Journal of Defense Management

OPEN ACCESS Freely available online

Research Article

Sparse Requirements Systems Engineering and Implications for Assured Autonomy

Gregory Gosian, Kelly Cohen*

Department of Aerospace Engineering & Engineering Mechanics, University of Cincinnati, Cincinnati, Ohio 45221, United States

ABSTRACT

Systems engineering in program execution identify comprehensive requirements definition as a prerequisite for successful programmatic execution. Conversely, not fully defining system requirements is frequently branded as being the root cause for programmatic failures. The failures manifest themselves as not meeting performance thresholds, exceeding budget, or delivering late. With respect to executing engineering programs in the 21st century, and especially for the Department of Defense (DoD) and other Government customers, there are drivers that require systems engineers to commence system specification prior to fully defining (or having defined for them) all of the system or mission requirements. As such, there is a need for a new method of systems engineering that allows for progress in environments of uncertainty – a Sparse Requirements Systems Engineering (SRSE) paradigm. Further, a consequence of architecting Learning Enabled – Cyber Physical Systems (LE-CPS) in such a reduced requirements environment is the need to assure safe, intended operation. SRSE emulates an incomplete training data set of inputs, further increasing the potential for unintended response and operations of such LE-CPS.

Keywords: Incomplete requirements; Quick-Reaction capability; QRC; Best-Effort; Systems engineering

INTRODUCTION

Part one – Sparse requirements systems engineering

Systems engineering is concerned with distilling customer requirements into programmatic and other self-imposed requirements. Those requirements further drive component or system of systems (SOS) specifications. Throughout the process, the Systems Engineer (SE) has to define the methods to integrate, verify and validate that the components and SOS that will comprise the overall system, and meet the performance, cost and schedule requirements imposed by the end user. Illustrative cases in Systems Engineering typically focus on programmatic successes with respect to performing the complete requirements definition prior to decomposition to system specifications. Successful engineering projects are claimed to have characteristically taken the required time to execute all comprehensive systems engineering prior to project commencement and proceeding to the design, analysis and procurement phases of the program. It is important that the SE exercises due diligence with respect to these responsibilities lest inadequacies remain undetected until late in the build cycle, where disruptions to resource loading, budget and schedule are greater than if such shortcomings had been detected earlier.

Conversely, breakdowns in engineering program execution and delivery are often tied to a failure to adequately define the system at the onset of program execution. Such failures manifest themselves as not meeting performance thresholds, exceeding budget, or delivering after due dates. They are blamed on a variety of traditional reasons, such as "the customer changed the requirements" (failure to deliver performance), "the program schedule was unrealistically aggressive" (failure to deliver on time), or "the Business Development Team bid to win instead of bid to make money" (failure to deliver to the budget) [1].

With respect to executing engineering programs in the 21st century, there are drivers that require SEs to embrace all of the horrors attributed to inadequate systems engineering; namely, how one defines and delivers when not all of the requirements or specifications are fully determined before design and procurement must commence. The drivers for the commercial world are often schedule and budget: e.g., competitive time to market, limited financial resources for developmental efforts, or execution prior to constituent component or subsystem obsolescence. Similarly, the Government end user might be responding to an immediate need identified by the war fighter in the field. U.S. Government acquisition reform was undertaken in 2003 because previous procurement guidelines dictated that it literally took decades to field new weapons systems [2].

Correspondence to: Kelly Cohen, Department of Aerospace Engineering & Engineering Mechanics, 735 Rhodes Hall, Cincinnati, Ohio 45221, United States, E-mail: cohenky@umail.uc.edu

Received: November 25, 2021; **Accepted:** December 10, 2021; **Published:** December 17, 2021

Citation: Gosian G, Cohen K (2021) Sparse Requirements Systems Engineering and Implications for Assured Autonomy. J Defense Manag. 11:220.

Principally for the government customer, the need to execute with speed is not driven by profit or shareholder value, however even more dire consequences can result from failure to move concisely. The need to acquire rapidly are due to a number of considerations, among them are: identification of a capability required for the war fighter already in theatre, the discovery of an adversary capability that needs to be immediately countered, or (in the case of space-based capabilities) having to meet an immovable deadline (such as a launch date dictated to accommodate celestial mechanics). So important is the objective of swift acquisition (acquisition in the sense of having received from the supplier and ready to deploy the desired capability), that these U.S Government organizations have stood up their respective offices to insure such:

- U.S. Army - Rapid Equipping Force [3].
- U.S. Air Force – Space Rapid Capabilities Office [4].
- U.S. Navy – Maritime Accelerated Acquisition [5].
- U.S. Marines – Rapid Capabilities Office [6].
- NASA – Rapid Spacecraft Development Office [7].

The yearly war games exercise covering the defense of Taiwan from a Chinese invasion, was recently (2021) won by the U.S. but at an enormous cost to life and equipment, yet is a better outcome than similar exercise in 2017 and 2018. The 2022 exercise is expected to further improve by determining what mix of aircraft, unmanned systems, data transport networks and other systems will be needed to defeat China in a potential conflict. Lt. General Clint Hinote, the U.S. Air Force's Deputy Chief of Staff for Strategy, Integration and Requirements commented on our adversary's agility: "China is iterating so rapidly, and I think that forces us to change. If we can change, we can win'" But General Hinote is not referring to winning in the traditional military sense: 'We're trying to help people see the future, what it might look like, the types of choices it would take to win a war…the evidence based possibility that if we were able to change, we probably wouldn't have to fight, and that's a reason [in and of itself] to change" [8].

The common thread is that as our adversaries (either commercial or military) become more agile, our own system delivery must be executed more rapidly, especially if it is accompanied by the challenges of a less-than-fully-defined system. Further, engineers will be required to minimize the issues that missing requirements generate. As such, there is a need for a new method of systems engineering when uncertainty is an integral element of the effort. The International Council on Systems Engineering (INCOSE) Systems Engineering Book of Knowledge (SEBOK) mentions the concept of incomplete requirements only twice in all of its 1063 pages, and then only as something to be avoided [9].

One can craft a new term: Sparse Requirements Systems Engineering (SRSE). While hardly known by this label, most systems engineers have been forced to work under such time, resource and a-priori knowledge constrained conditions. Indeed, having <u>all</u> of the requirements necessary before commencing on system architecture is almost a luxury in many fields. Historically, programs (especially large and complex ones) that have eagerly commenced execution without complete and correct requirements have tended to experience significant increases in costs and schedule. Trying to operate a program utilizing legacy systems engineering tools and carrying incomplete or deferred requirements, is incongruous with purposely executing under those conditions. Some of these complications have been so great as to result in program cancellations [10].

Therefore, the following differentiation must be recognized before proceeding: SRSE is not synonymous with inadequate systems engineering. SRSE is a conscious decision to proceed toward system definition, design and procurement before all requirements are fully defined and with an acceptance of any additional associated risks. This implies also carrying forward the risk mitigations as desired (including their additional costs), an awareness that the system performance and the final design iteration will become fully defined at some point later in the integrated product lifecycle, and an acknowledgment that the final deliverable may not meet 100% of the system design objectives (but will likely be "good enough"). By contrast, inadequate systems engineering is comprised of:

- Accepting a fully defined set of requirements, but limiting the application of analysis and design trade tools to examine all alternatives before settling on a particular design implementation, either through conscious decision or poor execution.
- Not performing due diligence and commencing on design while requiring a complete set of requirements, but failing to obtain them all

Being mindful of the distinction, it can be ascertained if an SRSE approach is warranted for a particular objective.

METHODOLOGY

Motivation and execution mode selection

There are many reasons for wanting to move a program along as expeditiously as possible:

- Time to market and competition considerations; providing a product to satisfy market demand
- Programmatic milestones correlated with award funding availability
- Addressing the needs of warfighters already in theater
- Immediate deployment of a system to counter a newly discovered adversary capability
- Executing under a quick reaction capability paradigm
- Rapid capability reconstitution or technology insertion

Not being able to fully define a system may be due to issues beyond control. For example, in the case of developing a countermeasure to an adversary's newly discovered offensive system, the initially observed capabilities are unlikely to be complete or be fully ascertained. The full knowledge of the opponent will evolve as intelligence and subsequent experience in the field drives revisions to initial capability estimates. This in turn will drive amendments and additions to the initial system requirements.

Requirements in an SRSE environment can be categorized into four types (Table 1).

No matter what the cause, waiting for a system to be fully defined may actually be more expensive than starting while system definition is still on-going. To decide if that proposition is true, the

Table 1: SRSE requirement types.

Requirement Type	Definition	Example
Fully defined	Subsystems characteristic that have to be met; no compromise is allowed	Mission duration, flight altitude, type of host platform (specific aircraft or satellite bus), sensor metrics (waveband, NIIRS, SNR, P_d, P_{fa}) Not-to-exceed limitations (power, volume, heat dissipation, weight) Pre-defined interfaces
Undefined	Parameters that have to be met but the final values have yet to be determined; intermediate to-be-revised (TBR) values are allowed	Sensor architecture (a camera's format: X pixels by Y pixels) Final values for sensor figures of merit, payload power, payload weight, sensor frame rate
Negotiable	Values that require specification, in order to simplify the system architecture; penalties for variance are able to be accommodated. A defined range of values is allowed	Example 1) The figures of merit of a component system can be specified, but the risks of a developmental program (cost overruns, late delivery and sub-optimal performance) to deliver a custom system may be unacceptable. Alternatively, an existing system that is close to specification may be selected, and a reduction in desired performance may be accepted in trade in order to mitigate risks to schedule and cost. Example 2) The share of platform power and volume available for a particular subsystem, at the expense of other subsystems.
Dependent	A subsystem characteristic that relies on another subsystem's final definition	Available SWaP based on final aircraft or spacecraft bus selection; sensor type(s) based on final mission requirements definition

cost of fully defining the requirements, *vs.* embarking on a SRSE effort has to be evaluated. This can best be done by evaluating the following:

Cost = ∑ fully defined requirements costs – ∑ sparsely defined requirements costs (Equation 1)

If the value defined in Equation 1 is a positive number, then the incentive to proceed under the conditions of a sparsely defined set of requirements is present.

The equation seems innocent enough but assigning actual values so that each summation is comprehensive can be challenging if not impossible. The constituent components that go into each summation range from actuals costs, to costs derived from "similar to" prior experience, from estimates to guesses. Correspondingly the fidelity of such individual contributors to the equation decreases as one goes from "known" to "guesses." The span of what has to be studied is considerable. When analysing the cost of executing a program with fully defined requirements *vs.* a SRSE type program, some costs will be very similar if not completely common to both, such as final assembly, integration and testing. It will be the more major differences between the two that go into the Cost analysis that we are most concerned with in Equation 1.

Costs associated for program execution with fully defined requirements

While carrying out extensive requirements analysis and comprehensive risk burn-down, there are costs associated in addition to the direct Systems Engineering costs. These are the often-overlooked Program Office: program management, project engineering, information technology, configuration management, administrative assistance, contracts, purchasing, and mission assurance (aka "Quality Control"). Further, there are the overhead costs associated with this standing army of personnel; an engineering salary comprises roughly 40% of the per-hour cost to a company: vacation & benefits, facilities, operations…the additional costs are interminable. Travel, material, software, computers, and office products comprise other direct costs (ODC). Loss of revenue due to delay in market entry, allowing competitors to seize the consumer high ground (there is a saying among DOD system providers with respect to market share: "the first hog to the trough wins") has to be factored in. Reduced product life as time spent fully defining a system eats into the amount of time the deployed product is relevant to its purpose; in other words maximizing time to obsolescence.

The funding profile for this type of execution is expected to be heavy on labor during the Systems Engineering and early design phases, followed by material acquisition. Costs are principally labor-only during assembly, integration, test (AIT) and delivery (unless new test equipment or infrastructure is required to support such activities).

Costs associated with sparsely defined requirements

The costs capturing associated with SRSE efforts require more diligence than the traditional fully-defined requirements type of execution, because this type of program execution will entail early material purchases in addition to the systems and design engineering that commence early in the program, as well as the need to support parallel development efforts. The funding profile for this type of execution is expected to very front-heavy due to material acquisition and commitment to long-lead items, followed by tapering off to a spend rate driven predominantly by labor for most of the remainder of the program (again, AIT activities).

It is important to continuously re-examine the program as a whole and not get lured into looking at small portions, however problematic certain elements may seem to become. This is to prevent being encumbered by a large number of unconsidered unknowns (as opposed to a large number of TBDs and TBRs) which will drive additional unplanned expenditures for materials and labor, well into program execution, causing the program to come in over-budget.

Parallel path developments as part of risk mitigation and to ensure design and program progression are up for consideration when figuring costs

- It is advantageous to utilize components that are common to co-supporting multiple design paths, *vs.* components that are unique to each individual design path. Economy of scale and fewer sources to manage are primary benefits; the need to maintain fewer on-hand spares as a percentage is a cost saver as well.

- It is advantageous to develop software/firmware (SW/FW) that is common across multiple design solutions, but that may not be possible. Modular open standards architecture (MOSA) for software/firmware is anecdotally believed to be advantageous, but each program is unique and there are many instances where dedicated SW/FW costs less to develop and the host hardware space, weight, power and cost (SWaP/C) burden is lower.
- Assigning a dollar value to the risks of having to make system specifications changes to a system already proceeding under full steam because as the requirements become more fully defined later in the build cycle is a complex challenge. The "roll of the dice" with SRSE is that at least one of the designs being carried forth is able to accommodate the requirement changes later in the build cycle with minimum costs to budget and schedule.

One additional element in determining if a SRSE approach is warranted, beyond the desire to move expeditiously, is the contract type. A Cost-Plus Percentage or a Cost-Plus Fixed-Fee may be better served by a traditional, fully defined requirements type of program; there is little incentive to control costs that are borne by the customer. But SRSE execution could offer a Firm Fixed Price or Quick Reaction Capability/Best Effort program some cost and schedule advantages, such as increasing the profit margin of the overall budget. Such is to be negotiated among all of the stakeholders.

Enablers

Once a SRSE approach has been made, there are design considerations that must be included so that the system maintains as much flexibility in delivering performance and meeting the requirements, as late in the build cycle as possible and even after placed in situ, such as the following examples suggest:

Hardware that allows for reconfiguration:

- o Field-Programmable Gate Arrays (FPGAs) with reprogram capability
- o Tunable radio frequency filters
- o Common optical path/common aperture for multi-waveband sensors
- o Digital-pixel Read Out Integrated Circuits (DROICs) w/ software defined reconfigurable read

Software defined systems for reconfiguration or upgrades:

- o Waveforms for telecommunications
- o Updatable modem processing chain elements for RF and laser communication systems
- o Processing algorithms (for sensor data, etc.)
- o Interfaces - MOSA supporting common interfaces
  - Sensor open systems architecture
  - Future airborne capability environment

Software/firmware:

- o Common interface soft blocks

Artificial intelligence/machine learning:

- o Self-reconfigurable
- o Auto-retasking [11].

With respect to the timeline

- o Previously deployed systems may have to be interfaced to and be interoperated with, so elements of backward compatibility for information interchange may be required [12].
- o Immovable deadlines must be accommodated; war fighter needs, major deployment operations, spacecraft launch dates, new product introductions

The incorporation of the ability to define performance and function late in the build cycle or while in the field through component and subsystem performance margin, increases the duration of system relevance after deployment. But as such additional margin requires additional capability, it must be remembered that additional capability requires additional SWaP/C.

Minimizing requirement changes

Scope changes are a major source of program disruption; they cause delays, increased costs, and risks to meeting SOS performance objectives. One of the ways to reduce the disruption caused by changing requirements is to reduce the period of time that changes can be accepted. There are a number of tools available to help minimize scope creep:

Reduce the overall program duration:

- o Carry out as many paths concurrently as possible. This may be difficult to accomplish in all cases, as the proverbial "a baby can't be made in one month by nine women" frequently applies. There are certainly operations that have to be carried out serially as with any program execution, but beware of the tendency to determine a schedule based on assumptions made by similar, previously executed serial programs. Question assumptions and lessons learned, and their applicability if based on prior work.
- o Schedule Reductions can be accomplished by parallel loading personnel; however this entails utilizing resources as defined by skill code, and abandoning identifying specific individuals for roles unless absolutely necessary and defined by person-specific capabilities.

Commit to points of no-return early:

- o Committing to long-lead items can "seal the deal;" it is impossible to accommodate requirement changes after committing to a particularly unique piece of hardware that comprises a significant portion of the SOS, especially when that item involves a substantial portion of the budget.
- o Key personnel required for early stages of trade space analysis can force an end-user's hand by acknowledging that exhaustive, continuous trades and redesigns cannot be accommodated open-ended.

Remove schedule margin:

- o Engineers typically like to have some margin in cost and schedule and find ways to sneak them into the Integrated Master Schedule. When trying to minimize the opportunity for a stake-holder to change requirements, use a 100% success approach, especially for processes that are well defined and have a high manufacturing readiness level (MRL) application factor.

- Question (even blindly) every estimate provided by the program staff as to how much unnecessary margin is being retained
- Schedule margin should be generated, retained and dispensed by the Program Manager
    - The PM has final authority for program execution, schedule and budge adherence.
    - The PM and Project Engineer have authority to assign schedule margin as an element in risk mitigation (for re-work, second "spins" of design and fabrication, etc.)

Identify changes as "Out of Scope":

Programs are executed to contracts, and contracts are specific as to what the final deliverable shall be. While caution should be observed with regard to protecting a relationship between a Stakeholder and the SOS provider, changes in requirements can be identified as having both cost and schedule impact. The End User will want to change things as long as there is minimal impact. By identifying that the smallest change has implications to the budget due to:

- Overhead costs associated with increased the contract period of performance
- Increased labor costs as less efficient personnel are brought in to cover previously scheduled resource roll-off. New personnel have to be brought up to speed on program execution; new personnel may be less experienced and require more hours to accomplish what seasoned professionals can provide.

Accommodating unavoidable requirement changes

No matter the effort to minimize them, requirement changes are bound to occur to some extent during program execution. They could be the result of newly available technologies that become part of the SOS design, or the need to change SOS capabilities. The danger of accepting changes after execution begins will come in the form of resistance to backtracking within the program. Resistance will come from internal management who wants to close the contract and deliver the system, as well as from the End User once the costs to accommodate requirement changes are identified.

As an example, suppose the change request came after a successful Critical Design Review. The CDR is an activity that is time and cost intensive; it may also represent a payment milestone. Program Management is inclined to keep the customer happy by not imposing additional fees to execute a Delta-CDR (a CDR that captures changes or open items from the CDR that require close-out), and the customer would certainly prefer not to pay them or accept the schedule burden. However, changes to an already-designed system have to be analysed with respect to impact to other systems, to interfaces, to Beginning-of-Life (BOL)/End-of-Life (EOL) performance.

Nonetheless, any requirement changes accepted must be categorized, because only one type of change comes with unavoidable costs, and that is the "must have" requirement. The other requirement change categories are "nice to have" and "gold plating," neither of which should be acceptable if they cause impacts to cost and schedule. Every new requirement or "to be determined (TBD)" retired, should be tracked in a scope log. Initially identified TBDs should be accounted for in the initial budgetary estimate; unplanned scope change costs and are to be fed back to the customer as financial and schedule overruns (which will generate an incentive in the customer to minimize scope creep). Again, minimizing cost and schedule is the goal of SRSE, so all efforts should be taken to remain true to its objectives [13].

Executing in a SRSE environment requires that an atmosphere of trust exist between all stakeholders, otherwise the march towards a common objective will degrade into a string of activities that have no further objective other than the covering of one's own behind. Chief among the ideals that must be exercised is an understanding that the rapid execution will lead down engineering cul-de-sacs, with associated costs and schedule that are inevitably lost, discarded, or useless in the final design iteration. Associates cannot be punished for telling the truth about such dead-end pursuits; their info has to be treated as input and updates to the tracking tools.

Additionally, customer expectations must be managed (especially up front). Written contracts addressing agreed upon goals and deviations allowed need to be revisited often so there is no possibility that any stakeholder acts astonished and presents a "I never knew we weren't going to be getting X, or Y, or Z" posture. Establishing clear lines of frequent, recorded communication is useful to help keep expectations consistent and prevent unreasonable hopes, but it is still no guarantee of end-user happiness with program deliverables.

Finally, as for the contracted company providing the system, SRSE is being undertaken in an era of limited budgets and tight cost controls. A system provider should not accept penalty clauses when delivering on a SRSE effort, because there will almost assuredly be some differences between the contracted-for and the delivered system. It is not desirable to be penalized for missing a delivery by one day or a performance metric by 2% due to execution under SRSE.

Risk

It can be assumed that risk is minimized when programmatic requirements are fully defined. However, when the requirements are sparse and the system isn't fully defined, risks will be Present and must be tracked and quantified, either with a dollar amount, a schedule impact (delay), a reduction in performance or all three. The Risk Cost is calculated by multiplying the Impact by the probability, as would be determined in any nominal programmatic risk analysis. The probability is defined according as one of three types: the probability of occurrence if nothing is done, the probability of occurrence with some mitigation plan implemented, and the probability of occurrence with a fully defined recovery plan. Each of these risk types has successively lower impact. The cost of each individual risk can then be calculated:

$$\text{Risk Cost} = \text{Impact} \times \text{probability of occurrence} \qquad (2)$$

For the fully defined system, it is expected that the risks are minimal.

For the Sparse Requirements Systems, the risks are present but well defined.

No matter which type of program execution is chosen, it can be difficult to determine costs as risks are not fully understood, identifiable or can be assigned an impact. However, as risks materialize, their impacts will manifest themselves upon the end-user and supplier as additional cost and schedule delays. The contract type defines who it impacts:

- Cost-plus programs (cost plus fixed-fee, cost-plus fixed percentage) impact the customer
- Fixed-price programs impact the supplier with cost, and the end user with schedule [13].

But as the field of Risk Management is well covered by any number of seminal texts, the author refers the reader to them and will not address risk management any further herein.

Part two - Implications of SRSE for assured autonomy

As the 21st century has seen the advent of systems that implement artificial intelligence and machine learning along with operational autonomy, there are implications for utilizing sparse requirements systems engineering on autonomous systems:

- A fully defined input environment is absent by sole virtue of utilizing SRSE. Without a fully defined set of requirements, the complete set of inputs cannot be described.
- A fully defined output or set of responses by learning enabled systems is absent, by virtue of the definition of learning enabled systems. They are relied upon to determine their own reactions based on inputs and how their input classifiers work upon deployment as well as how they evolve. In the absence of an a priori defined set of inputs, a fully defined set of responses is not possible.

Autonomous systems

- In order to understand the impact utilizing SRSE has on developing LE-CPS, it is sensible to first examine Autonomous Systems, those with the ability to accomplish their objectives independently without human supervision. They are proliferating and the United States Department of Defense is particularly interested in incorporating them in the array of deployed systems, having identified a range of critical capabilities and objectives to support:

Enhanced situational awareness

- Cognitive workload reduction
- Force protection
- Unmanned vehicle C&C
- Cyber defense
- Logistics
- Image processing & pattern recognition
- Sensor information extraction
- Satellite constellation management
- Underwater vehicle C&C There exist complex and unpredictable environments where autonomous systems can generate what DARPA (the Defense Advanced Research Projects Agency) calls "high regret unintended consequences," such as the much-publicized Tesla automobile crashes where the failure of the autonomous driving systems has generated fatal results [14].

If trust in autonomous systems cannot be guaranteed, then such systems will either:

- Not be adopted or deployed.
- Be deployed with an acceptance of potentially undesirable or unsafe operation, along with the unintended consequences and damages [15].

According to Dr. Sandeep Neema, a DARPA Program Manager, several factors impede the deployment and adoption of autonomous systems:

1. In the absence of an adequately high level of autonomy that can be relied upon, substantial operator involvement is required, which not only severely limits operational gains, but creates significant new challenges in the areas of human-machine interaction and mixed initiative control (human *vs.* AI prioritization of commands).
2. Achieving higher levels of autonomy in uncertain, unstructured and dynamic environments, on the other hand, increasingly involves data-driven machine learning techniques with many open systems science and systems engineering challenges.
3. Machine learning techniques widely used today are inherently unpredictable and lack the necessary mathematical framework to provide guarantees on correctness, while DOD applications that depend on safe and correct operation for mission success require predictable behavior and strong assurance." [16].

There are three environments that the LE-CPS will operate in:

1. **The cooperative environment**: All LE-CPS elements share a common objective and cooperate to achieve their individual objectives as well as facilitate other LE-CPS' member objectives. Such actions may involve communicating intent and sharing telemetry.
2. **The non-cooperative environment**: Like the cooperative environment but with no coordination among constituent group members. This environment merely assumes that all members are working toward their respective individual goals with no coordination among any constituent member.
3. **The uncooperative environment**: Where other elements in the environment, not necessarily part of an extended group, are actively seeking to prevent achievement of mission goals. This may be through hostile intent (e.g. - attack with weapons systems) or manipulation of the environment (e.g. – GPS "spoofing).

The autonomous system has to be able to demonstrate safety that is as good as or better than comparable systems operated by humans (with the assumption that safe operation of any system is the objective; malicious use is specifically excluded from this premise). To that end, the goal of the [DARPA] Assured Autonomy Program is to create technology for continual assurance of Learning-Enabled, Cyber Physical Systems (LE-CPSs). Continual Assurance is defined as a guarantee of the safety and functional correctness of the system:

- provided provisionally at design time
- Continually monitored, and updated
- Evaluated at operation-time as the system and its environment evolves [16].

This may be less of a challenge when operating in a cooperative environment, such as the autonomous operation of vehicles driving on a divided highway: all vehicles are travelling in approximately the same direction, at approximately the same speed, with similar objectives to completing a journey safely. Events such as a child chasing a ball onto the road surface, on-coming traffic venturing into our path, or cross-street traffic causing a conflict, while all low-probability events in this scenario must nonetheless be accommodated and corrective actions anticipated. A suitable response to these conflict examples might be to bring vehicles to a halt to avoid the conflict, and then resume once the conflict is resolved.

At the other end of the spectrum, an Unmanned Aerial System, such as a combat aircraft, may operate in a cooperative environment with other same-force assets, but most assuredly will operate in an uncooperative environment, one which is adversarial and specifically designed by the enemy to present situations intended to elicit unplanned responses, presumably with the objective of gaining superiority in an engagement.

Challenges for autonomous systems

The Institute for Defense Analyses has identified ten challenges for test and evaluation, verification and validation, specific to autonomous systems (Table 2).

Autonomous system assurance

Non-learning systems: Currently, autonomous system assurance can be achieved for non-learning systems. Such systems are based on fully defined set of operational stimuli and a closed set of known responses (aka "evidence"). The system model is subjected to formal verification, with simulations covering the full range of stimulus and response; inputs that are unrecognized are dismissed as outside of the input space and not acted upon. Comprehensive testing

Table 2: Ten challenges for autonomous systems.

S. No	Autonomous System	Function
1	Instrumenting machine thinking	Diagnose the causes of incorrect behaviour. Engineering: coding errors, bad algorithms, and inadequate training data. Operational: perception, sensing, algorithms, actions taken.
2	Linking system performance to autonomous behaviours	Understanding how the system's various autonomous capabilities interact to enable or hinder mission execution. Requirements: The requirements specification for autonomous behaviour is can be problematic if the specifications are incomplete and based on human behaviour analog.
3	Comparing AI models to reality	Degree to which the internal modeling of reality supports accurate Perception, valid Reasoning, and effective Selection. Not generally a function of how detailed the model is (resolution) or how closely the models represent reality (fidelity). It is a function of whether the right information is incorporated into the model and that the resolution and fidelity are sufficient for mission needs
4	CONOPS and training as design features	Traditional systems engineering develop CONOPS and training are developed after the technical solution. For autonomous systems, where the system operates itself and interacts autonomously with humans, the CONOPS, tactics and training) are part of the system design, and will have to be identified, verified, and validated much earlier in the development process.
5	Human Trust	In human-machine teaming (HMT) contexts, how the humans behave (and thus how well the system performs) depends in part on the humans' psychological attitudes toward the autonomous systems. "Trust" is the term generally used to describe those attitudes, though in practice those attitudes are generally more nuanced and multi-dimensional than simply asking "how much do I trust it?" In order to design, debug, and assure performance, TEV&V will need to be able to measure the various dimensions of trust, to support understanding of how trust affects human/autonomous system performance.
6	Elevated safety concerns and asymmetric hazard	Autonomous systems potentially take many of the decisions underlying routine safety out of the hands (and minds) of operators, and depend instead on complex software that allows the system to "operate" itself. During Developmental Test and Evaluation and into Operational Test and Evaluation, it is likely that the software will still contain major bugs and that the algorithms and training data being used might not be the final choices. The potential exists - especially for weapon systems, highly-mobile systems, or other systems that could be dangerous in the hands of an unreliable operator, for circumstance to develop that are hazardous not to the autonomous system, but to the humans interacting or in close proximity. Few active research programs today are addressing applications with highly asymmetric hazard functions
7	Exploitable vulnerabilities	AI based on machine learning has its own set of potential vulnerabilities (cyber, electronic, physical), both during training of the AI and in operation. There are circumstances where they are more vulnerable than comparable human operated systems. TEV&V of autonomous systems will need to be aware of this expanded attack surface.
8	Emergent behaviour	Unanticipated emergent behaviour resulting from the effects of complex operational environments on autonomous or semi-autonomous systems. Developing T&E methods to analyze the potential for emergent behaviour in order to avoid it will be central to providing assured dependability for autonomous systems.

9	Post-fielding changes	Systems that employ unsupervised learning or adaptive control during operations will change their behaviour over time. This creates a need not only for periodic regression testing, but also for predictive models of how post-fielding learning might affect system (or team) behaviour. Traditional TEV&V is concerned with the effectiveness and suitability of the system as it is today. Needing to be able to predict the effectiveness and suitability of the system as it might become is a new requirement when compared to non-learning systems.
10	Verification and validation of training data	Supervised or reinforcement learning depends critically on the data used to train the autonomous system. It is an axiom that "the intelligence is in the data, not the algorithm." Supervised learning data must not only be representative of the range and type of data the system will take as input during operations, but must also be correctly and completely labeled. This leads to a need for verification, validation, and accreditation (VV&A) of the data used to train the AI that is similar to the need for VV&A of modeling and simulation.

prior to deployment prevents unanticipated system actions and assures desired autonomous behavior. The defined safe response space that represents desired and safe actions, can be designed with sufficient segregation from the unsafe region, that real-time monitoring of the actions the CPS are not required (Figure 1) [16].

Non-learning Cyber Physical Systems (CPS) are relatively simple in terms of components and interaction. They are composed of sensors, actuators and controller/plant components. Like any engineering system, a defined set of tests can be applied to insure that the requirements of the CPS are as anticipated. Evidence (E) is gathered in the form of laboratory and field test data, and validation of correct system operation supports the claims of safety and security, assurance being comprised of a logical chain of evidence supporting a claim (Figure 2). The assurance that can be claimed is limited by the testing that is executed.

Incorporation of system models: By adding comprehensive system models of the components of the non-learning system, the CPS can be defined with a higher degree of operational fidelity, and the claims of assurance are strengthened. The model validity is confirmed by its prediction of stimulus/response as observed in training data in controlled environments. It can also be utilized to judge responses to expected stimuli that have not been presented in training data sets through simulation and model-based formal verification. Overall, adding a system model can provide additional avenues of system behavioral prediction (interpolated or reasonably extrapolated) to provide further evidence of assurance (Figure 3). Models typically employ simplifying assumptions so they are not 100% representations of reality; as such testing is still required to validate the models [16].

Learning systems: Unlike non-learning systems, learning systems cannot offer the guarantees of intrinsic high-regret unintended consequence avoidance. The initially trained response space is subject to modification due to the system being inherently able to modify its response to inputs that lie outside of the initial training input data. The response space is no longer fixed with well-defined boundaries; it is free to grow beyond what was initially defined and to capture it completely is an impossible task. Besides the objective of delivering desired behavior, the learning system must prevent the response space from degenerating into the unsafe system operation region. The formal methods of software engineering can be applied to autonomous system development to the extent that as much about behavior can be designed into the system during creation as possible, to keep the scope of empirical testing manageable. But for learning systems that are constantly evolving, this will not be all-inclusive because once deployed, any change to the system through learning and evolving invalidates all of the assurances initially obtain, that is unless reconfirmed with the evolved system [17].

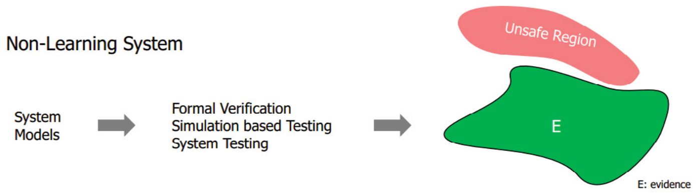

Figure 1: Non-learning system architecture.

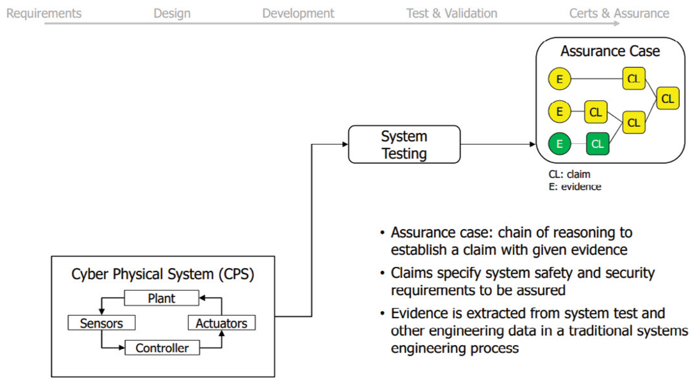
Requirements
Design
Development
Test & Validation
Certs & Assurance
Assurance Case
E
CL
CL: claim
E: evidence
System Testing
Cyber Physical System (CPS)
Plant
Sensors
Actuators
Controller
• Assurance case: chain of reasoning to establish a claim with given evidence
• Claims specify system safety and security requirements to be assured
• Evidence is extracted from system test and other engineering data in a traditional systems engineering process

Figure 2: Non-learning system functional flow.

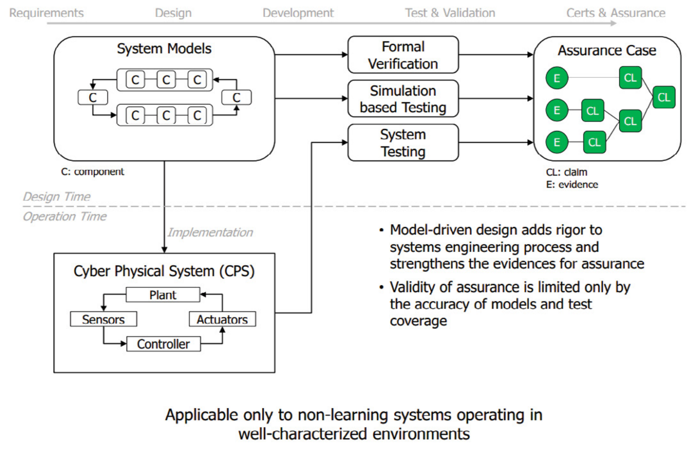
Requirements
Design
Development
Test & Validation
Certs & Assurance
System Models
C
C: component
Formal Verification
Simulation based Testing
System Testing
Assurance Case
E
CL
CL: claim
E: evidence
Design Time
Operation Time
Implementation
Cyber Physical System (CPS)
Plant
Sensors
Actuators
Controller
• Model-driven design adds rigor to systems engineering process and strengthens the evidences for assurance
• Validity of assurance is limited only by the accuracy of models and test coverage
Applicable only to non-learning systems operating in well-characterized environments

Figure 3: Non-learning system functional flow with system model incorporated.

Adding learning enabled components (LECs) to the configuration in the system models as well as in the Autonomous LE-CPS itself, introduces uncertainties and only makes obtaining assured autonomous operations more difficult. The response space broadens, and unpredictable operation is almost a certainty.

Inability to guarantee predictable behavior presents itself and so Sclaims of assured autonomy are no longer possible (Figure 4).

The Department of Defense maintains an office of Research and Engineering, managed by an Under Secretary [18,19]. Within the Office is the Autonomy Community of Interest (COI) Test and Evaluation, Verification and Validation (TEVV) Working Group. Recognizing that autonomous systems present unique challenges in designing for safe and predictable operation, it was acknowledged that,

“The notion that autonomous systems can be fully tested is becoming increasingly infeasible as higher levels of self-governing systems become a reality. As these systems react to more environmental stimuli and have larger decision spaces, the standard practice of testing all possible states and all ranges of inputs to the system becomes an unachievable goal. Existing TEVV methods are, by themselves, insufficient for TEVV of autonomous systems; therefore, a fundamental change is needed in how we validate and verify these systems. For example, agent- and model-based design and verification techniques and simulations are gaining ground as methods to guarantee assured safety in multi-agent systems in industry; however, acceptance of this paradigm shift toward simulation-based validation and verification has yet to replace many physical tests in military programs of record.” [20].

Four challenges to autonomous systems were identified in the 2015-2018 Technology Investment Strategy Report that concern assured autonomy in learning systems:

1. **State-space explosion:** Because autonomous learning systems have algorithmic decision spaces that are dynamic, the outputs are non-deterministic. Further, as the systems grow they become increasingly complex; interactions grow exponentially and the state space cannot be comprehensively searched or tested.

2. **Unpredictable environments**: Autonomous learning systems’ objectives are to perform and deliver desired results in unknown, untested and even contested environments. They include agents capable of making their own decisions. As the multiplicity of different situations experienced accumulates, the state space of responses grows and contributes to the state space explosion. Assurance of correct behavior is vital but the potential for unplanned action is growing, contributing to the state space explosion

3. **Emergent behavior**: The challenge is to identify and constrain potentially harmful behavior at autonomous system design time, when the behavior that one wishes to control doesn’t emerge until subjected to operational environments and stimuli in the field. As the autonomous system evolves over time and resembles less and less the system initially conceived, seemingly insignificant factors can generate wildly unforeseen and undesired actions. These emergent behaviors have little if any chance of being caught during initial verification and validation testing.

4. **Human-machine interfacing and communication:** An autonomous system that is completely defined and won’t evolve has no need to generate “trust” in its operation to the human monitoring or sharing control. But if the system is able to venture into behaviors that are unanticipated, how is a CPS to provide assurance? [20].

In order to address these challenges, DARPA identifies two new capabilities to be introduced to provide feedback and explanation (to the system or to the human on/in the loop) that previously unknown stimuli are not generating undesired actions: guarding and monitoring. Guarding prevents an unsafe response no matter the directive; it is a forcible denial of venturing into unsafe operation space. Monitoring is a real-time complete state-space description (Figure 5).

But monitoring is not the explanation for system behaviour and the monitoring itself can present challenges:

- For advanced autonomous systems, a complete state description will generally be too complex to be able to derive useful information about the system behavior and make timely interdictions. The result can be a case of “one won’t know ‘what’s enough’ until presented with ‘that’s more than enough.’”

- Examining a subset of the state description elements to limit the complexity runs the risk of being inadequate to support behavior explanation. The task of identifying the minimal set of measurements that will be understandable for diagnosis, prediction, bounding (actions) and developing trust in the autonomous system are not trivial. Decimation of some elements of a complete description for the purposes of manageability runs the risk of overlooking key characteristics.

The lack of a completely known response space has instigated a new approach to address assured autonomous operation. DARPA has divided the problem of assured autonomy for systems into three technical areas (TA’s) (Figure 6).

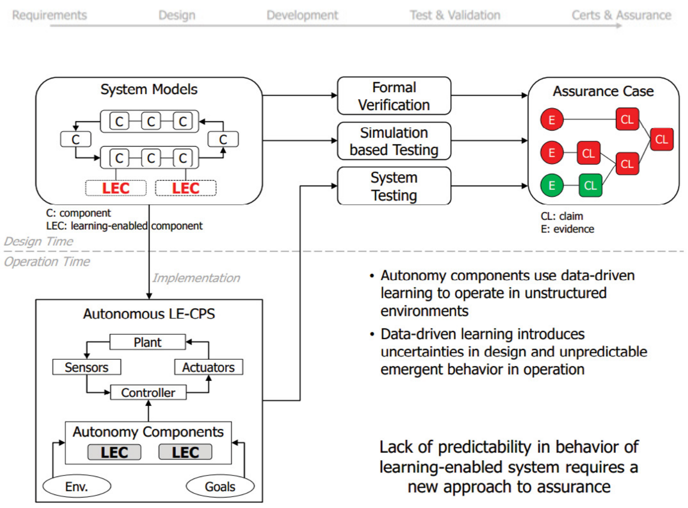
Requirements
Design
Development
Test & Validation
Certs & Assurance
System Models
C
LEC
C: component
LEC: learning-enabled component
Formal Verification
Simulation based Testing
System Testing
Assurance Case
E
CL
CL: claim
E: evidence
Design Time
Operation Time
Implementation
Autonomous LE-CPS
Plant
Sensors
Actuators
Controller
Autonomy Components
Env.
Goals
• Autonomy components use data-driven learning to operate in unstructured environments
• Data-driven learning introduces uncertainties in design and unpredictable emergent behavior in operation
Lack of predictability in behavior of learning-enabled system requires a new approach to assurance

Figure 4: Learning system functional flow.

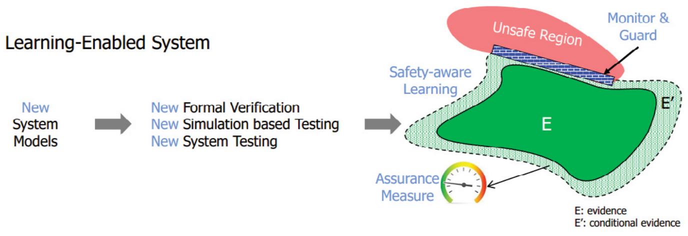

Figure 5: Learning system architecture.

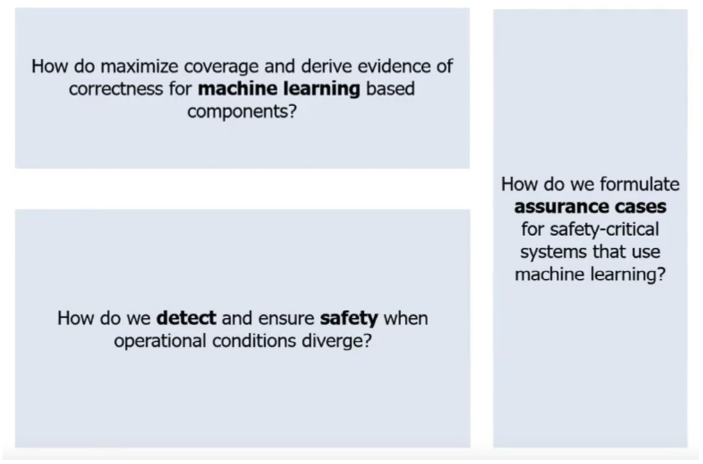
How do maximize coverage and derive evidence of correctness for machine learning based components?
How do we detect and ensure safety when operational conditions diverge?
How do we formulate assurance cases for safety-critical systems that use machine learning?

Figure 6: Learning system challenges.

- **The TA1 region:** Design for assurance, is where classical systems engineering makes accommodation for incorporation of learning enabled components, and what the implications are for formal verification of systems with learning enabled components.
- **The TA2 region:** Operate with assurance, is where operations incorporate safety features to comprehend and act upon real-time monitoring of behavior and safe operation. It is not unexpected that the aforementioned monitoring and guarding functions reside in this TA.
- **The TA3 region:** Quantify assurance, is where evidence of safe operation and actions that supports claims of assured autonomous non-anomalistic operation is generated. In the best case scenario, a metric is assigned to the operational state that can facilitate affirmation of safe operation.

The Learning Architecture last described in Figure 4 is further modified by adding additional elements to carry out guarding and monitoring:

- A Safety Aware Learning capability is added to the Autonomy Components of TA2.
- Artifacts of the environment and operational goals are determined and sent to the Monitor and Guard functions from T1 to T2.
- Dynamic Assurance Monitors and Guards are implemented in TA2.
  - o They function to actively prevent known unsafe system responses.
  - o TA2 also provides "return to safe operation" vectors out of known unsafe operation space, back into expected operation.
- A real-time Assurance Measure is utilized to provide closed loop feedback to the autonomous component(s)' learning in TA3.
  - o Previous responses (desired and undesired) are utilized to delineate the scope of controller behavior, with the objective that such delineation ultimately minimizes undesired responses, as described in (Figure 7).
- Finally, Dynamic Assurance provides an element of quantitative measure of the confidence that the action taken is congruent with the actions desired. This includes:
  - o Desired action
  - o Self-safety of the host system
  - o Safety to entities within the sphere of system influence

Tools for assured autonomous operation

While there the number of software packages that facilitate nearly all traditional engineering (computer aided design, finite element analysis, computational fluid dynamics, multi-physics analysis), there are currently few that facilitate design and analysis of autonomous system architectures, let alone provide environments within which such systems can be stress tested and safe operation (or lack of it) confirmed.

Architectural Analysis and Design Language is a modelling tool that supports software and hardware modelling to master complex systems; autonomous system architecture and the means to ensure safe operation can be captured within its framework. The field of autonomy assurance is dynamic, and while certainly not the only tool and method, assurance case languages based on architectural models are finding favour in determining the required constructs for safe operation of complex autonomous systems. One such environment, Resolute, tracks design changes that might invalidate some aspect of an assurance case [21-23].

Assurance Monitoring and Control provides an environment where stimuli which were not part of the initial training data are accommodated (Figure 8).

- Safe Reinforcement Learning enables adaptation to and incorporation of previously un-encountered operational

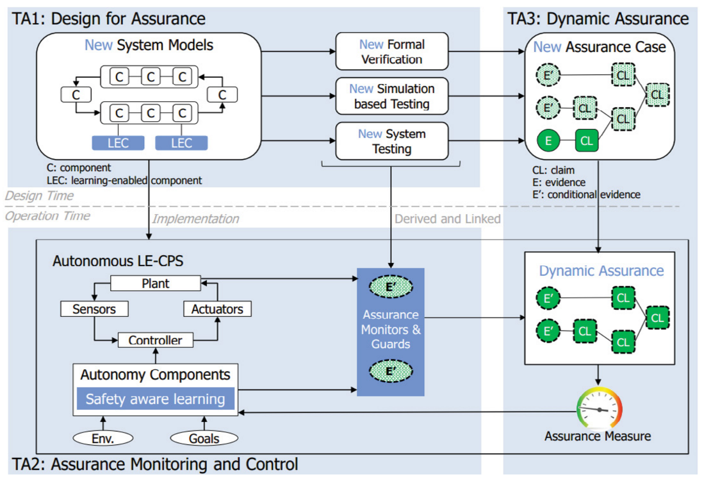

Figure 7: Learning system functional flow with system model, dynamic assurance, assurance monitors and guards, and safety aware learning incorporated.

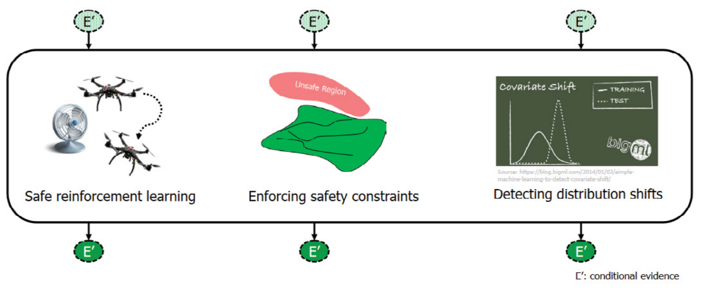
E'
Safe reinforcement learning
Unsafe Region
Enforcing safety constraints
Covariate Shift
TRAINING
TEST
Detecting distribution shifts
E': conditional evidence

Figure 8: Assurance monitoring & control.

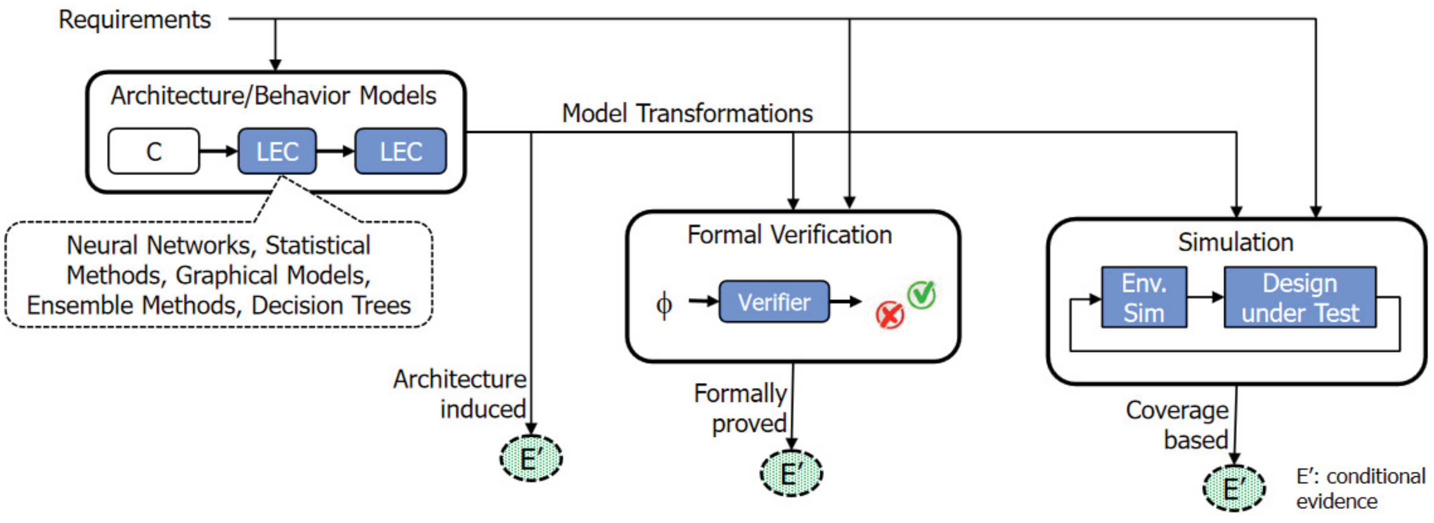

Figure 9: Design for assurance.

stimuli, by insuring that the Learning Enabled Components are able to comprehend the new environment. The new stimuli are managed by the most appropriate decision and classification algorithms so that adherence to a safety paradigm is followed.

- Enforcing Safety Constraints insures that System Responses do not stray from reliable, predictable or recoverable behaviors [24].
- Application of Covariate Shift determine if there's been a significant difference between training data set and real-world stimuli that requires system retraining or other adjustments to the CPS and system model to re-assure safe operation [25].

The formal representation of assurance can impose a significant computation penalty on the system, manifested as SWaP/C, as verification of desired actions must be carried out in near-real time. Any trend toward unsafe operation must be corrected almost immediately before "high regret unintended consequences" are generated [26-30].

Engineering for assurance

In order to achieve autonomous system operational assurance, systems engineering, analysis, design, and the required engineering disciplines are implemented to define the system. Requirements are linked to behaviour of the passive and learning enabled components; these in turn drive specification of neural networks, decision trees and other classifier/decision algorithms. Formal Verification, traditionally implemented as functional testing, provides stimulus/response Conditional Evidence of desired behaviour based on training data sets. Simulated Environments provides further Conditional Evidence of desired behaviour for input stimuli outside of the initial training data set. Conditional Evidence can be thought of as desired responses elicited in well controlled stimulus environments (Figure 9).

RESULTS AND DISCUSSION

Sparse requirements systems engineering describes program execution in an environment where not all of the requirements are fully defined prior to commencement, and is a contemporary reality driven by reduced resources available to the system provider. All stakeholders must be fully engaged and knowledgeable of costs, schedule, expected system performance, as well as risks and mitigations of the proposed plans of execution. The execution must be approached by carrying a multiplicity of possible designs that will span the unknowns of the current solution trade space, with a full awareness of the costs and risks associated with each. Those designs that are identified as not workable as the program progresses and more requirements are levied to more fully define the objectives must be left eliminated from the solution space. Standard best practices and systems engineering tools and practices such as design, analysis, test, comprehensive interface definition, remain applicable to the program execution. SRSE is a means to allow a program to proceed and minimize costs, schedule, risks, and scope creep, while taking advantage of opportunities to accelerate delivery. But the advantages offered by Sparse Requirements Systems Engineering, such as reduced schedule and costs, can deliver for today's end users if executed properly and attentively.

Assuring safe autonomous operation in LE-CPS built under SRSE paradigm are really two sides of the same coin. It must be remembered that LE-CPS cannot be trained with a comprehensive set of expected input stimuli and produce a closed set of responses; their very nature is to be constructed to respond to new stimuli and unanticipated responses are almost guaranteed. The undefined requirements in an SRSE approach represent a larger region in the unknown stimulus space for LE-CPS.

- With respect to Systems Engineering, those requirements that are known are analogous to the initial training data for a LE-CPS.
- Those requirements that are yet to be determined or defined, are analogous to the unanticipated stimuli on the input side of a LE-CPS

The TBD requirements for an autonomous system can be thought of as a head start toward operation in a new environment, one that has the system encountering stimuli outside of the realm of its initial training data set, for the undefined requirements manifest themselves as lack of operational constraints. Stimuli that do not fall into or can be classified as stimuli related to the initial training data with a high degree of certainty, have to be treated as new data, with the attending risk of unanticipated responses.

CONCLUSION

So in order to implement SRSE to move quickly and concisely in developing and fielding a new system on an abbreviated timescale, the additional risk that has to be carried forward is a highly likelihood that for an LE-CPS, there is greater likelihood that high regret unintended consequences are more likely than for similar LE-CPS systems that are granted greater development cycle time. The only way to achieve safe operation for LE-CPS, especially those created under SRSE, is to eliminate reliance on the requirements verification compliance tracking and verification Validation approach of classical systems engineering:

- Formulate assurance cases for learning systems and determine how to derive evidence of correct operation
- Implement n-dimensional correct response spaces and employ a means to return the system to safe operation when a generated response to a stimulus is unsafe or undesired
- Incorporate a means to deliver dynamic assurance verification as part of the system; it will have to update and evolve as the system accrues time in the field.

REFERENCES

1. Kendrick T. Identifying and managing project risk, New York: American Management Association. 2015.
2. Estefan JA. Survey of model-based systems engineering (MBSE), Incose MBSE Focus Group. 2007.
3. Association of the United States Army, Rapid Equipping and the U.S. Army's Quick-Reaction Capability. 2015.
4. Brissett W. The creation of a space rapid capabilities office. 2017.
5. Serbu J. Federal news network. 2018.
6. Marine Corps Warfighting Laboratory. Marine Corps stands up rapid capabilities office. 2017.
7. NASA. Rapid spacecraft development office - mission statement. 2016.
8. Insinna V. If we can change, we can win. Defence News. 2021;1:32-33.
9. INCOSE. Systems engineering book of knowledge. International Council on System Engineering. 2019.
10. Oppenheim BW, Haskins C. Powerful opportunities to improve program performance using lean systems engineering and lean program management. In advances in systems engineering, Reston, American Institute of Aeronautics and Astronautics. 2016;269-286.
11. Puckett R. Interviewee, Senior principal systems architect. 2019.
12. Henshaw M. Systems of systems engineering for nato defence applications. Loughborough University, Leicestershire. 2020.
13. Vallance D. Interviewee, senior program manager. 2019.
14. The Associated Press, Tesla's self-driving Autopilot system under scrutiny after 3 deadly crashes. 2020.
15. Neema S. Assured autonomy seeks to guarantee safety of learning-enabled autonomous systems. Darpa Defense Advanced Research Projects Agency. 2017.
16. Neema S. Assured autonomy. DARPA Defense Advanced Research Projects Agency. 2017.
17. Brian AH. The status of test, evaluation, verification, and validation (TEV&V) of autonomous systems, Institute for Defense Analyses, Alexandria. 2018.
18. Neema S. Asured autonomy. DARPA. 2020.
19. Department of Defense. Office of the under secretary of defense for research and engineering. 2021
20. Department of Defense Office of Research and Engineering. Utonomy community of interest (COI) test and evaluation, verification and validation (TEVV) working grouptechnology investment strategy. DoD research & engineering autonomy community of interest (COI) test and evaluation, verification and validation (TEVV) working group technology investment strategy 2015-2018 office of the assistant secretary of defense for res & eng, washington, D.C. 2015.
21. Institut Supérieur de l'Aéronautique et de l'Espace. The architecture analysis and design language: An overview. 2021.
22. Darren DC, Isaac A, Ram S, Taejoon B, Sanjai R. At al. Run-time assurance for learning-enabled systems. 2020.
23. Andrew G, John B, Darren C, Konrad S. Mike W. Resolute: An assurance case language for architecture models. 2014.
24. Miller MJ, Kevin MF. Unmanned systems integrated roadmap. United States. Office of the Under Secretary of Defense for Acquisition and Sustainment, Washington, D.C. 2018.
25. Martin FJ. A simple machine learning method to detect covariate shift. Big ML Blog. 2014.
26. Blanchard BS, Fabrycky WJ. Systems engineering and analysis, upper saddle river: Prentice Hall. 2011.
27. Richard EW, Rafael CG. Digital image processing, upper saddle river: Prentice Hall. 2002.
28. Powell V. Image kernals explained visually. 2021.
29. Wikipedia. Kernel image processing. 2021.
30. Neema S. Workshop on safe autonomy: Learning, verification and trusted operation of autonomous systems - assured autonomy. C3.AI - Digital Transformation Institute. 2020;7-8.