16th International Conference on Space Operations, Cape Town, South Africa - 18 - 22 May 2020.

SpaceOps-2020,9,2,3,x269

Automated Software for Crewed Spacecraft - Bridging the Gap from Sci Fi to Reality

Robert C. Dempsey PhD[a], Edward A. Van Cise[b], Michael L. Lammers[c], and Richard S. Jones[d]

[a] *Flight Director, NASA Lyndon B. Johnson Space Center, Houston, Texas, 77058, USA,* robert.c.dempsey@nasa.gov
[b] *Flight Director, NASA Lyndon B. Johnson Space Center, Houston, Texas, 77058, USA,* edward.a.vancise@nasa.gov
[c] *Flight Director, NASA Lyndon B. Johnson Space Center, Houston, Texas, 77058, USA,* michael.l.lammers@nasa.gov
[d] *Flight Director, NASA Lyndon B. Johnson Space Center, Houston, Texas, 77058, USA,* richard.s.jones@nasa.gov

Abstract

With a voice command or a few taps on the console, the spacecraft pivots on a dime at high velocity and gently docks to an orbiting space platform. This is the image most people have of the complex software computations and integrated hardware performance necessary for a spacecraft to successfully perform an automated launch, rendezvous, and docking. Today's reality is that while computer operations are advancing rapidly, science fiction over-simplifies and over-sells current capabilities. This paper discusses the integration of spacecraft computer automation into the operation of one of the United States' new Commercial Crew vehicles - the Boeing CST-100 *Starliner*. Lessons learned by the Boeing Mission Operations team, a private-public partnership with NASA, from conceptual design through real-time operation of the first test flight will be discussed. Focus will center on how operations has learned to use the automated software to their advantage while also knowing how to adjust the automation in response to spacecraft or mission anomalies.

One goal of advanced spacecraft automation is the ability to reduce both the crew workload and the ground control footprint while at the same time increasing spacecraft and mission flexibility. Historically, crewed spacecraft required a large number of operators on the ground to use a plethora of tools to compute nominal and contingency mission trajectories. Moving those sophisticated software tools to being onboard the vehicle can reduce the need for such complex ground support. Given that today's spacecraft software is not yet as capable or as flexible in all circumstances as the computers depicted in movies, there is usually a trade-off between software automation cost and the flexibility of that software resulting in a trade-off between what is performed on the spacecraft and what is left to onboard crew or ground control.

For missions that go beyond the Moon, software that autonomously controls nearly every aspect of a crewed mission will become a necessity given the long time delays between the spacecraft and Earth's ground control teams. The lessons learned by Boeing and its Mission Operations team, through the design and implementation of *Starliner's* hardware and software automation, will be able to inform future public and private spacecraft design. As the technologies and capabilities evolve, incorporating lessons learned in successful low Earth orbit commercial crew vehicle missions, spacecraft designs will continue to improve and be able to better enable safe execution of human missions to the Moon and beyond.

Keywords:
Spacecraft Automation, Human Spaceflight, Spacecraft Safety, Software Configuration Management

Acronyms/Abbreviations:

Commercial Crew Program (CCP), Concept of Operations (CONOPS), Coordinated Universal Time (UTC), Crewed Flight Test (CFT), Emergency Detection System (EDS), Fault/Failure Detection, Isolation, and Recovery (FDIR), Flight Activities Officer (FAO), Flight Operations Directorate (FOD), Keep Out Sphere (KOS), International Space Station (ISS), Orbital Flight Test (OFT), Orbital Insertion (OI), Orbital Maneuvering and Abort Control (OMAC), Manual Delta Velocity (MDV), Mission Control Center (MCC), Mission Elapsed Time (MET), Mission Operations (MO), Sequence Command Input File (SCIF), Tracking Data and Relay Satellite System (TDRSS), Training Simulation Integration Laboratory (TSIL), United Launch Alliance (ULA), Vision-based Software for Track, Attitude, and Ranging (Vis-STAR).

1. Introduction

Spacecraft in movies or books often easily perform amazing feats of computation such as calculating real-time launch profiles, rendezvous trajectories and landings on any type of surface. As is often the case, the reality of today's spacecraft operations is far different. As on-board, radiation hardened computer process capacity continues to expand and mission requirements grow ever more challenging, automated operation of spacecraft is growing rapidly. All of today's spacecraft use some level of automated software but it has largely been relegated to only performing functions associated with Fault/Failure Detection, Isolation and Recovery (FDIR). Routinely used for activity or observation scheduling, automation is usually in the form of short-term plans of events built regularly during the mission by ground operators (e.g., [1]). Even in the most well-known examples of extraterrestrial probes, true automation is often related to specific mission phases (e.g., entry and landing) with humans still performing day-to-day exploration activities (e.g., [2]). Allowing software to control every aspect of spacecraft operations, including computing a rendezvous trajectory, performing proximity operations and docking with other crewed spacecraft, automated undocking both in nominal and off-nominal circumstances, and performing automated re-entry and landing is relatively new in the realm of human spaceflight. Automation in crewed spaceflight has evolved considerably from the 1960s where computers aided the human operators to having the capability of performing every function during an entire mission. While Artificial Intelligence, where the onboard software evaluates its environment, assesses its mission plan and then self-determines the next action, offers significant advancement in automation, its use to date on spacecraft is limited [3]. The Russian "Soyuz" spacecraft, generally regarded as a mostly automated rendezvous vehicle, still requires tracking and ground uplink of target parameters and ground verification of rendezvous burns [4].

NASA's Commercial Crew Program (CCP) required providers to develop fully automated vehicles. After several years of competition, NASA ultimately selected two providers: SpaceX *Dragon* and Boeing with its CST-100 *Starliner*. Both companies conducted uncrewed test flights in 2019 with test flights carrying astronauts to the ISS planned for 2020.

This paper does not discuss details of specific algorithmic approaches such as the proximity operations ones discussed by [5] but focuses on the development, integration and operation of an automated system for a crewed vehicle, in particular the *Starliner*. Below we describe how the flight control team shaped the vehicle's automation and learned how to interface with such an equipped spacecraft. Although focused primarily on the Boeing CST-100 *Starliner* crewed spacecraft, this paper also relies on extensive experience with the International Space Station (ISS). *Starliner's* automated rendezvous and proximity operations was demonstrated on the Defense Advanced Research Projects Agency and NASA Orbital Express Project ([6-8]).

Boeing conducted an uncrewed Orbital Flight Test (OFT) in December of 2019 in preparation for a Crewed Flight Test (CFT) in 2020. Designed to test all aspects of the hardware and software, the mission profile included launch on the United Launch Alliance (ULA) Atlas V, rendezvous with the ISS about 25 hours later, and return to the Earth using parachutes and landing on airbags at the White Sands Space Harbor in New Mexico. As originally envisioned, all of this mission was to be performed via autonomous software, with Mission Control largely playing an oversight role with minimal, periodic instructional uplinks (See also [9]). However, a timing error in the autonomous system caused a key orbital insertion burn to not take place as expected. Due to extensive training and preparation by Mission Control for various scenarios where the automation may not be able to function as originally planned, the control team in Houston was able to uplink a "manual" burn, placing *Starliner* in a stable orbit. Unfortunately, too much propellant was used to allow a rendezvous with the ISS. This software error and its impact on the automated software, as well as the ways in which the ground team were able to intervene, is discussed below.

2. Development

In 2011 NASA outlined the requirements for its commercial crewed vehicles, stating at the outset that the integrated vehicle critical systems shall be autonomous [10]. At the same time, the crew would have the ability to override automation during all phases of flight. While this duality of requirements would lead to a robust and safe design, it also led to a number of challenges due to having to develop a system that could seamlessly transition between completely autonomous and fully manually controlled. Also NASA initially required that all such crew actions would be performed by a single pilot with all other astronauts serving only as passengers. NASA later decided to train a co-pilot to assist in all operations based on lessons learned from the fields of both Cockpit Resource Management and Spaceflight Resource Management [11-14]. The CCP requirement documents stated that fault tolerance for the control of catastrophic hazards would be based on analysis of hazards, failure modes and associated risks. NASA defines a catastrophic hazard as “The <END ITEM> shall be designed such that no combination of two failures, or two operator errors, or one of each can result in a disabling or fatal personnel injury, or loss of the [spacecraft]” [15]. In general, dual fault tolerance or single fault tolerance with dissimilar redundancy for these controls was required. As Boeing designed the *Starliner*, crewed action during safety critical activities (e.g., ascent abort and docking) were sometimes utilized to meet NASA’s safety fault tolerance requirements. By design the crew would nominally not take any action during the entire flight. An exception to this rule was the release of the parachutes after landing which resulted from a NASA concern that software could prematurely jettison the parachutes before touchdown was achieved.

In some contingency cases the crew was the preferred response to a failure, most notably the execution of an ascent abort due to an issue with the *Starliner* spacecraft, such as a cabin leak. In this scenario no spacecraft failure or automated response was deemed appropriate for initiating the complex, and potentially risky, abort sequence to separate the *Starliner* from the rocket and perform an emergency landing in the water and the crew would be in the best position to make the decision to abort. Instead the ground would either direct the crew to abort or the capsule commander would do so based on situational awareness. The health of the launch vehicle is separately monitored by the Emergency Detection System (EDS) onboard the Atlas V and the EDS will take actions to initiate spacecraft aborts in time critical conditions where human response time would be inadequate, such as a catastrophic failure of the launch vehicle [16].

Early in the development of *Starliner*, Boeing entered into a Reimbursable Space Act Agreement with the Flight Operations Directorate (FOD) at the Johnson Space Center to provide Mission Operations (MO) for the program (see also [9]). This allowed the operations teams that would ultimately operate the spacecraft to participate in the design of the vehicle.

Starliner, by design, is a crewed spacecraft. Per the contract with NASA, Boeing was required to perform an uncrewed test flight prior to putting astronauts on the vehicle, this was the intent of the OFT. This presented a significant challenge to the development of the OFT mission. Per design the crew was a “leg” in the fault tolerance. This meant that Boeing was either going to have to add additional hardware (e.g., an additional flight computer to provide additional redundancy), additional software (i.e., to provide an independent processing method), or require the ground to respond or perform additional testing and analysis. Since the OFT was a single mission, Boeing generally elected to not add additional hardware or software. In general, the ground was an adequate method for responding to contingencies, however this did add additional requirements on a communication link with the ground and was only viable in cases where there was enough time-to-effect to allow the ground time to identify an issue and respond. Software changes were made for a few cases where the ground could not be expected to respond in time, such as performing a rendezvous abort if near the ISS and a single computer were to fail. Thus, for the OFT mission without a human crew onboard, oversight and management of a number of safety hazards in close proximity to the ISS took on the approach of utilizing a “three legged stool” (see Fig. 1).

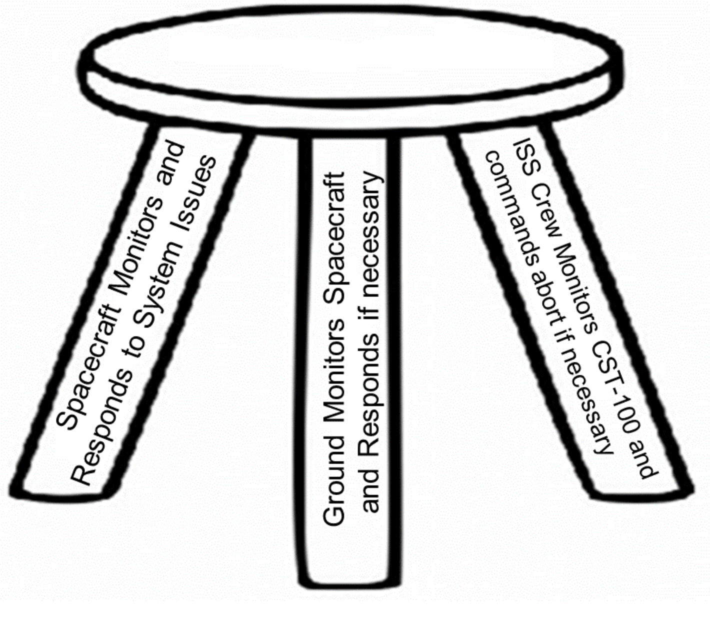
Spacecraft Monitors and Responds to System Issues
Ground Monitors Spacecraft and Responds if necessary
ISS Crew Monitors CST-100 and commands abort if necessary

Fig. 1. The OFT Three Legged Stool utilized the onboard flight software as the primary means of detecting and responding to hardware or software faults. Without a crew onboard, the ground team would have also monitored the spacecraft and commanded the spacecraft to take recovery actions if it failed to do so on its own. During rendezvous with ISS, the ISS crew would have also monitored telemetry from the CST-100 and would command an abort if the spacecraft violated predefined limits.

Generally, software performs stochastic or repeated operations on the vehicle such as activating a fan or cooling pump to run with a specific value of revolutions per minute. Most spacecraft utilize a process of FDIR to identify failures and automatically perform an operation to either recover the function or at least disable the defective part/function. For example, if the above fan or cooling pump experiences an overcurrent condition, the software will detect the threshold violation, safe the offending unit (usually by deactivating it) and then activating an alternate fan or pump to maintain cooling. Such FDIR is not part of the discussion in this paper. *Starliner* employs more sophisticated automation of operations by creating a set of sequencers that control every aspect of operations from launch until post landing vehicle power down. Two sequencers controlled transitional and pointing operations and a third managed all general operations. All sequencers operated independently of each other but were designed to work together (for example, each sequencer had its own commands to conduct a maneuver or burn in a coordinated fashion). During the highly dynamic phase of atmospheric ascent the spacecraft software generally is in a monitor mode only, conducting minor activities while the Atlas V controlled the launch vehicle and ascent trajectory.

Spacecraft actions were performed based on various trigger conditions such as range from the ISS, time since a previous action, altitude and so on. These sequencers executed the same commands that the ground control team could execute. In fact, it is the reliance on the Mission Elapsed Time (MET) trigger that caused the shortened OFT mission (cf. Section 6).

Mission Operations was involved in the design and testing of the software and sequencers. Using their many years of experience with crewed vehicles, the MO involvement was important in ensuring that the software not only met NASA and Boeing requirements but provided operability by the ground and the crew. Operability comprises two key areas: flexibility and usability. With over 60 years of experience, the FOD flight controllers have learned that flexibility is often critical to mission success. Often, situations arise that were not foreseen during the design or development of the spacecraft or mission. These could range from deficiencies in designs not meeting specifications or, as in the most famous case of *Apollo 13*, in reaction to an unanticipated failure. In the case of *Starliner*, NASA requirements were still evolving as the CST-100 Critical Design Review was completing.

Usability has several aspects. Clearly, the crew and ground interfaces must be simple to use as well as intuitive. In terms of automation, usability is of limited import if the interface is properly designed. An example of where the interplay becomes significant is how easy it can be for the ground to perform an operation. For *Starliner* some operations require exiting the automated sequencers, performing some action and resuming the self-directed, autonomous control. This complex transition between ground control and autonomous control, in turn, limits such actions to when the vehicle is in a benign state (e.g., coasting) as opposed to dynamic (e.g., performing a rendezvous burn).

A key characteristic of the sequencers is the ability to modify their behavior. At any time during the mission, an entirely new flight plan could be uplinked to the spacecraft, though this was not actually a practical option. The reason for this is that to make a change to the plan would require careful development including agreement among stakeholders the change was appropriate, followed by simulation and validation testing on a ground based software test rig to ensure its safe operation in both nominal and numerous potential off-nominal conditions. This process would normally take many months. While feasible during flight, in practice this was not considered anything other than an extreme contingency case due to the short (1 to 3 day) flight from launch to ISS docking. Instead, the ground relied on behavior modification capabilities. For example, the ground was able to selectively inhibit an instruction. This might be needed if the concept of operations changes or to work around a hardware or software deficiency found late in development.

Another operator command allows the ground the ability to force the autonomous sequence to perform an instruction and move on to the next event even if a logic condition is not met. This command was mainly a contingency case where the sequencer would otherwise hang up for some unexpected reason, such as if sensors failed to detect a condition the crew or ground knew was met. MO used this command in its toolbox to provide flexibility without changing software. For example, since the OFT was uncrewed NASA required the vehicle to perform certain demonstrations of safety measures (e.g., precise attitude control, station keeping, etc.) prior to entering the Keep Out Sphere (KOS) of the ISS [17]. The sequencer would force the *Starliner* to hold outside the KOS and when the ground teams were confident everything was working correctly the MO team would issue a command to advance the sequencer to allow the vehicle to proceed closer.

During development, the MO team created a number of canned operational sequences. These included planned operations such as Far-field rendezvous, proximity operations, and the ability to fly a 360-degree loop around the ISS for visual inspection. MO also created a number of contingency sequences such as rendezvous abort to make an emergency landing or stopping a rendezvous and going to a safe parking orbit, performing an unplanned translation maneuver, and so on. The software also had the ability to jump from one sequence into another, i.e., a software GoTo command. This allowed for quick transitions between the nominal timeline and, for instance, an emergency landing in case of a fire or an unexpected rapid depressurization. The ground, crew or automated software could make these jumps. When NASA-required demonstrations were added to the mission plan it was easiest to GoTo the appropriate OFT-only sequence rather than rewrite the nominal sequence. For example, the abort sequence demonstration would be performed at a convenient time in the far field phase of the mission, rather than trying to insert a condition such as a system failure near the ISS. At the appropriate time, the MO team commanded *Starliner* to "GoTo" the abort sequence and when the sequence was complete, command it to resume the nominal mission sequence.

A common design requirement for software is the ability to make changes. Of course, at any time the code itself can be changed but this is usually a time intensive, laborious process which involves careful testing and review. Normally such changes are made only if the software is completely not meeting a requirement. More adaptability is provided via a parameter file where thresholds, initialization values or even a list of commands can be updated in real-time. *Starliner* and ISS benefit significantly by this ability. Usually the process to validate these files is easier and faster to perform, though of course with flexibility comes complexity. While the change of a given value might be easy to make, for example changing a 0 to a 1, the process of validating the change might take a significant amount of time. The ISS can generally make these updates within 24-hours (a capability which is used very sparingly and cautiously) while the dynamic nature of the CST-100 mission required updates to be available in 4 hours in some situations. Besides having multiple stakeholders verify the right parameter was changed to the correct value, the modified software might need to run in a simulator for several hours to verify that it performs as expected in all possible usage cases, nominal and off-nominal.

Sometimes even this update flexibility was not fast enough for the operations team. For example, consider the communications link which, for *Starliner,* was primarily the NASA Tracking Data and Relay Satellite System (TDRSS). With many users the TDRSS requires a great deal of integration in scheduling specific satellites which typically takes several weeks. If a launch slips at the last minute or a mission timeline changes suddenly, either due to late priority change or to respond to a real-time contingency, adjusting the communications schedule can take a lot of effort. When the schedule changes the heavily utilized TDRSS time may not even be available at a critical time of need (e.g., docking) requiring negotiation with the NASA TDRSS network director. Satellite time might become available minutes before the needed time. Thus, the communication link schedule might need to be updated within hours or minutes of an event. Since the Mission Operations team performed the actual scheduling of the TDRSS it made sense to provide them the ability to build and uplink the required file. The communications file, in this case, was a file that informed the spacecraft of all TDRSS events that had been scheduled so that the spacecraft knew which of its antennas to activate at specific times to ensure a solid communication link. Therefore, tools that were certified to build a validated TDRS schedule file were developed and implemented in the operations environment in Mission Control instead of in the offline flight software engineering environment.

A key decision early in the *Starliner* program was to have the operations team "own" the key input to the sequencers, a flat file that is essentially the English readable form of what the sequencers would run, not unlike ground timelines used in previous programs. This Sequence Command Input File (SCIF) allowed the operators to shape the mission plan as development progressed. For example, about a year before flight, MO conducted simulations with the full flight control team using the draft file. These simulations allowed the flight controllers to see how the integrated hardware and software performed. In this manner early versions of Concepts of Operations (CONOPS), flight rules ([18]), and procedures could be table-topped and evaluated. As experience was gained, as well as mission requirements evolved, the input file was modified. This file is treated very similarly to the flight software. Therefore, its design and any changes must be carefully evaluated, impacted (e.g., there is cost to make an update) and tested or retested. This in turn limits the amount of flexibility that can be incorporated. Since the CST-100 program is a new one with a significant amount of discovery, there was a constant balance required to making updates as the team gained experience working with the automated system while preventing a perpetual series of updates and potential delays.

Significant evolution in the CONOPS for *Starliner* occurred in regards to calculating the orbital maneuvering burns. From the initial orbital insertion (OI) burn up through docking, *Starliner's* onboard software calculates a multiple sequence burn targeting plan that requires no interaction from the ground. For this automated vehicle, the ground was mainly supposed to support in the case of contingencies. Initially, there were limited options to adjust the burn plan. For example, if *Starliner’s* trajectory were predicted to intercept orbital debris an abort sequence would be commanded from the ground which took the vehicle out in front of but below ISS in a safe “parking orbit.” From that orbit, the ground can recommence the rendezvous targeting a docking the next day. As the ground team began training this technique and became more familiar with both the limitations and the flexibilities of the sequences, it was realized alternate approaches could be utilized.

By leveraging the integrated training environment, the ground team subsequently developed several CONOPS and procedures to perform different debris avoidance maneuvers. If the effect to the overall rendezvous was minor, one option was to just put the vehicle into a coasting mode until the debris conjunction was safely in the past. Automation would then be resumed and the onboard system would calculate an updated rendezvous trajectory and continue with the mission. Another option was for the ground to calculate a specific burn vector via Manual Delta Velocity (MDV) burn where the ground calculates a burn vector and uplinks the values to a special sequence. This option however, used more ground resources and leveraged heavily off the extensive experience of the Mission Operations team which by design was small (see [9]). Thus, through leveraging of the training environment that included flight software and vehicle system models coupled with a ground control team that collectively leveraged nearly 60 years of human spaceflight successes and failures, a diverse set of capabilities were established for a wide variety of potential uses.

An additional challenge was presented by the NASA-required demonstrations of the rendezvous sensors. Although Starliner’s sensors were based on the Vision-based Software for Track, Attitude, and Ranging (Vis-STAR) system developed for Orbital Express [6] NASA required careful validation prior to completely relying on them for a docking with the ISS. Redundant hardware on Starliner provides fault tolerance to system failure while robust software algorithm, with extensive testing and simulation, provides confidence in the overall safety of the system. Even then the crew provides a cross-check that the system was operating correctly. For the OFT mission, there was no crew to perform this vetting. Adding one-time only software changes were not required since tools on the ground could compensate. Instead, the flight controllers on the ground used raw measurements of where the spacecraft sensors measured the position of the target – in this case the ISS – and ran the data through independent navigation software to independently verify that the onboard system was correctly and safely heading to the ISS.

3. Training

Traditionally, NASA human spaceflight flight control team training encompassed individual flight control team positions learning about how the various systems, subsystems, and components within their assigned area of responsibility functioned (see [19]). These systems can range from life support to communications; guidance, navigation, and control; flight dynamics; power systems; mechanical systems; rendezvous operations; the flight plan; and so on. Training included the actual functionality as well as how the various piece parts integrated together, not just within that system, but with the spacecraft as a whole. The flight controller needs to understand how their portion of the spacecraft affects, influences, and achieves the various mission objectives. The knowledge needs to scope not only the planned mission but also any myriad of failure situations. The flight controllers are always learning and assessing from the perspective of crew safety, vehicle safety, and mission success – in that priority order. For human spaceflight missions, the flight controllers must also learn how the onboard crews interact with their systems, how they are affected by it, and how the crew can be utilized to help the system in various failure cases. Due to the development nature and complexity of every crewed space system, NASA flight controllers are typically deeply involved in discovering and solving operational and system engineering problems unanticipated by the hardware and software design teams.

One of the most useful activities for the operations team was to conduct simulations using the Training Simulation Integration Laboratory (TSIL). Ostensibly, the TSIL environment is designed to provide a flight like real time simulation to train the crew in the cockpit and ground operators in the Mission Control Center (MCC).Since the TSIL ran flight software in a mostly plug-and-play mode, on multiple sessions, with an easy to operate user interface, it also provided an excellent opportunity to stress the software in a fully integrated fashion as well as in smaller research sessions. Although many of the hardware systems in TSIL utilize low fidelity models, the facility provides invaluable opportunity to see how the system performed. While this opportunity for insight could be made about many integrated space systems, it is especially critical for highly automated crewed vehicles.

As automated spacecraft come into reality, flight controller training has had to change and adapt to account for and include this automation. In previous spacecraft, the ground team directly controlled nearly everything that the spacecraft was doing, either because the ground directly commanded functions to occur as is the case of the ISS or because the ground worked with the onboard crew to take specific actions (e.g., the Space Shuttle).

The ISS now uses an automation engine, known as Timeliner [20], on core systems in a simple implementation that monitors for specific failure conditions and takes specific actions. A more extensive use is in place on non-critical payload systems. Stepping up to the use of, and reliance on, this simple software set took a number of years with guarded step-by-step progression. For CST-100, the move to a fully automated spacecraft transpired quickly at the very outset of the design and the flight controllers had to learn not only what automation meant and looked like but then also how to augment their training to incorporate and ultimately exploit the automation to ensure crew safety, vehicle safety, and mission success.

By the time the OFT flew, the flight controllers had developed significant depth of knowledge of all the automated processes that the spacecraft could utilize. Each system discipline then moved on to understand how their system could be affected by the course of automation's sequencing, especially if the sequencing deviated from the original planning. In failure cases, the flight controllers must have solid knowledge and foresight of how a failure may cause the automation to stop functioning properly because it is waiting on a trigger condition from hardware that it not operating (and would thus continue to sit and wait unless told by the ground or onboard crew to continue).

As preparations for *Starliner's* Orbital Flight Test progressed, the flight controller training revealed that there was a new need in the mission control room. There needed to be a position that specifically was trained on the overall functionality and integration of the automation itself. This new assignment fell to the Flight Activities Officer (FAO) position, the position that traditionally built ground and crew timelines on previous missions. The FAO team built expertise and console tools to watch over the progression of the automation. They also developed the ability and knowledge to determine where the automation may have complications or stop working in the future due to failure conditions in the spacecraft. A key finding from much of this training was that in the face of contingency the operations team might have to inhibit steps in the sequencers and perform functions manually. This might occur because a failure might cause a trigger to get "stuck" or the automation could potentially power off the final unit in a redundant system.

A significant finding in the first years of training for *Starliner* operations was developing the techniques to integrate knowledge of the automation's functionality into the overall cadence of operating the mission. This training also led the flight control and training teams developing the ability to stress, poke, and prod on the automation in order to find both its strengths and weaknesses. Weaknesses were passed back to the engineering teams to determine if improvements or changes should be made. Strengths were documented and leveraged so that the flight controllers could develop techniques to utilize those strengths in a variety of ways. These additional operational uses were not initially planned when the software was developed but showed (in training and during the OFT mission itself) to be extremely useful in helping the ground teams work through mission problems not initially foreseen.

This training merged well with MO's role in software development mentioned earlier. Since MO owned the SCIF there was a great deal of synergy between training and its development. Often the operations team would take lessons learned from a simulation and make significant modifications to the SCIF. However, every update had to be carefully reviewed, its impacts to the flight software development schedule evaluated and testing plan updated.

While this approach was a significant advantage for ground and crew training, software development, and actual mission execution, a disadvantage was that MO's role in performing these tasks was not clearly understood or anticipated in the early stages of development of the *Starliner* program when work was scoped and manpower and other resources were allocated. Thus MO was continually assessing the relative priorities of SCIF or other software changes (and the associated testing and verification) against the cost in time, personnel, and other resources that were already targeted to be used for other aspects of the program. See also [9].

Although the focus was primarily on the upcoming OFT mission, training simulations with the CFT crew were also conducted in parallel. Besides providing some early training to the crew, this provided two significant benefits: allowing lessons learned from crew interaction to be folded into the SCIF early on and it ensured the OFT mission reflected a planned crewed timeline as much as possible.

Additionally, as a completely software based simulator, trades are made by simulator developers on the level of complexity to build into the simulation models of hardware components. Software running in a power controller, for example, can be emulated in a relatively straightforward manner. Software running in an Inertial Measurement Unit, however, may approach that of the vehicle flight computer itself in complexity. Being able to properly explore interactions between automation on the flight computers and the hardware the automation is controlling often depends on the fidelity of the hardware simulations.

This then presents two challenges the training teams must always consider. One is that the software, which includes the various automation sequences and files, in the simulator may be lagging the software planned for use in flight and the teams must ensure they understand the differences between simulator and flight software. The second is that configuration management of the various training loads (which include the flight software as well as all the various model parameters) must be maintained so that the training teams, flight control teams, and crews always understand how closely their training configuration matches what will actually fly on a given mission.

A critical lesson learned during training was that automation does not always behave as expected, especially in the face of failures. As mentioned above, the ground operators were involved in the design of the software from nearly the beginning of the program. With MO having extensive, recent experience with the Space Shuttle and the ISS, sequences were developed based on lessons learned in these programs. Of course, there are significant differences between these vehicles and *Starliner*. For example, the Space Shuttle could take as many as 3 days to land after undocking from the ISS to touchdown, enjoyed numerous landing sites around the globe and with its wing structure had a significant amount of cross range allowing "steerability" from many orbits to a landing opportunity offset over a thousand kilometers from its orbit ground track. On the other hand, *Starliner* touches down only 4 to 5 hours after undocking, has a much smaller entry cross range, and has only a few landing sites concentrated in the Southwest continental United States. In one simulation of the undocking to landing phase, due to a failure, one of the steps in the main sequencer was inhibited since it would not occur correctly. Unrealized at the time, the step inhibited was tied to a trigger tied to a specific event, with subsequent commands set to follow as soon as that (now inhibited trigger) was satisfied. Therefore, when the desired step was inhibited, the subsequent commands executed right away, unexpectedly, until the sequencer hit the next unmet trigger condition because the trigger that normally forced the automation to wait had been inhibited. This forced the flight control team to manually back out of some entry configuration that had been entered prematurely.

5. Execution – the Orbital Flight Test

Starliner's uncrewed Orbital Flight Test lifted off from the Kennedy Space Center on 20 December 2019 at 11:36 UTC. In general, the automation worked as expected; however, a flight software error in the calculation of the MET caused a series of events to occur that forced the operations team to make some real time changes. While most of the mission objectives were achieved, a key one – docking with the ISS – could not be performed.

The MET clock begins counting up from zero at liftoff, maintaining a running time for the entire mission. Normally, it is just another way to track time during the mission when other measures such as UTC become cumbersome. In the case of *Starliner* the only place MET is directly used during the mission is to perform the Orbital Insertion (OI) burn. A significant part of the spacecraft's mass is the Launch Abort System and the propellant it uses to remove the Crew Module from the rocket in the case of catastrophic failures. Once beyond the need for an ascent abort and after separation from the Atlas V, the Orbital Maneuvering and Abort Control (OMAC) engines are used to perform the OI burn, putting the *Starliner* into a sustainable orbit.

To simplify the trajectory calculations the OI burn was designed to occur at a fixed MET. Multiple events in different sequencers are designed to perform orbital insertion. First, the guidance software must calculate where in the orbit the Atlas V has deposited the spacecraft (which for the OFT was exactly as targeted) and what translational burn maneuver is needed to put the *Starliner* into orbit. The burn calculation for the OI maneuver also sets up all the subsequent burns all the way to docking. The translational and attitude pointing sequencers ensure that the spacecraft is oriented in the correct thrust vector alignment mode.

After launch vehicle separation, the controlling sequencer configured for orbital operations and set the OI time of ignition to the planned time. Unrealized at the time to the Boeing team, a software error actually initialized the MET clock not at liftoff but approximately 11 hours prior to launch. Thus the burn was calculated to be in the past, as was the maneuver to thrust alignment.

The ground team executed a contingency plan uplinking the MDV command designating alternate burn target parameters.. Performing a manual burn that still remains within the capabilities of the onboard automation requires several commands from the ground to get the various sequencers synchronized. Due to complications with commanding caused by intermittent satellite communication with *Starliner caused by various sources of interference*, the MO team was unable to completely synchronize the translation and pointing sequencers before the time of ignition of the contingency MDV sequence. Therefore, the commanded burn initially started off performing a multi-axis thruster burn while maintaining an attitude keeping its solar arrays pointed at the sun. The ground team's attempt to improve the *Starliner's* burn orientation to better align the intended thrust direction with the *Starliner's* OMAC engines was not completely successful due to the difficulty of getting the commands onboard at the right time. Both OMAC and reaction control system engines continued to fire in a non-optimal vehicle attitude, expending extra propellant that was normally allocated to dock with the ISS. Once the vehicle had achieved a safe orbital altitude, the ground team intervened for a final time to disengage the manual maneuver and prevent further inefficient consumption of the remaining propellant. Approximately one orbital revolution later, the ground commanded another burn, successfully raising the *Starliner's* perigee by a small amount to prove that the *Starliner's* translational maneuver capability had been fully recovered. See Fig. 2.

Fig 2: (Left) From Left to right flight controllers Joe Jones, Ramon Gonzalez and Carson Sparks discussing how to perform the OI burn. (Right) Flight Directors Mike Lammers (left) and Richard Jones (right) look out over the control room as the team uplinked the MDV maneuver.

At this point *Starliner* was in a stable orbit. However, the higher than expected propellant made getting to the ISS and back to the ground just barely out of reach and the rendezvous was terminated. The mission was replanned and two orbit adjustment burns were performed to target a landing approximately 49 hours after liftoff. While in orbit number of operations were performed using the full autonomy of the spacecraft and all performed flawlessly.

On December 22, 2019, the Mission Operations team engaged the entry automation sequence to bring *Starliner* home during its 33rd orbit of the Earth. *Starliner* proceeded to target and execute a fully autonomous deorbit burn and atmospheric re-entry and parachute landing to White Sands Space Harbor in New Mexico. The only human intervention performed was a planned operator command to separate the parachutes following landing (a function normally performed by the crew when onboard).

6. Summary

Although automated spacecraft operations allow safe operations for crew, it is critical that flexibility and appropriate interfaces with the ground team be incorporated in the design from the start.

In the case of the *Starliner*, which is designed to ferry crews between Earth and the International Space Station, reliability is crucial to ensure the safety of the crew under extreme conditions including returning deconditioned astronauts who have been in space for 6-months or more. In addition, there is significant risk to both *Starliner* and the ISS (the ISS crew and the multibillion-dollar advanced laboratory) should a collision between the two vehicles occur. The risks associated with ensuring crew and vehicle safety mandate a robust verification and validation process of the *Starliner*, including its automated software.

A key realization when determining the extent to automate a spacecraft is to discern when it is most beneficial to use the full capabilities of the automation software and when it is best to bypass the software and rely on the human operator (either the onboard crew or ground-based control team). A key factor in this decision-making process includes time criticality of actions as well as the impact of responses. Computing and calculating an orbital adjustment burn requires a quick calculation, lending itself to computer automation to perform the task. However, an incorrect burn computation could have catastrophic repercussions if it changes the trajectory to put it on an intercept path to the ISS. Further adding to the consideration is the extent to which flight crew must have the ability to seamlessly manually intervene on actions the automation is taking. The incorrect orbit insertion and subsequent flight control override to achieve orbit provided a reminder that, especially in a first flight situation, where human intervention can be critical.

Although rendezvous was not performed on the OFT the development of the ground tools to independently verify the navigation system shows that in certain circumstances a robust ground control team is still required with an automated vehicle under nominal situations. Otherwise, further software or hardware may be required in critical situations like an uncrewed vehicle docking with the ISS.

Although the clock issue on the OI burn for OFT was a relatively straight forward software error its impact on the automated system was significant. Each sequencer performed as expected given the erroneous software inputs but the integrated outcome of the sequencers being out of sync had not anticipated an error such as the burn time being in the past. Sync points or cross checks, used elsewhere in the software, were not in place here but will be in place for the next flight.

One goal of advanced spacecraft automation is the ability to reduce both the crew workload and the ground control footprint. Development of *Starliner*, to include its first test flight, has shown that significant progress has been made in this area. As flights continue, with the Crewed Flight Test and then regular crewed missions to the ISS, continued improvements and enhancements will occur as more experience is gained.

For missions that go beyond the Moon, software that autonomously controls nearly every aspect of a crewed mission will become a necessity given the long time delays between the spacecraft and Earth's ground control teams. The lessons learned by Boeing and its Mission Operations team, through the design and implementation of *Starliner*'s hardware and software automation, will be able to continue to inform future public and private spacecraft design. As the technologies and capabilities evolve, incorporating lessons learned in successful low Earth orbit commercial crew vehicle missions, spacecraft designs will continue to improve and be able to better enable safe execution of human missions to the Moon and beyond. The fully autonomous spacecraft depicted in science fiction is not quite here yet, but swift progress is certainly being made towards it.

Acknowledgements

The authors wish to thank Chris Ferguson, Steve Gauvain, Pooja Jesrani, Ray Bigonesse for their help in creating this paper.

References

[1] A. S. Fukunaga, G. Rabideau, S. Chien, David Yan, ASPEN: A Framework for Automated Planning and Scheduling of Spacecraft Control and Operations, In Proc. Int. Symposium on AI, Robotics and Automation in Space (i-SAIRAS), Tokyo, Japan, 1997

[2] P. Ferri, E. M. Sørensen "Automated mission operations for Rosetta." *Proceeding of the Fifth International Symposium on Space Mission Operations and Ground Data System: SpaceOps*. v98. (1998) doi=10.1.1.547.2575&rep=rep1&type=pdf

[3] J. Straub, et al. "Application of collaborative autonomous control and the open prototype for educational NanoSats framework to enable orbital capabilities for developing nations." *Proceedings of the 64th International Astronautical Congress*. 2013.

[4] R. Murtazin, N. Petrov, Acta Astronautica, v77 (2012) 77-82

[5] P. Z. Schulte, D. Spencer, Acta Astronautica v118 (2016) 168–186

[6] M. R. Leinz, C. T. Chen, and M. W. Beaven. "“Orbital express autonomous rendezvous and capture sensor system (ARCSS) flight test results,” SPIE Defense and Security Symposium." *International Society for Optics and Photonics* 6958.8 (2008)

[7] Tom Mulder "Orbital express autonomous rendezvous and capture flight operations, Part 2 of 2: AR&C exercise 4, 5, and end-of-life." *AIAA/AAS Astrodynamics Specialist Conference* and Exhibit. 2008

[8] T. A. Mulder, "Orbital express autonomous rendezvous and capture flight operatio AR&C exercise 4, 5, and end-of-life." *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. 2008.

[9] R. C. Dempsey, E. Van Cise, M. Lammers, R. Jones. Operating a Crewed Spacecraft in the Age of Commercial Space Using Private/Government Partnership: SpaceOps-2020,16,1,3,x268

[10] CCT-REQ-1130, NASA ISS Crew Transportation and Services Requirements Document

[11] B. G. Kanki, R. L. Helmreich, J. M. Anca. Science Direct. Amsterdam: Academic Press/Elsevier, 2010. Internet resource. <http://www.sciencedirect.com/science/book/9780123749468>.

[12] D. Rogers, "NASA's Space Flight Resource Management Program: A Successful Human Performance Error Management Program." AIAA SpaceOps 2002 Conference. Houston, Texas: AIAA, 9-12 October 2002. <http://arc.aiaa.org/doi/pdf/10.2514/6.2002-T4-12>.

[13] W. O'Keefe, "Space Flight Resource Management Training for International Space Station Flight Controllers." AIAA SPACE 2008 Conference & Exposition. San Diego, California: AIAA, 9-11 September 2008.

[14] E. Baldwin "Integrating Space Flight Resource Management Skills Into Technical Lessons for International Space Station Flight Controller Training." Proceedings of the 3rd Annual Conference of the International Association for the Advancement of Space Safety. Rome, 2008.

[15] SSP 50021, "Safety Requirements Document"

[16] M. Holguin, G. Herbella, R. Mingee. "Commercial Crew Launch Emergency Detection System The Key Technology for Human Rating EELV." Proceedings of the AIAA SPACE 2010 Conference & Exposition. Anaheim, California, 2010. AIAA 2010-8670 (doi https://arc.aiaa.org/doi/pdf/10.2514/6.2010-8670)

[17] D. S. Koons, Craig Schreiber. "Risk Mitigation Approach to Commercial Resupply to the International Space Station." (2010)

[18] A. Herd and R. Dempsey, Flight Rules: Purpose and Use, 2013, in Safety Design for Space Operations, Editor in chief: Tommaso Sgobba, Editors: Firooz A. Allahdadi, Isabelle Rongier, Paul D. Wilde; The International Association for the Advancement of Space Safety, Butterworth-Heinemann

[19] "The International Space Station - Operating an Outpost in the New Frontier", 2018, Gov. Printing Office, Edited by Dempsey, Robert (https://go.usa.gov/xQbvH)

[20] R. Brown, E. Braunstein, R. Brunet, R. Grace, T. Vu, D. Zimpfer, W. Dwyer, "Timeliner: Automating Procedures on the ISS", 2002 (https://ntrs.nasa.gov/search.jsp?R=20100036766)

https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100036766.pdf