Fast Augmented STPA

Odd Ivar Haugen[1,*] and Børge Rokseth[2]
[1]DNV GL, Group Technology & Research, Trondheim, Norway
[2]Department of Marine Technology, Norwegian University of Science and Technology, Norway

ABSTRACT

All elements (agents) in the STPA control structure (control algorithm, actuator, sensor system, process model) consist of a set of functions. These can be visualised and analysed using the Functional Analysis System Technique (FAST). The control action is executed by the *control algorithm* agent. By using FAST we can analyse the sub-functions of the control action and identify scenarios that may cause unsafe control actions. In the same way, the actuator agent, sensor agent and the process model agent can be visualised and analysed through FAST to identify scenarios that may cause unsafe control actions. When identifying scenarios that may lead to unsafe control actions, analysts tacitly create a mental model of these dependencies. One of the strengths of STPA is in *agent analysis*, by identifying the system agents responsible for enforcing safety constraints as well as other agents whose actions (or lack of them) may cause unsafe control actions. The strength of FAST is *function analysis* through making the functional dependencies explicit. Small FAST trees within the STPA control structure increase the information density without creating too much clutter. The semantics in FAST are relatively easy and quick to learn for Subject Matter Experts (SMEs) and others. FAST trees can guide refinement of the control structure by identifying functions as new lower-level or higher-level control actions that need further investigation in new control structures. The original purpose of FAST was to spark the creativity to find an alternative solution to a problem, or alternative ways of achieving a function. This is valuable early in the concept and design phase of any system development, including when using STPA in early system safety engineering phases.

Keywords: STPA; Functional Analysis System Technique; FAST; Function analysis; Agent analysis.

1. INTRODUCTION

It Identifying scenarios that may lead to unsafe control actions requires profound knowledge about the system under consideration, the technology used and of the intended operation, and the environment of which the operation is conducted. This paper suggests augmenting the System-Theoretic Process Analysis (STPA) introduced by Leveson (2011) by using Functional Analyse System Technique (FAST) to assist in the identification of causal scenarios leading to unsafe control actions. Moreover, FAST may also be used by the analyst in the refinement of the control structures.

FAST was first described by Charles W. Bytheway in 1964 and presented as a paper to the Society of American Value Engineers conference in 1965, and later published in the book of Bytheway (2007). An introduction to FAST is given in the Appendix of this paper, and further information can be found in the Society of American Value Engineering (SAFE)†. In FAST, a functional relationship is developed by repetitive asking two questions: *How* a function is accomplished, and *Why* a function must be accomplished. The answers are formulated as

* Corresponding author: +4791715040, odd.ivar.haugen@dnvgl.com
† https://www.value-eng.org/

functions together with their relationship. The How/Why relationship is also described by Rasmussen, Pejtersen, & Goodstein (1994) within the means-end relationship. Both work mentioned above describe how we can move up and down in a hierarchy of abstraction levels, which is a well-known strategy for dealing with complexity and human problem-solving. This aligns with Leveson (2011):

> *"To create causal scenarios, the control structure diagram must include the process models for each component. If the system exists, then the content of these models should be easily determined by looking at the system functional design and its documentation. If the system does not yet exist, the analysis can start with a best guess and then be refined and changed as the analysis proceeds. (p. 221)"*

FAST is a way of visualising, structuring and analysing the "*system functional design*", and a help in providing a "*best guess*" early in the system concept and design phases. If STPA is performed on a "black-box" system, perhaps by a third-party analyst who is not granted access to the internal system design documentation, she/he can use FAST to identify the required generic functions, and then perform the analysis.

2. APPLYING FAST IN STPA

Although FAST makes functional dependencies explicit, it lacks strong semantics for temporal relationships between functions (i.e. identifying the *supporting* functions in FAST will give some temporal information), and the ability to analyse agency. Agency is central to STPA through the identification of the controller and thereby the control algorithm - that is instrumental for the accomplishment of control actions. Temporal causes of hazards are also part of STPA through some of the guidewords. Hierarchical functional relations, however, can be modelled in other ways, like FAST.

The control algorithm in the controller is basically a set of functions and sub-functions. When identifying scenarios that may lead to unsafe control actions, the control algorithm is one place to investigate. Sometimes, the mechanisms of the control algorithm may be well known to all participants in the analysis, and there may be no need to explicitly state these functional dependencies. However, other times it would be beneficial to explicitly depict these dependencies to make certain that important aspects are not missing from the analysis, and to inform non-expert team members about these relationships.

The "output" of the control algorithm is the control action(s). Control actions are functions executed by agents that have the ability to control systems states that are important for safety, or in other words: "...*enforce constraints on the behavior of the controlled process*". (Leveson & Thomas, 2018, p. 22). What we want to do is to investigate the functions executed by the agents in the control structure (i.e. control algorithm, process model, actuator, sensors) that can cause Unsafe Control Actions (UCAs).

Figure 1 shows a dynamic positioning control system (DP control system), where the control action is the command of force through the use of the thruster system. The hazard to avoid is losing the ship's position and heading. For safety-critical operations, this is seen from regulatory bodies as the hazardous system state (International Maritime Organization [IMO], 2017), or in other words: remain at the location where you are supposed to be located. The ship is to be designed in such way that it will remain stationary in case of a single fault. The standard means of achieving this goal is component redundancy. Of course, component redundancy is not seen as an adequate requirement for safety in complex software-intensive systems (Leveson, 1995; Ericson II, 2013), however, this is the current focal point in (IMO, 2017), where FMEA is the required method for analysing the system.

At this level of authority (DP control system), we assume that the desired position and heading is set at a safe location. Selecting a safe location is the responsibility of a higher-level authority (e.g. DP operator or an autonomous navigation system) and is therefore excluded for

now. However, we will return to it later in the example. If the commanded thrust is too low, or too high, or in the wrong direction, the ship will move away from the required location. So, the objective of the STPA analysis is to find scenarios that may cause the commanded thrust to be too low, too high, or in the wrong direction compared to what is necessary to keep the ship stationary. To assist in the identification of scenarios, we must investigate the different agents in the control structure.

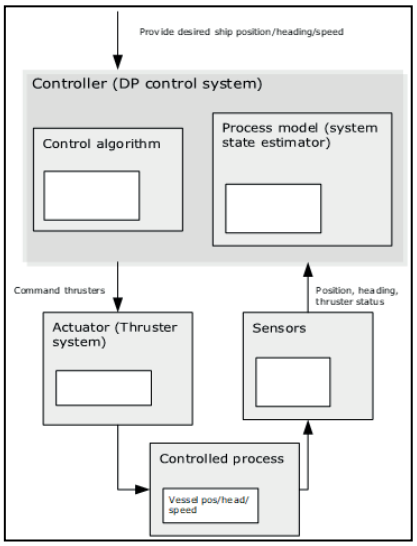

Figure 1: Control structure of a dynamic positioning control system

As stated above, the control action is a function executed by an agent. A function is always a sub-function of a principal-function, or as they say: "There's always a bigger fish". At the same time, it can be divided into several sub-functions. We know that the agent responsible for the control action is the control algorithm. What is the purpose of this agent? The purpose is to **minimize the deviation between the position/heading setpoint and the actual position/heading of the ship** by **providing force and azimuth setpoints to the thruster system**. Thus, the functions the particular agent contains are “Minimize pos/head deviation”and “Provide force/azimuth setpoint”. We may claim that the principal function is “Minimize pos/head deviation”, because if this function were not necessary, then the other function would not be needed. The two functions identified so far are placed in a hierarchy of principal-functions and sub-function:

Minimize pos/head deviation
  Provide force/azimuth setpoint

The purpose of STPA is to identify scenarios that may cause "Provide force/azimuth setpoint" to fail in different modes that may put the system into a hazardous state or condition. The guidewords help to identify these modes at a high abstraction level and form a complete set of fault modes (Leveson & Thomas, 2018). By keep asking How, with respect to the control action, we identify the sub-functions needed to perform it. And through these sub-functions, we can identify how an unsafe control action can be executed.

There are two functions related the How-question: “Compute pos/head deviation” and “Allocate force to each thruster”. These are supporting functions which must be executed to perform the function: "Provide force/azimuth setpoint”. These functions are marked with ^ in front of the name in the FAST tree as they all are necessary support functions. They are numbered to indicate that they belong to the same function. A FAST tree will look like this:

Minimize pos/head deviation
Provide force/azimuth sepoint (CA)

Provide force/azimuth setpoint
^1Compute current pos/head deviation
^2Allocate force to each thruster

Different teams will certainly come up with different names for the functions. The point is whether or not the list is useful in identifying scenarios for unsafe control actions. With a "few" words (name of each function), and a limited set of semantics (format of the FAST tree, the why/how logic, and the supporting functions), we can already start to get an idea of how unsafe control actions may occur. Any fault in the sub-functions or supporting functions of the control action can potentially cause an unsafe control action.

Note that FAST lacks strong semantics for a temporal relationship between functions, and agency is not part of FAST; however, STPA includes both. STPA and FAST are therefore complimentary. The suitability with respect to hazards analysis of the combined STPA and FAST approaches can be thought of as an emergent property of its constituents, or in other words: the combined use of the methods exceeds the sum of its parts.

Of course, someone could dig even deeper into each sub-function or supporting function. For instance, regarding the supporting function: "Compute current pos/head deviation":

Q: "How is "Compute pos/head deviation" performed"?

A: "Compute distance between desired and current pos/head "
Furthermore, what other function(s) must be performed when/if the newly identified function is to be accomplished? The answer may consist of two functions: "Retrieve desired pos/head/speed" and "Retrieve estimated pos/head/speed". The revised FAST tree may look like this:

Minimize pos/head/speed deviation
Provide force vector
Provide thruster force/azimuth setpoint (CA)
^1Compute pos/head deviation
^^1 Retrieve desired pos/head/speed
^^2 Retrieve estimated pos/head/speed
^2Allocate force to each thruster

Any of the two newly identified functions may cause "Compute current pos/head deviation" to fail, resulting in an unsafe control action. Notice the two carats in front of the newly identified functions, they indicate that the functions are support functions to another support function.

2.1. FAST Tree and STPA Control Structure

In Figure 2, the FAST tree is included into the STPA control structure. Whether the FAST tree should be included inside the control structure, or as an add-on on the side, will depend on the level of refinement in the FAST tree. Large trees are difficult to visualize inside without introducing clutter and therefore decreasing the usefulness of the control structure in achieving its purpose. However, keeping the FAST tree small may maintain a "clean" control structure and at the same time provide useful insight into the inner workings of the different agents, helping to spark the imagination and creativity of the analysis team to identify scenarios leading to unsafe control actions.

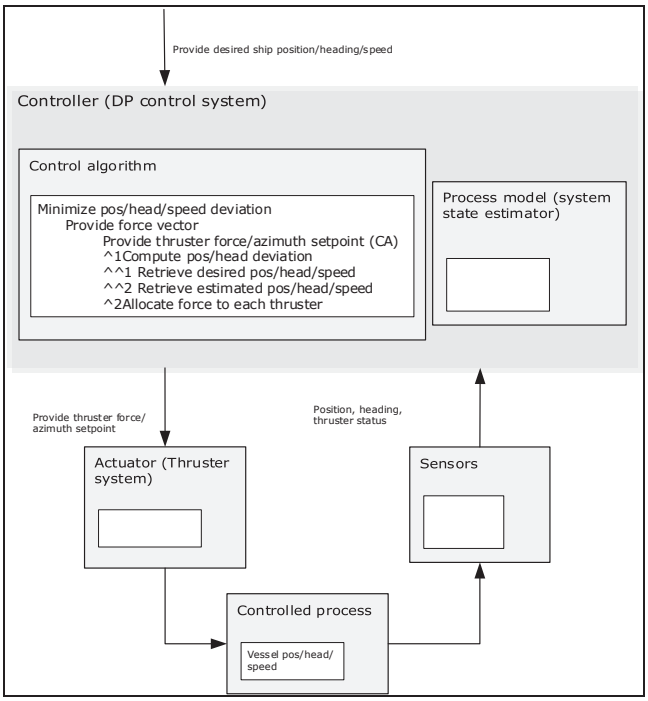

Figure 2: STPA control structure augmented with the FAST tree in the control algorithm

2.2. FAST Trees Inside Other Control Structure Agents

All agents in the control structure serves a purpose, and that purpose is achieved through a set of functions. Again, STPA is well suited as an agency-analysis method.

The actuator in the running example of the DP control system, is a thruster system consisting of a number of units with their own local control system. STPA instructs us to investigate all agents in the control structure for scenarios of unsafe control actions.

2.2.1. Thruster Control System

Assuming pitch-controlled azimuth thrusters, thrust is generated through altering the propeller blade angle. The blade angle, or just pitch, is altered by the use of hydraulic pumps. Thrust direction is altered through rotating the thruster, i.e. altering the azimuth. There are two control actions coming from the DP control system; the “Provide force setpoint”, and “Provide azimuth setpoint”. Therefore, there must exist two agents responsible for achieving the objectives expressed through the control actions (force and azimuth). Below we will only focus on the control action related to force. Without going into the same level of detail as with the control algorithm, the FAST tree for the thruster system may look as follows:

Minimize force deviation
Provide thruster pitch setpoint (CA)
^1Map force to pitch
^2 Compute delta pitch
^^1 Retrieve desired pitch
^^2 Retrieve current pitch

The FAST tree within the actuator agent in the STPA control structure is shown in Figure 4. Notice that the FAST tree of the actuator contains the function, "Provide thruster pitch/azimuth setpoint", that affect the system state that is important for safety, or in other words, can put the system into a hazardous state. It may be tempting to just continue with FAST and elaborate around

possible causes of why this function may fail. However, recall the deficiency of FAST; semantics for temporal information and agency (i.e. the sequence of action, and who is responsible for what action), and how these agents depend upon each other, which is where STPA excels.

By identifying control actions among the functions in the FAST tree, we can refine the higher-level control structure into details about the actuator. The control action is identified as the function directly affecting the system state that is important for safety. In this case, thruster pitch affects the generated force acting on the ship. The FAST tree describing the actuator can be thought of as describing the functional dependencies of the control algorithm in the lower level control structure (Figure 3). The responsibility of the particular controller is to minimize the deviation between desired and current force. Another controller will be responsible to minimizing delta azimuth. Both are important for avoiding the defined hazard, but on different level of authority compared to the DP controller.

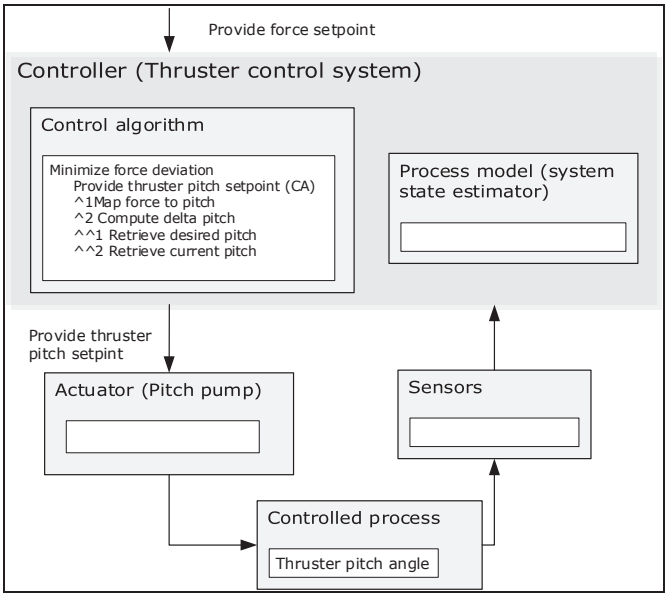

Figure 3: STPA control structure of the actuator augmented with a FAST tree in the control algorithm

2.2.2. Sensor System

The sensor system can also be visualised through FAST. In a DP control system, the sensors consist of system state sensors, environmental sensors, and actuator sensors:

Measure ship pos/head
  Collect position
    Collect GPS data
  Measure ship heading
    Collect Gyro compass data
Measure ship pitch/roll
  Collect Vertical Reference Unit data
Measure environmental variables
  Collect wind data
Measure thruster variables
  Collect pitch data
  Collect azimuth data

The FAST tree within the sensor agent in the STPA control structure is shown in Figure 4. The team of analysts can get an overview of the feedback path in the control structure and identify possible scenarios for unsafe control actions.

2.2.3. Controller Process Model

The last agent to check for scenarios that may cause unsafe control actions is the controller's process model:

Estimate pos/head
Update pos/head model
^Retrieve ship states
Estimate ship speed vector
Compute force balance to ship speed
^1Estimate thruster forces
^2Estimate environmental forces

The FAST tree within the process model agent in the STPA control structure is shown in Figure 4. Note that the DP control system control algorithm contains the supporting function: "Retrieve estimated pos/head/speed". These system states are calculated by the process model agent, and by investigating the sub-functions of the process model, possible scenarios about why these functions may fail and cause unsafe control actions may be found.

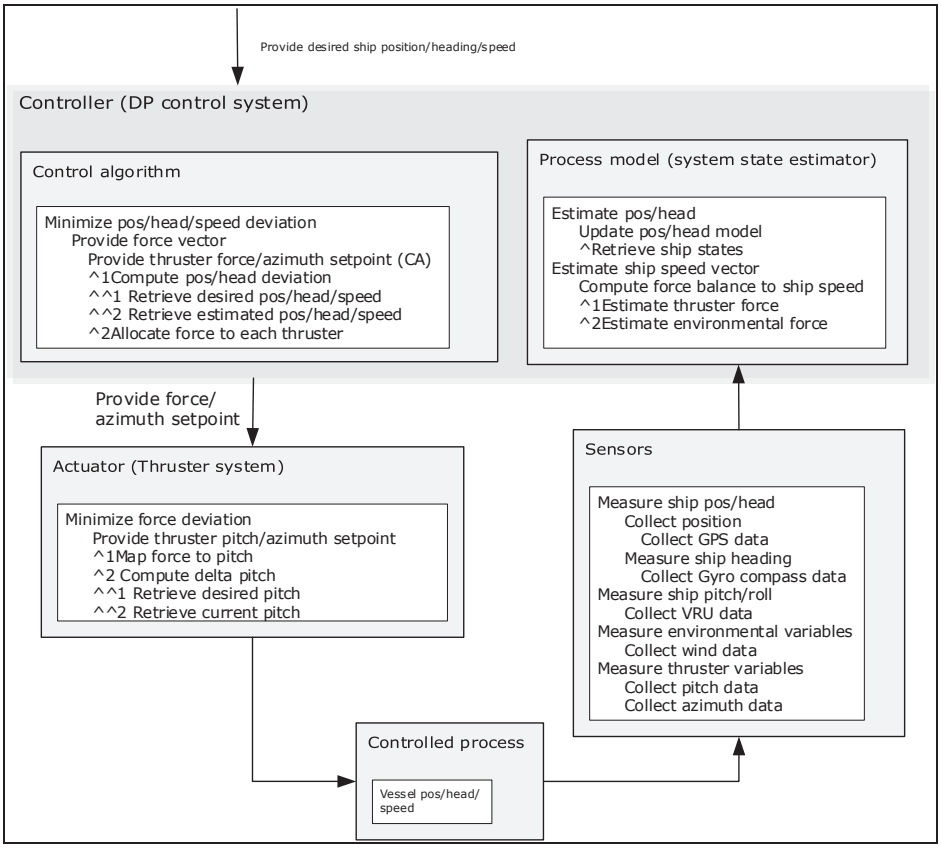

Figure 4: STPA control structure augmented by FAST trees in all agents

3. IDENTIFYING AND ANALYSING HIGHER-LEVEL AUTHORITIES

Higher level authorities can be identified and analysed in the same way as lower level authorities. The input to the DP control system "Provide desired ship position/heading/speed" is in

fact a control action given by a controller at a higher level of authority. Providing and keeping the ship position is just a means to an end, where the end is the goal or the *mission* of the operation. In this higher-level system, the DP control system is one part of the actuator system, other actuators are all other means necessary to achieve the mission. However, we must limit our analysis to hazards that may be caused by the system under investigation, which in this example is the DP control system.

We perform the steps of FAST to identify the functions in the control algorithm, starting with the control action going into the DP control system: "Provide desired ship position/heading/speed". Another well-known function in any controller is "Minimize operational goal deviation" and "Compute current operational deviation":

Minimize operational deviation
Provide desired ship position/heading/speed
^Compute current operational deviation
^^1Retrieve desired operational goal
^^2Retrieve current operation status

Figure 5 shows the STPA control structure of the higher-level authority including FAST trees within all control structure agents.

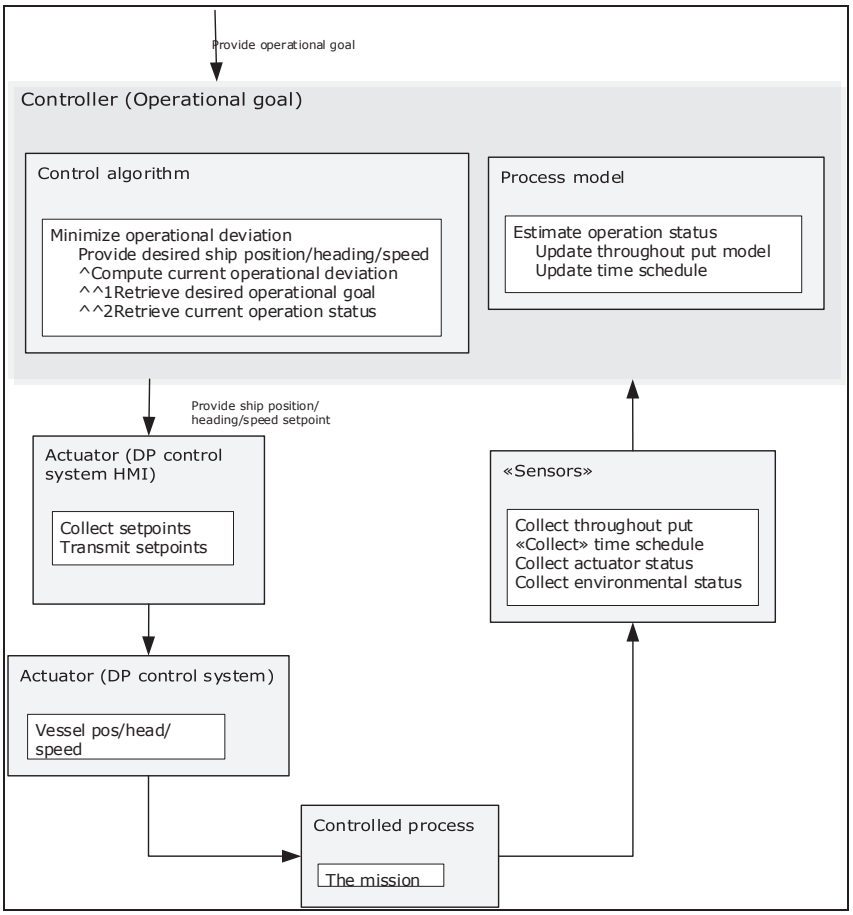

Figure 5: STPA control structure of higher-level authority augmented with FAST trees

4. SMALL FAST TREES WITHIN THE STPA CONTROL STRUCTURE IN A STUDY SESSION

STPA is a function-centric analysis method. While other function-centric methods, like Functional Hazard Analysis (FHA) make function names and their relationships explicit, STPA

relies on the functional control structure for representing functions, and their relations. Diagrams and drawings are imperative when a team is gathered to study a system. These "studies" should be conducted in all hazard identification and analysis methods, including STPA.

Drawing boxes with names, and linking them with lines and arrows, in the context of FHA, is an intuitive way of depicting relationships between functions. However, the diagrams may quickly become cluttered, and the meaning of the lines and arrows may be ambiguous. To improve the situation, the diagram may be spread over several pages and the lines may be tagged. Decreasing the number of boxes on a single page will also decrease the information density, which result in decreasing the usefulness of the diagram when used in a study-session. Tagging lines and other objects increases the information density, while at the same time adds to the clutter, thus decreasing the readability. A good diagram is packed with relevant information, conveying important system aspects with respect to the objective of the study, with minimum clutter and maximum readability.

The most obvious way of maximizing information density, while keeping clutter to a minimum, is to use dedicated and well-defined symbols. The more meaning put into a symbol, the more information will that symbol carry, leading to increased information density, while keeping clutter to a minimum. The challenge with this approach is that these diagrams may only be readable by method-specific specialists and not Subject-Matter Experts (SMEs). In a study, this approach will make the diagram less useful because the SMEs become alienated, preventing the analysts' creativity to be ignited and reducing their involvement.

The functional control structure used in STPA depicts dependencies and relations essential to the understanding of what may lead to a hazard in a fairly compact way, while at the same time, it is readable and understandable to SMEs. This makes the functional control structure well suited for a study-session with a team of STPA specialists and SMEs. Small FAST trees have been shown to reveal more information about functional relationships in order to ignite creativity and increase the understanding of implicit functional (hierarchical) dependencies. Together with the STPA "guidewords", this will increase the probability of identifying more scenarios for inadequate control that could lead to hazards.

5. SUMMARY

All agents (control algorithm, actuator, sensor system, process model) in the control structure consist of a set of functions which may be visualized and analysed using FAST. The control action is the "output" from the control algorithm. By using FAST we can analyse the sub-functions of the control action and identify scenarios that may cause unsafe control actions. The actuator is often a control system on its own, and FAST can help identifying new control structures and control actions though explicitly showing functional dependencies within the actuator system.

Even the "sensor system" in the feedback path, may in some instances be analysed as a control structure including functions visualized as FAST trees, if there is an agent that is responsible to trigger the sensor system to start transmitting feedback. Other times, the feedback path consists of only the sensor agent that automatically transmits feedback, which however, still may be analysed using FAST to identify why the feedback may lead to unsafe control actions. Last, the responsibility of the process model agent is to create an adequate and useful model of the process under consideration. The functions needed can be analysed using FAST to understand how flaws in the process model agent can cause unsafe control actions, and why they can occur.

6. DISCUSSION AND CONCLUSION

FAST trees make hierarchical functional dependencies explicit. When identifying scenarios that may lead to UCAs, analysts tacitly create a mental model of these dependencies. However, the strength of STPA is in agent analysis, by identifying system agents responsible for enforcing safety constraints, while the hierarchical functional dependencies are more implicit.

Hierarchical functional dependencies and agent analysis may be thought of as two important dimensions in function analysis. Function dependencies depicted in FAST creates *holarchy*-like functional abstraction levels ("Holarchy" was first coined by Arthur Koestler in his 1967 book "The ghost in the Machine", which in contrast to a hierarchy does not have a top or a bottom, and each level constitutes a set of *holons*). On the other hand, agent analysis in STPA creates holarchy-like levels of *authority*. These two dimensions are linked in such a way that the function on a certain level in the abstraction-holarchy can identify new control action levels that further will identify a new controller, and control structures in the authority-holarchy. When a new level in the authority-holarchy is identified, the functions within each agent in this control structure can again be analysed using FAST.

There are two types of functions visualised in FAST trees: inter-agent functions (agent-to-agent functions), or intra-agent function. An inter-agent function requires a correct command/response pair in order to be successfully accomplished, while an intra-agent function only depends on the agent responsible for conducting it. A control action is one special case of an inter-agent function, requiring that the actuator agent responds adequate to be successfully accomplished, and of course it will directly affect the system state important for safety. In addition to a control action, an inter-agent function may also be that the state estimator or "process model" agent in a control structure requests a measurement from the sensor agent. The principal-function dependent upon the inter-agent function may fail (perhaps causing an unsafe control action) if the request comes too late, or too early, but also if the sensor agent transmits the wrong value, or transmits too late and so on.

Small FAST trees within the control structure increase the information density hopefully without creating too much clutter. The semantics in FAST are relatively easy and quick to learn for SMEs and others. The level of detail in the FAST trees should be adjusted to fit the purpose of the analysis. If they become too large to fit inside the control structure, they can be printed and labelled to identify the agent in separate drawings.

The human brain is predisposed to think in hierarchies to deal with complexity and problem-solving, therefore, FAST trees resonate well with the human brain. Also, FAST trees can guide refinement of the control structure by identifying functions as new lower-level or higher-level control actions that need further investigation in new control structures. Moreover, FAST can be used both when the system details are known to the analyst and when these details are not known, either because it is early in the design, or the analyst is analysing the system as a black-box (third party analyst).

The original purpose of FAST was to spark the creativity to find alternative solution to a problem, or alternative ways of achieving a function. This is valuable early in the concept and design phase of any system development, including using STPA in system safety engineering. The paper has investigated the potential for using FAST within the SPTA technique. We believe that FAST can augment STPA by systematically analyse the hierarchical functional dependencies within each agent in the STPA control structure. FAST may guide the refinement of control structures as well as assist to identify new lower level, or higher-level control actions. Moreover, FAST, together with STPA, can assist in analysing and selecting alternative designs early in the development of new systems.

ACKNOWLEDGEMENT

The authors would like to thank **Dag McGeorge** who is a colleague in DNV GL, for the introduction to FAST. Without this introduction, FAST would probably remain unknown to the authors.
This work is funded by the project “**Online Risk management and risk Control for Autonomous Ships**” (ORCAS). The Norwegian Research Council, DNV GL and Rolls Royce Marine are acknowledged as sponsors of project number 280655.

REFERENCES

Bunge, M. (2003). *Emergence and Convergence.* University of Toronto Press.
Bytheway, C. W. (2007). *FAST, Creativity & Innovation.* Florida: J. Ross Publishing.
Ericson II, C. A. (2013). *Software Safety Primer.*
International Maritime Organization [IMO]. (2017). GUIDELINES FOR VESSELS AND UNITS WITH DYNAMIC POSITIONING (DP) SYSTEMS. *MSC.1/Circ.1580*.
Leveson, N. (1995). *Safeware, System Safety and Computers.* Boston: Addison-Wesley.
Leveson, N. (2011). Engineering a Safer World, Systems Thinking Applied to Safety. MIT Press.
Leveson, N., & Thomas, J. P. (2018). *STPA Handbook.*
Rasmussen, J., Pejtersen, A. M., & Goodstein, L. P. (1994). *Cognitive System Engineering.* Wiley-Interscience.
Society of American Value Engineering (SAFE). (n.d.). Retrieved from https://www.value-eng.org/

APPENDIX: A QUICK FAST OVERVIEW

Fast is a method for sparking the imagination and creativity to identify functions. Moreover, it facilitates communication of the processes that make the system behave the way it does, or the Mechanisms in the CESM-model, (Bunge, 2003). FAST can be done by one person, but the best result is achieved when it is performed in a team.

A central feature in FAST is the *function*, and its name. In FAST, there are certain rules or conventions for naming functions. These rules serve different purposes, such as being specific about the intension of the function, and sparking the imagination and creativity.

There are two ways of visually communicating functions and functional dependencies within FAST: FAST diagrams, and FAST trees. Both will be described below, but briefly, FAST diagrams are good for brainstorming about smaller systems, or parts of systems because the method is more graphical in the context of a brainstorming session. The challenge is that diagrams take up a lot of space, and therefore become large if the analysis goes into much detail. FAST trees, on the other hand, take less space, and therefore are more convenient for visualising larger systems, or larger parts of systems.

The most central concept in FAST is the Why-How logic. In order to establish cause-effect relations between the branches in the FAST tree, one asks one either (Why or How questions). This relation can also be thought of as a means-end relation (Rasmussen, Pejtersen, & Goodstein, 1994).

Naming functions

"*The function name should immediately say something about what is to be accomplished without disclosing the method of accomplishment*" (Bytheway, 2007). The name should be short and concise. The main rule in FAST is that functions names starts with an active verb, and end with the noun. An active verb defines an action upon something, and the noun is this "something". A *snow shuffle* is an object; “Shuffle snow” is a function.

Sometimes a function may need to be more specific, and a modifier can be added to the function name. There might be a function that monitors the commands given to the system. These commands may have different origins, such as a remote operator location, from another system based upon some condition, or from a local operator. The function name may be "Monitor command", however, to be more specific about the context, the name can be rephrased to "Monitor remote operator command". Although the name contradicts the basic naming rule by containing more than an active verb and a noun, it is still better than e.g. "Command monitoring of commands given by remote operator location", or even "Monitoring of commands performed by remote operator".

Functions should be restricted to only one action, describing multiple actions should be done by adding more functions and linking them together in a FAST diagram or FAST tree. In

Table A.1 there are a few examples from a Power Management System (PMS), where the original function name is written in the left-hand column, and an alternative function name following the FAST naming convention is in the right-hand column. Note that some function names on the right omit some details in the original name. The names are a trade-off between length, specificity, and detail level, governed by the cognitive ease required to grasp the meaning of the function, and the knowledge of the people involved in the analysis. Longer names tend to require more cognitive effort to grasp the meaning which tends to decrease the ability to be creative in the analysis. However, shorter names require more system knowledge by the participants. It is expected that the names in the "FAST name"-column ignite more cognitive reactions compared to the original names.

Table A.1 Modified function names using FAST naming rules in context of the Power Management System

Original name	**FAST name**
Monitoring and command of PMS functions in HMI	1) Monitor operator commands 2) Execute operator commands
Active power load sharing between generator sets	Share active load
Active power unbalance detection	Detect active power unbalance
Reactive power load sharing	Share reactive power
Reactive power unbalance detection and handling	1) Detect reactive power unbalance 2) Control reactive power unbalance
Bus frequency control	Control bus frequency
Under and over-frequency detection and handling	1) Detect over-frequency 2) Control over-frequency
Power reservation functions	Reserve power
Start interlock and load shedding of heavy consumers	1) Shed load from heavy consumers 2) Interlock heavy consumers

Why-How logic

There is a reason for functions to be executed. To understand the reason, we ask "**Why?**" with respect to the function in question. "*Why do I want to perform this function?*" (Bytheway, 2007), or "Why do I want to perform" "Shed load from heavy consumers"? The answer is: *To prevent overloading the generators by limiting the power consumption to avoid a total blackout of the switchboard*. How many function does the previous answer contain? Let us now analyse the answer and transform it to functions:

The answer is to **prevent overloading of the generators by limiting the power consumption to avoid a total blackout of the switchboard**. The answer contains at least three functions. "Prevent generator overload", "Limit power consumption", and "Avoid blackout". It seems like all these functions are the reason for the function: "Shed load from heavy consumers", but at the same time it seems that there is some hierarchical structure built into the answer to the Why-question, represented by the three functions.

Let us have a look at the function "Limit power consumption". If we ask "Why", the answer is straight forward: "Prevent generator overload". If the "Why"-question is asked concerning "Prevent generator overload", the answer is also straight forward: "Avoid blackout". Now we can see the hierarchical structure of these questions and answers, starting at the bottom: "Shed load from heavy consumers" **Why**? -> "Limit power consumption" **Why**? -> "Prevent generator overload" **Why**? -> "Avoid blackout".

Turning the above Why question/answer "upside down", gives the How question/answer. "*How is this function actually performed?*" (Bytheway, 2007). Starting at the top function*:* "Avoid blackout" **How**? -> "Prevent generator overload" **How**? -> "Limit power consumption**" How**? -> "Shed load from heavy consumers".

Sometimes the answers are not obvious. Asking a verification question will clarify whether or not the answer to the How-question is valid: "Does this How-function help its Why-function?" If the answer is yes, then the logic holds (Bytheway, 2007).

This is the principle of the Why-How logic. The Why-question need not stop at "Avoid blackout"; there is a reason why we want to avoid blackout - we may keep asking the Why-question, and we will end up with the primary goal of the entire system or mission.

There are other ways than "Limit power consumption" to "Prevent generator overload", e.g. by connecting additional generators to the switchboard. Then the load of each generator is decreased, and "Prevent generator overload" is achieved. This has identified one new function: "Connect generator". In a brain storming session, another function may come into mind: "Start generator". One may think that "Start generator" is in a Why-How relationship to "Connect generator", but that is not the case.

By asking the "Why" question with respect to "Start generator", the answer could be to be able to "Connect generator" because you cannot connect the generator to the switchboard without first starting it. Asking the other way around: How is "Connect generator" actually performed?", the answer is **not** "Start generator". Instead, asking again the Why-question with respect to "Connect generator", and answering it by: "Add power source", then "Connect generator" and "Add power source" is in a why-how relationship. "Start generator" is instead a *supporting function* to the "Connect generator" function.

To identify a supporting function, we ask the question: "**When/if this function is performed, what other function must be performed?**" - at the same time or prior to "this function": "When/if connect generator is performed, what other function must be performed? The answer is (among other things) "Start generator". The function "Start generator" will now be the starting point of a new hierarchy on its own by continuing asking the “How”-question.

FAST diagrams and FAST trees

The above discussion indicates that there is a hierarchy of functions resulting from the Why-How questions. This hierarchy is visualized through **FAST diagrams** or **FAST trees**. Which one to select, depends on the objective and context of the analysis. FAST diagram is well suited for brain storming of smaller parts of the system, while the FAST tree is more compact and therefore is suited for larger parts of the system, and for documentation. Figure A.1 shows the FAST diagram of the above discussed PMS functions. A FAST tree will look something like Figure A.2. Notice the carat preceding the function "Start generator" function, this indicates a *supporting function*.

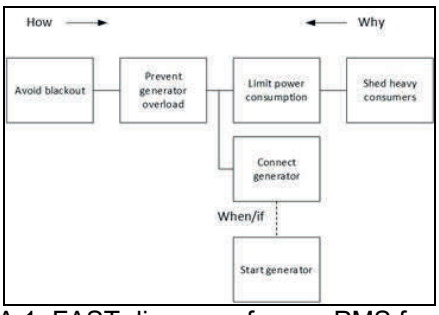

Figure A.1: FAST diagram of some PMS functions

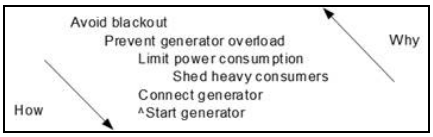

Figure A.2: FAST tree of some PMA functions