in system development, e.g. for fault analysis.

C) GRAPHICAL MODELLING

The individuality of mental modelling naturally demands supporting documentation so as to properly visualise the model in question. That is where graphic illustrations, schema or notations come in, which should help to make the author's train of thought more comprehensible to others. The motto 'one system, many views' can be used to sum up this concept. By focusing on selected aspects of the real system, appropriate simplifications and standardised design elements and symbols, it is possible to create individual graphical models of reality. SysML users are familiar with this in the form of the various predefined diagram types.

Be it SysML notations, drawings, block diagrams, process diagrams, electrical or hydraulic plans of a machine, illustrations make it possible to better understand and discuss the dependencies which exist between parts of a system and also help to simulate them mentally. In addition, the systematic presentation and naming of the pictorial elements should be considered, this being key to ensuring clear and unambiguous communication with stakeholders.

As well as reliability block diagrams (RBD), fault trees (FTs) are probably the best known form of graphical modelling in the fields of safety and reliability. The assumed failure logic of the system is visualised in a tree-like structure, deductively determined by the development team from the top event down to possible root causes or combinations thereof.

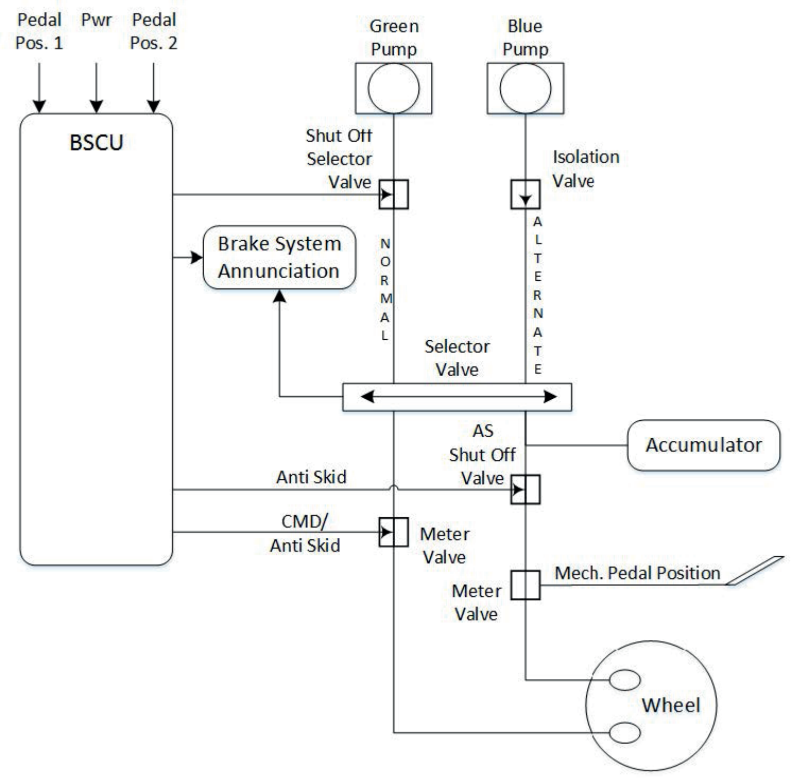


Figure 4 System diagram, example from ARP4761

While only graphically constructed from blocks for events and for Boolean operations, FTs nevertheless allow a quantitative calculation of the top event's probability in relation to failure probabilities in relation to the failure probabilities of the most basic components or nodes.

Graphical models support system and safety development across the entire process, from the discussion of alternative architectures in an early design phase (in the context of a preliminary system safety analysis according to ARP4761, for instance) and the derivation of safety requirements, all the way down to the validation of the final system as built (like the numerical proof of the overall probabilities of top hazards).

However, despite their effectiveness when compared to other forms of modelling, FTs still possess one weakness. Conclusions from models and the assumed behaviour of the components therein are made in the human mind, based on individual and possibly undocumented assumptions. Thus the quality, traceability and persistence of the results ultimately depend on the individual expertise of the specialists creating these graphs and their availability within the company.

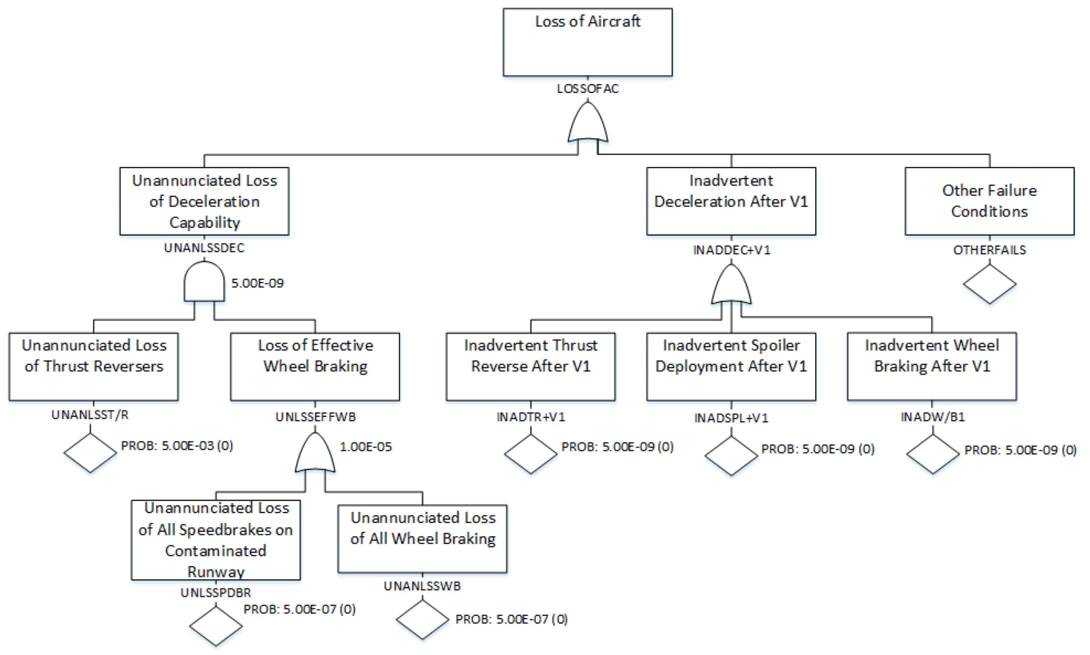
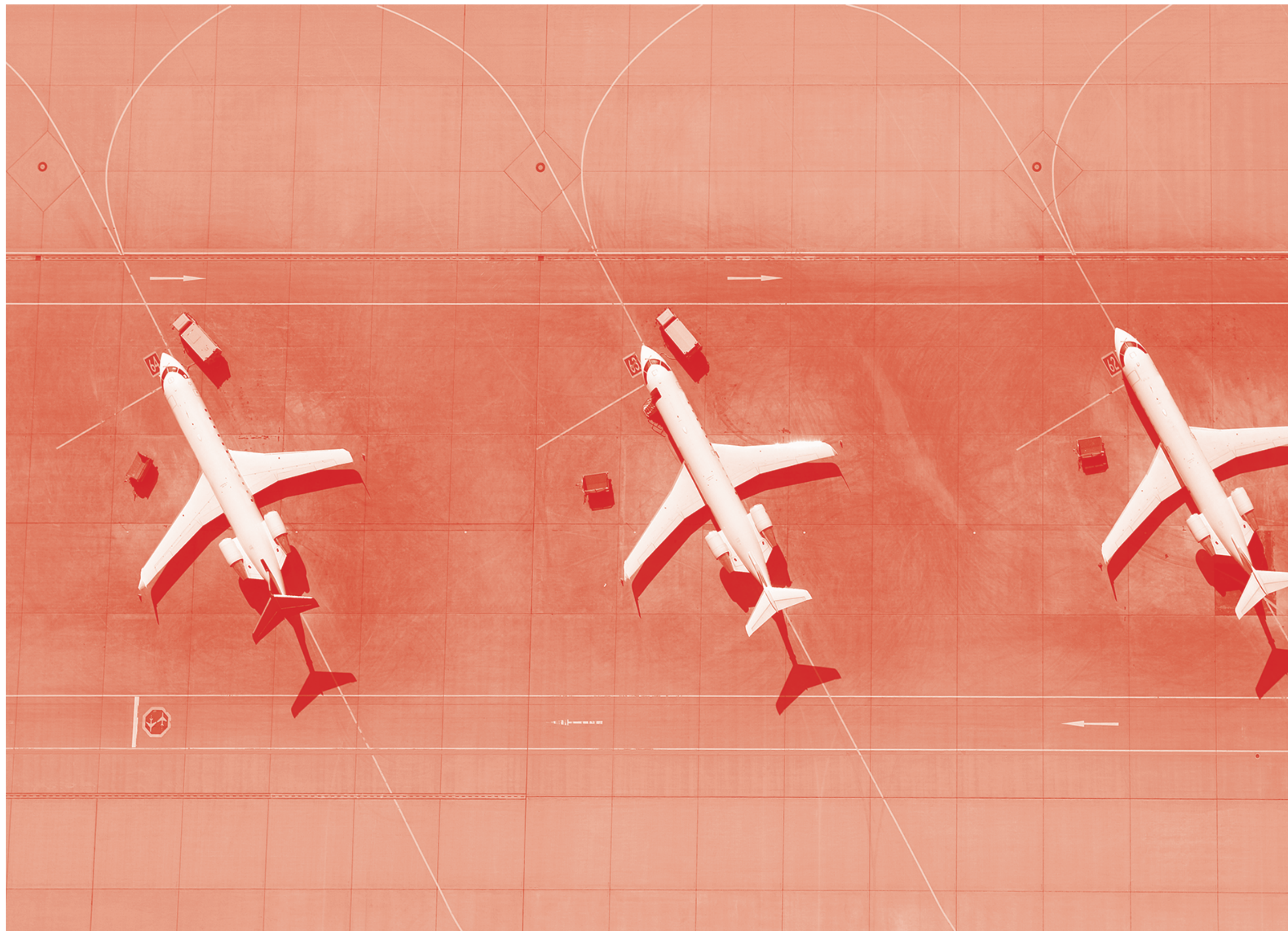

Loss of Aircraft
LOSSOFAC
Unannunciated Loss of Deceleration Capability
UNANLSSDEC
5.00E-09
Inadvertent Deceleration After V1
INADDEC+V1
Other Failure Conditions
OTHERFAILS
Unannunciated Loss of Thrust Reversers
UNANLSST/R
PROB: 5.00E-03 (0)
Loss of Effective Wheel Braking
UNLSSEFFWB
1.00E-05
Inadvertent Thrust Reverse After V1
INADTR+V1
PROB: 5.00E-09 (0)
Inadvertent Spoiler Deployment After V1
INADSPL+V1
PROB: 5.00E-09 (0)
Inadvertent Wheel Braking After V1
INADW/B1
PROB: 5.00E-09 (0)
Unannunciated Loss of All Speedbrakes on Contaminated Runway
UNLSSPDBR
PROB: 5.00E-07 (0)
Unannunciated Loss of All Wheel Braking
UNANLSSWB
PROB: 5.00E-07 (0)

Figure 5 Aircraft FHA Preliminary Fault Tree [SAE ARP4761, p. 182]



The 'executable' digital way

This is where actual executable digital models come into play. They allow for the reproduction of the respective behaviour of the real system via computer-aided simulation. This technique has been taught for many decades and is well established in engineering, but so far has mainly been used to represent and analyse nominal behaviour.

The relevant variables, parameters and constants of units and functionalities – within a wider framework of assumptions - are captured in a modelling language and put in relation to each other by equations, state machines, XYZ-characteristics or other analytical forms. In many tools supported by graphical user interfaces, the specified input-output connections between the individual model components create an integrated and executable mathematical model of stationary or dynamic behaviour.

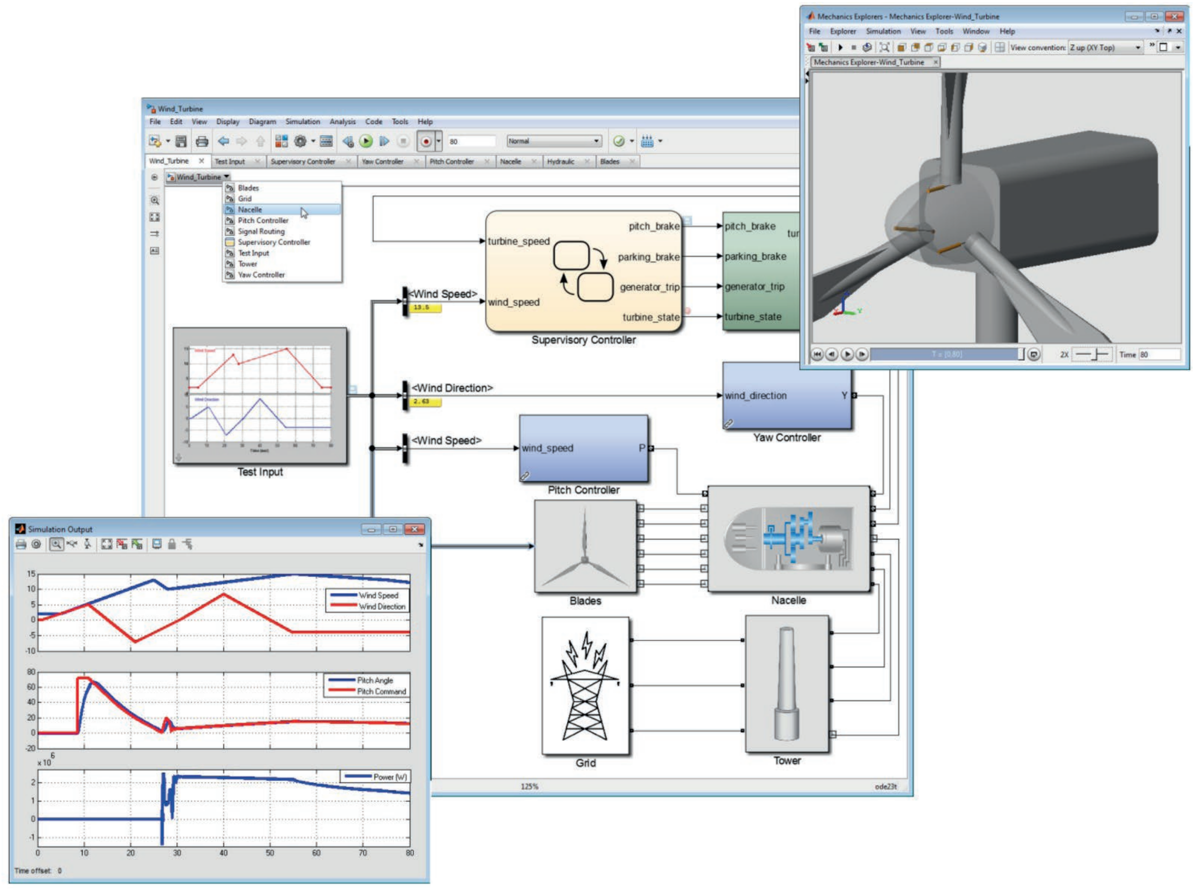


Figure 6 Examples of Simulink models (*https://commons.wikimedia.org/wiki/File:Simulink_model_of_a_wind_turbine.tif*)

A) CAUSAL BLOCK-ORIENTATED MODELLING

Modelling principles which follow the assumed functional chains of the system are collectively known as 'causal modelling', otherwise referred to as 'block-oriented' or 'signal-orientated' modelling. Here, the descriptions inside the model blocks express the clear dependence of the output variables on the input variables of each unit. To better visualise this, the inputs are often located on the left and the outputs on the right in graphical representations or icons of the model parts. The graphical arrangement of the blocks as well as the implicit evaluation is directional.

A popular representative of this modelling approach is Simulink. The system variables previously determined by system analysis, as well as the differential and algebraic equations (DAE) used to describe the behaviour, are assembled at the lowest level from the most basic mathematical operator building-blocks. Grouped together and provided with defined interfaces ('ports'), this results in reusable new blocks which are then used to build the entire mathematical description model of the real system.

After specifying adequate initialisation values and input profiles, the calculation-system can then be solved by the simulation algorithm according to the analytical data-flow, and the numerical values or time histories of previously unknown system variables can be displayed. For many engineering analysis tasks, this type of modelling has been common for decades, and many reusable cross-project model libraries have subsequently been developed.

But what about failure and safety analyses? How does this model help us to better understand the relationship between local defects and their system-wide impact?

You can only analyse by simulation what has been modelled before. The description of the nominal behaviour must therefore be extended to include models of faults. **This raises some fundamental questions:**

1. How can fault modes of components be organised within the model?

Faults in real life are often assigned to particular components and have a specific denomination ('disconnected', 'stuck open'...). Therefore, in addition to the 'intact' behaviour of a system component, alternative descriptions for one or even several different fault modes must be considered in the simulation model, including a suitable selection technique.

2. What about the native structure of the model equations?

Some fault modes, including purely parametric ones, can easily be considered in the existing causal framework, for example, a numerically flawed sensor characteristic or a 'bit flip'. The causal chain is basically retained. Other system faults, however, can impact the entire structure of the DAE-system previously established for the nominal behaviour. For instance, if irregular topological changes (e.g. a short circuit in the electrical network) result in unforeseen flows and new balance terms, a significant model modification would be required [Ref01].

3. Is the assumed causal chain of calculation still valid?

Maybe, maybe not. For many real-life faults or fault-combinations that we want to analyse, the global input-output behaviour assumed in the nominal model is no longer valid. For example, tube leakage can locally reverse the direction of flow, or an erroneously powered or insufficiently safeguarded electric motor can induce unexpected voltages in the network contrary to the assumed normal functional chain. This can mean changes to the model in question are necessary. Or, imagine a fluid-valve hanging closed: here it is not enough to just assign a value of "zero" to the output pressure.

Regarding safety, with all these challenges in mind, the causal modelling principle therefore raises challenges in the quantitative fault analysis of real components. Such an approach of numerical simulation can therefore be best applied to the modelling of traditionally procedural control logic and the development of software-related diagnostic functions or mitigation and backup strategies.

Beyond numerical simulation, however, causal input-output models can nevertheless be helpful in maintaining a view of the overall picture when developing safety-critical systems, including hardware.

The management of many internal dependencies existing between requirements, system functions, envisioned solution principles and allocation to architectural elements on different levels is a demanding task in itself, not to mention the assignment of adequate tests. With classic, document-based procedures, the limits can quickly be reached, engineering mistakes creep in, work products no longer fit together and the necessary traceability is lost.

Such problems can be avoided by model-based tools which are integrated over the entire V-process, with central data management for requirements, functions, technical solutions, components, interfaces, SW code, documents, amongst other aspects of the system.

Throughout the development process, modelling allows for the visualisation of safety dependencies in case of failure. For instance, with the ESCAPE technology, it is possible to 'inject' a fault at any location in the network and, at the click of a button, to determine and indicate system parts potentially affected by it, just like in a FMEA, 'downstream' (Figure 7). In the opposite direction, the failure of the toplevel functionality can be specified and the possible root causes of this failure identified, just like in a diagnosis, 'upstream' (Figure 8). The advantages and disadvantages of certain architectures can be evaluated and weighed against each other in the early design phase, and forgotten links across all system levels, funtionalities and hardware units within the entire system can hence be uncovered [Ref02].

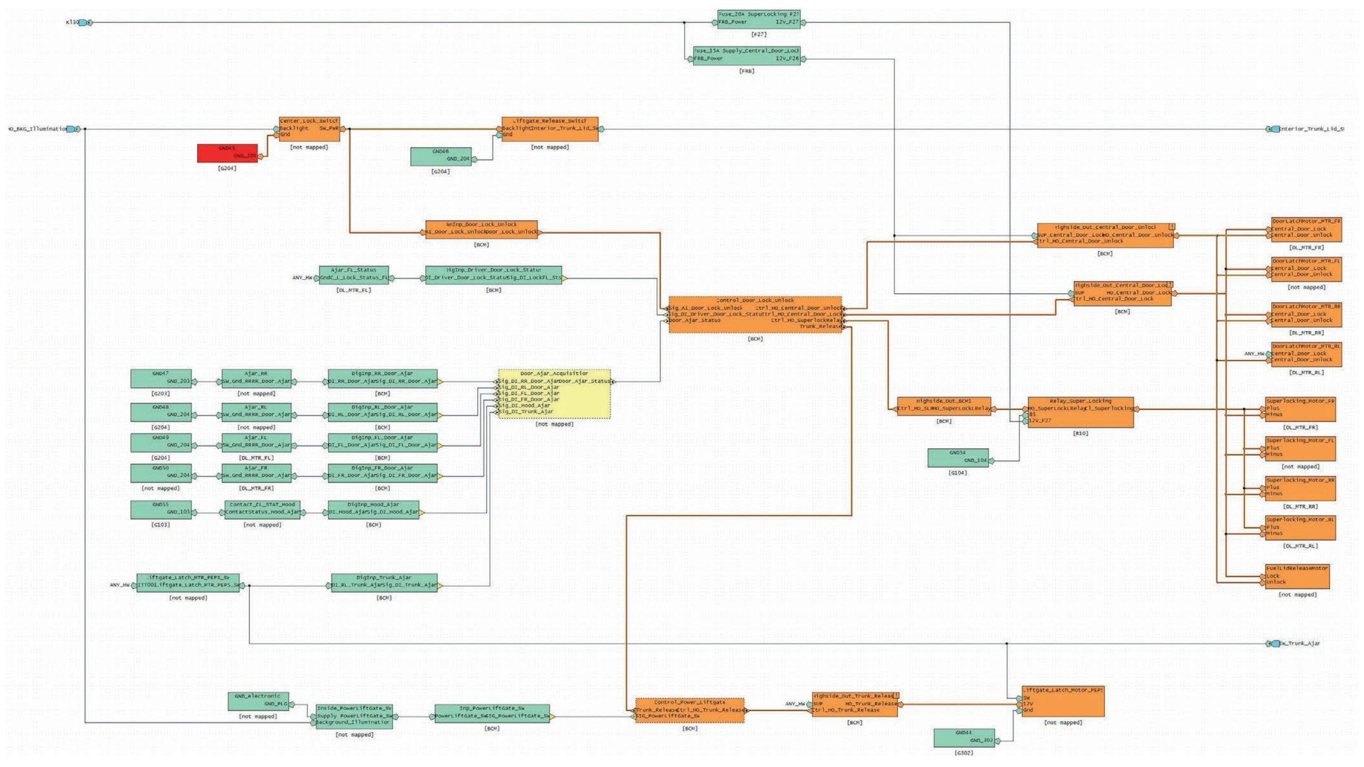


Figure 7 Model-based fault analysis within the dependency network: a) Presetting ("injection") of local faults (red) and identification of the affected system extent and interfaces (orange)

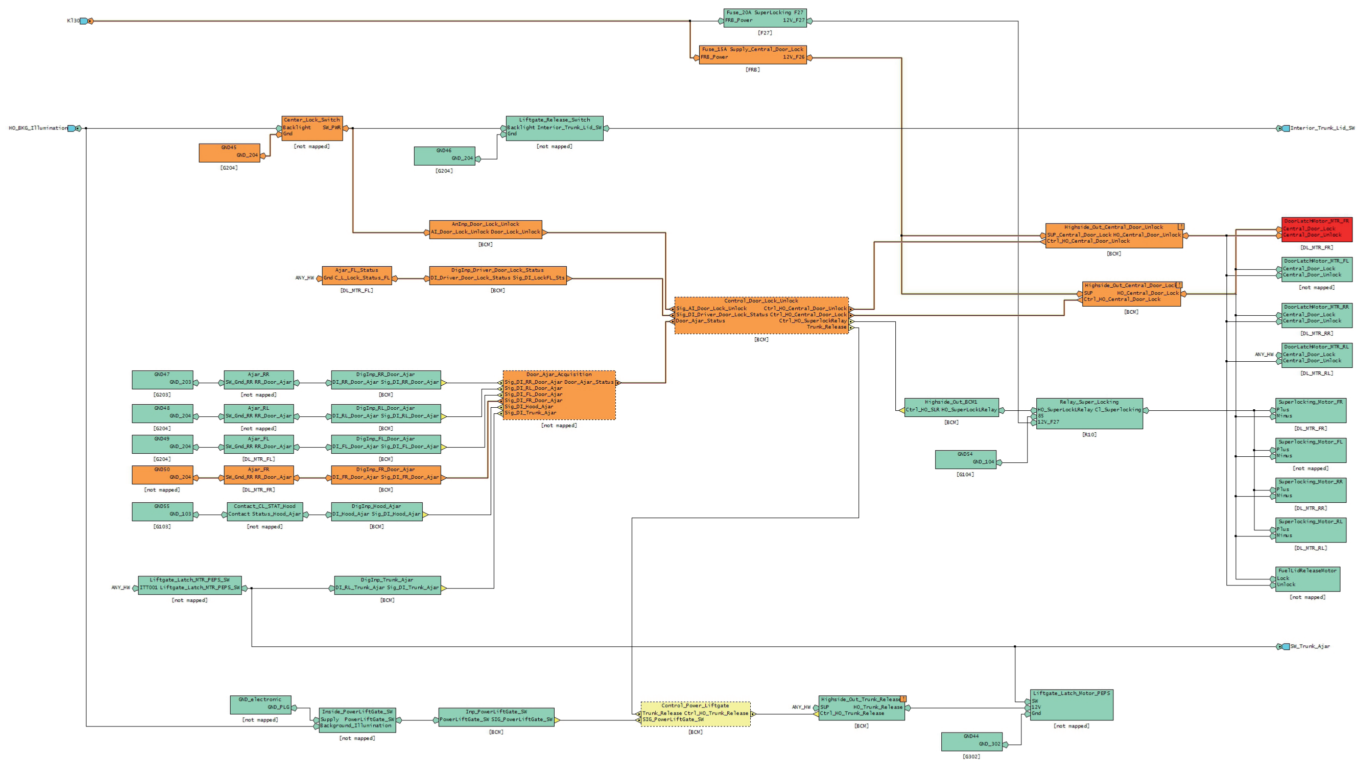


Figure 8 Model-based fault analysis within the dependency network: Presetting a failure case at the interface (red) and identification of possible root causes in the system network (orange)

In another category of modelling tools, the focus is less on the safety analysis of a system, but rather on a formally correct and verifiable methodology for the development of safety-critical software. Due to the functional nature of software systems, the causal modelling paradigm is also applied here. The SCADE tool, for example, enables a consistent representation of state machines and the internal data flow, which is used to generate software code for applications with the highest safety requirements.

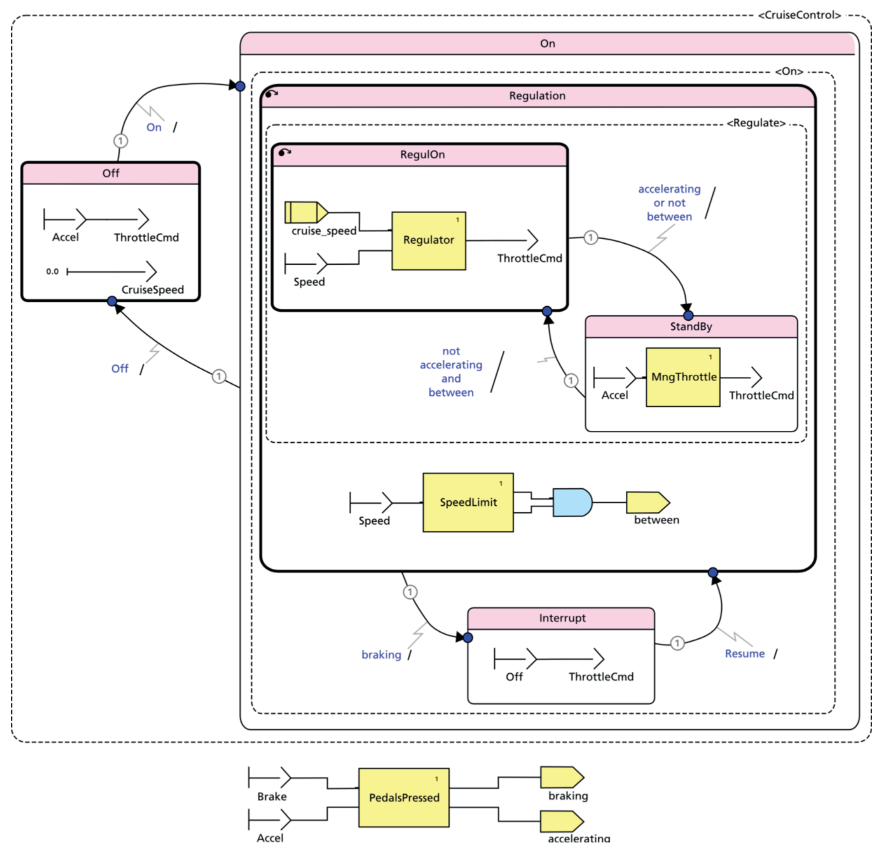


Figure 9 Model-based development of safety critical software in the form of a cruise control system with SCADE (https://commons.wikimedia.org/wiki/File:SCADE-cruise-control-design.png)

B) ACAUSAL PHYSICAL SYSTEM MODELLING

The need to assume a certain functional chain and to consider the calculation direction in advance is undoubtedly a handicap of causal modelling. So, would the model be more successful if it were based solely on a real physical system structure describing it one-to-one, incorporating components, interfaces, topologies and hierarchies?

For this declarative representation, the paradigm of object-orientation is useful: each model element is an image of a corresponding type class. Regardless of whether it stands for a simple variable data type such as *Current* with an assigned *Ampere* unit; an interface (port) such as *Pin* with several internal variables, a component such as *SolenoidValve*, each with two interfaces for the electrical and the hydraulic part or even a complex overall system, such as *DialysisMachine* or *JetEngine* - every attribute in the model originates from a well-defined class, as if it were a blueprint. This is the principle of the Modelica language [Ref03], with a wide range of applications in automotive, avionics, aerospace, robotics, power engineering and even non-technical systems from bio-mechanics to medicine. Various tools based on this principle have been established.

The language offers us an interesting perspective in that we no longer need to think about data flow direction. The evaluation algorithm takes care of whether the variable within an interface serves as an input or output variable in a particular analysis procedure. The model is therefore 'acausal'.

This way, all knowledge about the internal structure and the behaviour of a real physical element is described and stored in one place, encapsulated and accessible only via the real-life counterparts (electrical, mechanical, hydraulic, bus) interfaces. Thus, we achieve an easily understandable and navigable model structure, which could also have originated from an E-CAD tool (Figure 10) or be derived directly from the well-known bill of materials (BoM) in engineering.

This clarity in model visualisation helps increase system understanding during the development phase, avoiding mistakes and easily linking the modelling to PDM or QM standard processes, including part-orientated version management. It is very intuitive and easy to extend and use the model for different types of analyses without having to change the internal topology. The know-how investment in the model is secured, independent of employee fluctuation. All that matters is the algorithm.

With regard to fault analyses and RAMS analyses, the questions previously posed to causal modelling can be answered easily here: everything that belongs to a certain type of component is stored in the corresponding model class. This includes fault mode names, alternative local behaviour models, and even specific parameters such as fault probabilities. This allows for the activation of single or multiple faults within the system model and, using simulation, determine their effects in all directions - the FMEA may be automated.

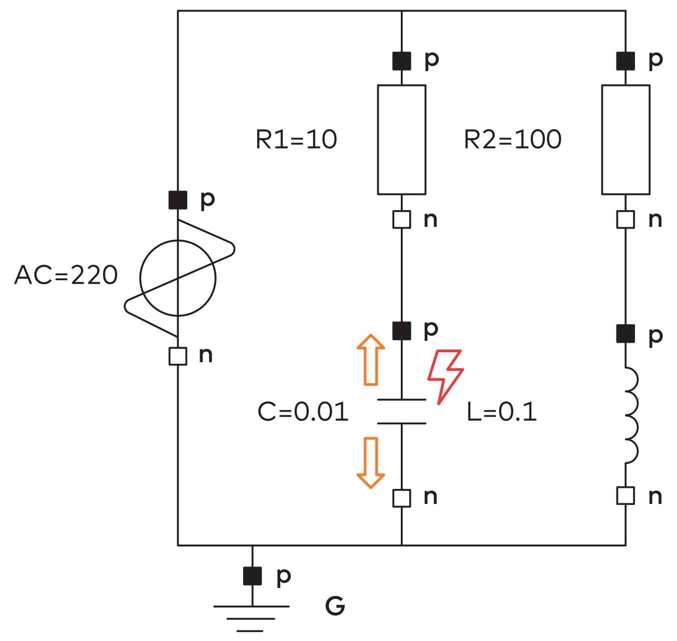


Figure 10 Physical system model of an electrical circuit (from: [Ref03]); bi-directional fault effect

Similarly, the acausal 'two-direction' paradigm allows for the evaluation of the same model in the opposite direction. Values may be assigned to model variables at any location in the system model, even if they are inconsistent with the nominal system behaviour. This observed behavioural pattern constitutes a 'symptom' (see Figure 1) that can be used to deduce automatically possible root causes with the help of an algorithm for so-called 'model-based diagnosis' [Ref04].

This way, our virtual playground also helps us to optimise the internal FaultCode-/BITE-concept resp. diagnostic coverage (DC) at an early stage of the process.

Whether through industry-proven model-based applications for diagnosis, monitoring and RAMS analysis; formal languages for safety-analyses like AltaRica [Ref05]; or through new ideas like smartIflow applied for example to rail systems [Ref06], there is no question that model-based safety analysis has in itself become an exciting area of development within engineering, not only in terms of modelling but also as a methodology and its effects on personal mindsets brought to engineering projects.

The promise of model-based analysis

'Model-based' analysis is a promising way to maintain a full system overview and to make valuable engineering know-how usable in a digital and, hence, more sustainable way.

Modelling may seem tedious and time-consuming at first. Yet this investment of time and resources pays off in several ways. By obtaining a better understanding of the system and its failure dependencies, implementing faster design iterations, and by optimised backup and mitigation strategies, modelling done well can truly revolutionise the way systems are run and kept safe.

Embedded in a seamless process chain without 'media breaks', the model becomes the central knowledge reference and foundation from which a reliable process for the development of safety-critical systems can be derived. This can serve as an 'executable specification' and can save a lot of V&V effort when used as a virtual testbed.

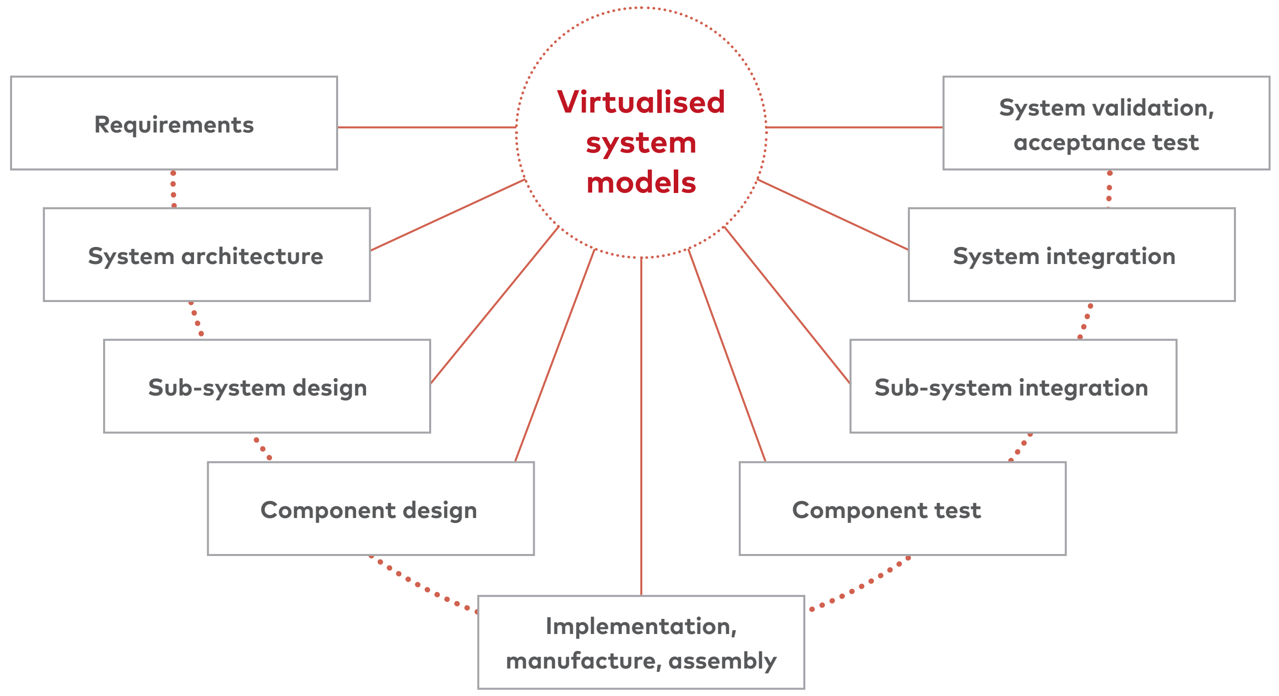


Figure 11 The virtual system as a central reference point in the system development process

The Critical model: how we can help

At Critical Software, we have experience in using modelling methodologies to guarantee the effectiveness and security of mission-critical systems in a range of industries, from automotive and rail to space and aerospace. The software validation facilities we provide can easily accommodate the comprehensive testing, verification and validation required for safety-critical systems across the board.

Our over 20 years' experience of not only developing, but testing and verifying complex and multi-faceted systems has given us the opportunity to innovate. By using the Agile methodology to develop, test and verify high-integrity systems, as well as more traditional methodologies like waterfall, we provide a dynamic approach to developing and validating critical software which places the customer at its very heart.

References:

[Ref01] Joshi A., Heimdahl, M.P.E., Miller S.P., Whalen M.W., Model-Based Safety Analysis, NASA/ CR-2006-213953, University of Minnesota, Minneapolis, Minnesota/ Rockwell Collins Inc., Cedar Rapids, Iowa; https://shemesh.larc.nasa.gov/fm/papers/Joshi-CR-2006-213953-Model-Based-SA.pdf

[Ref02] J. Kaiser, G.F. Soto, X.Tang/J.Li/ G. Guo/C. Wen, J. Brandscheid, EXCEED: Integrated, model based development of the E/E-System of CHERY''s new vehicle platform, Day of System Engineering, TdSE 2017, Paderborn, Germany; http://www.3e-motion.com

[Ref03] P. Fritzson, Introduction to Object-Oriented Modeling and Simulation with Modelica Using OpenModelica, Open Source Modelica Consortium v2012; https://www.modelica.org/publications

[Ref04] Fasol D., Münker B., Bunus P., A Model-Based Safety and Dependability Methodology for Missile Safety Engineering, International System Safety Society Congress ISSC2015, San Diego; https://icomod.com/mbsa_ISSC2015

[Ref05] Prosvirnova, T., Batteux, M., Brameret, P.A., Cherfi, A., Friedlhuber, T., Roussel, J.M., and Rauzy, A., The AltaRica 3.0 project for model-based safety assessment. IFAC Proceedings Volumes, 46(22), 127 – 132. doi: http://dx.doi.org/10.3182/20130904-3-UK-4041.00028

[Ref06] Lunde R., Hönig, P., Müller C., Reasoning about Different Orders of Magnitude of Time with smartIflow, University of applied Sciences, Ulm, Germany; https://smartiflow.bitbucket.io/

To find out more about our work, please get in touch: info@criticalsoftware.com

We are CMMI Maturity Level 5 rated.

For a list of our certifications & standards
visit our website.

criticalsoftware.com
info@criticalsoftware.com