SSC24-S1-05

Planet's Agile Software Development for Spacecraft

Kenneth Donahue, Kiruthika Devaraj, James Mason, Meric Ozturk
Planet Labs, PBC
645 Harrison Street, 4th Floor, San Francisco, CA 94107, USA
kenny at planet dot com

ABSTRACT

Planet Labs PBC (NYSE:PL), is a vertically integrated space and data company, and a leading provider of daily Earth data and insights; we design, build, and operate the largest constellation of imaging satellites in history. We have built and launched over 500 satellites by pioneering the agile aerospace philosophy that encourages rapid prototyping and iteration on the hardware, and building a minimum viable flight software to launch the satellites into space. This allows us to use space as a testbed and extension of the development environment as quickly as possible. Further software development occurs after the satellites are in space, and new features are progressively brought online to customers as the software matures. Planet's 3U Dove satellites were built with this approach in mind and over the last 10 years there have been 15 hardware iterations and ten times as many software iterations.

Pelican, the next-generation follow-on to SkySat, is a category-defining satellite constellation that is designed to deliver responsive, rapid, very-high-resolution data to our customers. This new constellation has been developed using this agile development process. We have focused on rapidly creating and integrating a prototype, and then evolving it with software updates over time to deliver satellites with relatively short design cycles. Prior to delivery for launch, the focus is on developing and validating a **minimum viable product** (**MVP**) with a base level of capability that enables reliable communication, security, and over-the-air reprogrammability. All low-level interfaces and devices are confirmed working but await a later software update to fully enable those features.

Planet launched our first successful Pelican Tech Demo (Pelican-1) on Transporter 9 in November. Within the first orbit, we made first contact and got critical telemetry. Since then, we have successfully commissioned and enabled multiple subsystems via more than 70 software updates (many of them in the first few months). In this paper we discuss the core software features that must be tested early in the program, what features can be delayed until later, and how we triage various needs to safely and incrementally improve the spacecraft post-launch.

Aerospace Development Methodologies

Historically, aerospace programs have been large, long-lived, and expensive. A traditional aerospace development flow follows the **Waterfall** project development methodology, in part because it was the military standard for contractor development[1]. Waterfall is typically used on programs that have long timelines with a priori known deadlines[2]. It generally involves a linear flow from initial concept through delivery with firm handoffs between teams as the program progresses in maturity.

NASA generally follows the Waterfall model because it is instilled in the NASA Procedural Requirements[3], which means there are many existence proofs of the Waterfall methodology successfully delivering world-class spacecraft.

Embracing Agile Aerospace

The Waterfall methodology works well in many cases. Using NASA as an example, it is clear that Waterfall works for large-budget, low-quantity spacecraft that need extremely high mission assurance and require coordination of many subcontractors (e.g. the James

Webb Space Telescope). One of the challenges with implementing programs like this is the serial nature of development. Software cannot be sufficiently matured and validated until hardware is available to develop/test against, and launch cannot happen until software is sufficiently mature. These dependencies extend schedule, increase program cost, and necessitate the addition of complex specification and interface management processes or emulators. Agile Aerospace decouples software development, leading to faster and lower cost/complexity programs. **Figure 1** depicts the notional development flows used in the traditional Waterfall and Agile Aerospace methodologies.

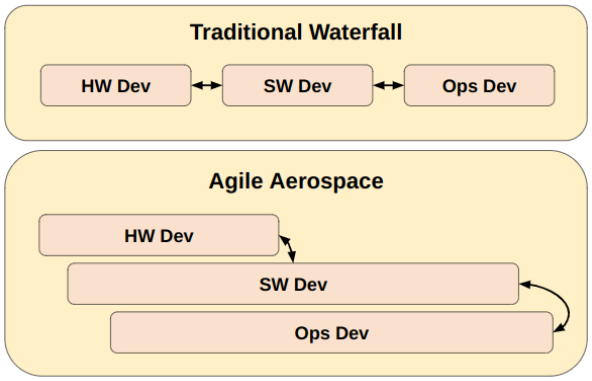

Figure 1: Development Methodologies

Planet on the other hand works on optimizing cost, schedule, and capabilities against a risk profile for a given mission. With a distributed network of satellites as in the Dove and SkySat constellations operated by Planet, it is acceptable to take risk at the individual component, system, or satellite level while ensuring mission success by working on redundancy at the component, system, or even satellite level. Reframing the success criteria in this fashion allows Planet to take a completely different approach to satellite development.

Planet has created a development methodology we refer to as **Agile Aerospace**[4]. **Agile** is a software development methodology based on iterative development using the following principles[5]:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile Aerospace is a spacecraft development methodology that borrows principles from the Agile software methodology and, with appropriate modifications, applies them to hardware components and mission design as well. The goal is to deliver a minimally working spacecraft prototype that enables all necessary hardware checkouts, exercise that platform in space, and then apply any lessons-learned as design updates in subsequent spacecraft. This feedback loop allows Planet to take risks with early spacecraft and use them to derisk future spacecraft.

The major problem with employing Agile Aerospace to a space program is the inherent risks with the first prototypes or tech demonstration satellites. These tech demo satellites may not meet all the performance requirements and may need more complex and involved operational support as new features are brought online with software updates over time. Additionally, the fact that the hardware may evolve from spacecraft to spacecraft implies that both the onboard-software and the operations stack need to be able to support heterogeneous feature sets as hardware configurations evolve through this iteration process.

However, the benefits of adopting Agile Aerospace at Planet are very clear: Planet has proliferated a constellation of 100s of cubesats and dozens of SkySats. With each iteration of these satellites, new features are added in both the hardware and software, thereby enabling Planet to integrate and leverage the latest technology innovations from adjacent sectors (consumer electronics, automotive, medical devices, etc.). These upgrades enable new features such as e.g. more spectral bands or faster image delivery to our customers. On the Smallsat program, and Pelican-1 in particular, Planet was able to deliver a working spacecraft prototype in just 2 years from original conception.

Background on the Smallsat Program and Pelican-1

Planet has taken on the task of creating a new spacecraft platform using state-of-the-art standards, cutting-edge processors, and blazing-fast protocols[6]. The goal of the Smallsat program is to create a multi-mission spacecraft bus that can accommodate a large family of payloads. The Pelican constellation[7] (30-cm class optical payload) and the Tanager constellation[8] (a VSWIR imaging spectrometer) will both be using the Smallsat platform[9]; Pelican-1, which was launched in Nov 2023, is our first technology demonstration using the Smallsat platform and is intended to be the first of many satellites.

The Smallsat platform has the following basic architecture as shown in **Figure 2**.

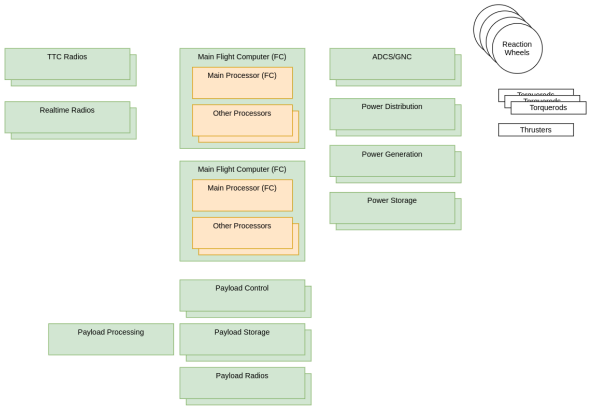

Figure 2: Smallsat High-level Architecture

Below is a list of features relevant to the agile process workflow employed on Pelican-1, which will be discussed in subsequent sections of this paper:

- Telemetry, Tracking, and Command (TTC) radios: the primary communication path between terrestrial ground stations and the spacecraft.
- Multiple heterogeneous processors: Each responsible for a specific function; there are hot or cold spares of nearly everything in the system.
- Power system: deployable solar arrays for power generation, batteries for power storage, and switches and conditioning for power distribution.
- Attitude Determination and Control system (ADCS) and Guidance Navigation and Control (GNC) system: The control loops using sensors (magnetometers, sun sensors, IMUs, gyros, GPS, and star trackers) to determine how to command actuators (magnetorquers, reaction wheels, and thrusters) to elicit the desired pointing mode of the spacecraft (e.g. point the solar arrays at the sun).
- Payload storage and processing: High speed special purpose processors and high capacity storage devices with high speed interfaces to the payload and radio associated with storing, processing, and downlinking payload data.
- Flight computer: the logical control hub of the system.

A few additional points of note in the architecture that are interesting from a software perspective.

- The software update processes should be tolerant to failures and interrupts and always be recoverable. As a result, each processor has multiple **software image slots** (locations from which software images can be loaded) and some even have multiple types of memory (each with multiple software image slots).
- There is a fair amount of cross-strapping between subsystems to increase reliability and flexibility, which means the number of possible configurations in which the spacecraft may be operating is very high.

Processes Supporting Agile Aerospace at Planet

Planet applied the Agile Aerospace methodology on the Smallsat program. The program started with requirements that were of varying completeness and certainty depending on the subsystem. The requirements were useful as an initial starting point, but refining the requirements became less important than strongly collaborating with stakeholders, prototyping and rapidly delivering piecemeal functionality as soon as it was available, and constantly iterating with stakeholders to make sure their needs were being met. This mindset has continued throughout the lifecycle of the program.

The main pieces of Agile that Planet uses are **scrum** and **feature-driven development**. The scrum scopes and prioritizes the backlog of work, and every two weeks (this two week period is called a **sprint**), our engineering team signs up to complete a certain volume of work. Sometimes we estimate that volume well and we pull in some of the backlog. Sometimes we estimate poorly and some of the volume carries into the next two week segment. Feature-driven development means we meet with stakeholders very regularly (at least weekly) and discuss the following:

- points of friction (what do they want to see changed)
- user stories to inform feature development (how do they want the system to behave)
- prioritization of known feature requests and bug reports
- milestone status, confirming the cognizant engineers driving the milestones are properly supported and the milestones are on-track (or discussing why they are not and how we rescope/replan)
- the current state of software rollouts (from release candidates through flight-like integrated tests on ground-based testbeds, to on-orbit checkouts and eventual on-orbit rollout to all images). (This software rollout process is described in detail later in this paper.)

These weekly meetings are an invaluable forum for transparency and flowing information across teams,

ensuring we are generally focused on the same things and moving in the same direction at the same time. However, they are not the only modality for communicating amongst the teams. Our teams are in constant communication with anyone they need to be via the company-embraced instant messaging communication platform and video conferencing tools. These tools allow for asynchronous communication when that is sufficient, or more structured meetings when needed.

Weekly reprioritization means that the plan can change very quickly. Whatever problem is most impactful to the overall program can become the focus of the entire team if necessary. More likely, the reprioritization causes some tasks to change, usually at the expense of knowingly pushing some work into a subsequent sprint (either just deferring work or knowingly accepting some **tech debt** (a known incomplete but passable solution to be replaced with a more complete solution as future work) in favor of transitioning to the new higher priority work). This interrupt-driven reprioritization is embraced as part of the nominal workflow, especially as we encounter new issues at the system integration stages that may not have shown up at the subsystem level stage, or occasionally when the space environment presents interesting anomalies that need triage. With the knowledge that the MVP for launch (where **launch-MVP** is the incompressible feature set necessary to keep the spacecraft safe and is required by launch) contains (by definition) all necessary features to keep the spacecraft safe, we generally have time to properly triage anomalies and insert them into our backlog with the proper priority.

It is worth noting that the MVP and its associated milestones are defined and agreed upon by the entire team with input provided from Systems Engineering, Program, Missions, Operations, Flight Software, and any other relevant stakeholders. These milestones allow us to reason about different features.

- Which features are dependent on one another?
- Which developers need additional support if milestone development needs to be parallelized?
- How might a delayed milestone impact the overall schedule?

While these milestones are defined, there is a constant negotiation between the various teams to ensure the definition of MVP for that milestone is properly scoped. If a milestone is vague or large in scope, it is decomposed into several milestones. If we realize that some dependency for a milestone is not going to be ready in time, we replan, reshape, or redefine the milestone and its associated MVP with whatever contingencies we agree upon. All this is to say that while we have a milestone and MVP roadmap, the milestones and MVP constantly evolve as we gain more understanding about the system. By ensuring there is alignment across the entire team about the MVP feature set for a given milestone, we are able to avoid scope creep which could result in massive delays to the program. Any additional scope gets factored into a planned future milestone and goes in the backlog.

Minimum Viable Product for Launch on the Smallsat Program

The complexity of the system coupled with a rigid launch schedule means that we as a team need to focus on and agree upon an incompressible feature set: the minimum viable product for launch (**launch-MVP**). Leading up to launch, we would regularly revisit this feature set to ensure we truly had eliminated everything that could be deferred until after launch. The team focused on delivering a known-working platform that had been electrically tested. The known-working platform could then accept a future software update that would unlock new (but known, as-designed) capabilities of the platform.

To that end, the focus prior to launch for a new platform is on the following features:

- bidirectional, reliable radio communications
- security;
- **over-the-air** (**OTA**) reprogrammability of everything that is reasonable;
- a gating mechanism to protect deployables from early deployment;
- functioning power system (generation, distribution, and storage);
- minimal attitude control necessary to maintain spacecraft safety while in our post-launch-vehicle-separation configuration;
- minimal thermal control necessary to maintain spacecraft safety while in our post-launch-vehicle-separation configuration;
- minimal fault protection necessary to maintain spacecraft safety while in our post-launch vehicle-separation configuration; and
- minimal propulsion integration.

It is worth noting that mission requirements and licensing conditions mandate that our smallsats be capable of performing collision avoidance; enabling, commissioning, and testing propulsion quickly after launch was important. The system was extensively tested on the ground, but full operationalization of the system came after on-orbit checkout and

characterization as part of Pelican-1 commissioning activities.

While that features set was required before launch, most of these features were also required months prior to launch as part of spacecraft integration: once the subsystems are integrated into a full spacecraft configuration, these features are necessary to support various tests in our verification and validation campaign.

The reader may notice some large exclusions from this feature set:

- the payload;
- fully-featured radios;
- time distribution;
- scheduling;
- (more) comprehensive fault protection;
- fully-configurable software; and
- star trackers

These exclusions are intentional for the launch-MVP. Subsequent software updates can and have unlocked these other features.

Radios

In general, space missions need to establish communication with a newly launched spacecraft shortly after launch to assess spacecraft health and, if necessary, command the spacecraft into different configurations to maintain health or perform a software update.

On the Smallsat platform, we have many radios; the only radios truly required for launch-MVP are the TTC radios. Further, the only required mode of operation for the TTC radios is the slowest symbol rate; this rate is sufficient for us to communicate, ascertain health state, and if necessary, command the spacecraft into a new state. This mode has the benefit of having the least stringent requirement on attitude pointing, which in our case, removes any attitude pointing requirements.

Lack of attitude pointing requirement means that even if the spacecraft is tumbling, we are able to communicate, assess the health of the spacecraft, command, and perform any necessary updates.

Security

As a provider of trusted imagery to businesses and governments, lack of proper security could cause a loss of mission. As such, and per federal mandate, security on the communications link between mission control and the spacecraft is a hard requirement: each packet requires encryption and authentication. Key management was also deemed sufficiently important for testing and interaction with the security system that it too became a hard requirement.

More extensive nice-to-have features were not required for the launch-MVP and are instead implemented as software updates.

OTA Updates

Software update capability is a must, but not necessarily everywhere. We chose to exclude some vendor-provided hardware from our initial set of update-able targets. The expectation was that these parts of the system, having heritage from other programs, would be much less likely to need a software update and if the need were to arise, we would be able to perform an OTA update to whatever parent subsystem communicates with the vendor hardware and unlock reprogramming to the vendor part on an as-needed basis.

Given that posture, there was a strong requirement that all non-vendor provided hardware be update-able. This implied work in several places, but included:

- bulletproof bootloaders
- easy software image slot selection
- full transparency of what software image slots are selected and running
- the ability to interrogate the system to ensure validity of software image slots
- demonstrable recovery from any individual image update failure due to e.g. early termination, fault recovery interruption, power interruption, SEU, or any other anomaly.

Deployables

Early deployment of the solar arrays (e.g. while integrated with the launch vehicle) could have resulted in damaged solar panels and a severely degraded mission. As such, great care was taken when testing and delivering the features for deployable release.

Power

While other parts of the system were intentionally pared down for launch, the power system was mostly feature complete. This system has to work reliably always.

Attitude Control

Minimal attitude control in the post-launch-vehicle-separation configuration required the ability to force a slight rotisserie roll on the spacecraft. This rotisserie would ensure the sun would hit at least some of the solar panels; this would ensure a safe, power-positive state for the spacecraft. The

rotisserie would also change the relative angles between our TTC radios and ground stations antennas ensuring that we would have a high chance of successfully communicating with the spacecraft in an early contact. Mission Control established communication with Pelican-1 on the first contact and proved this initial operational mode sound.

Early operations had further requirements that once we deploy the solar panels, we be able to point those panels at the sun; this was coupled with a strong desire to deploy the solar arrays early in the mission to prove out the power generation design. Some of the features related to these activities were included in launch-MVP; others were deferred until after launch.

The combination of all these requirements meant most of the attitude determination sensors and most of the attitude control hardware had to be fully feature-complete and plumbed into the attitude control loop. The necessary control modes were tested in software-in-the-loop, processor-in-the-loop, and hardware-in-the-loop configurations to gain confidence in their viability.

All other (more involved) pointing modes were deferred until after launch. Many of them required more hardware to be operationally supported (star cameras) or timing precision, none of which were provided in the launch-MVP feature set.

Thermal Control

The initial thermal control implementation hardcoded heater configurations and thermostat setpoints to known survival temperatures. While this was fine for the first spacecraft, each spacecraft is different and may require tweaking and each payload has its own thermal requirements. This was an area of accepted tech debt; moving these hardcoded configurations to configurable but non-volatile storage was an effort taken on well after launch-MVP delivery.

Fault Protection

Fault protection is an unending set of work. There can always be more and more conditions to check for and more and more responses to those conditions.

The set we decided upon was a very limited set agreed upon by thermal, power, missions, operations, and flight software. Originally, the recovery for every error was to move to a progressively "safer" system configuration until only the bare minimum set of hardware was powered.

Conditions we looked for that would elicit the safing behavior included:

- a small set of thermal out-of-bound conditions
- a small set of voltage out-of-bound conditions
- a small set of unresponsive hardware conditions

Fault protection has been a constantly evolving part of the system and has become more and more feature-complete as we have matured the program and learned from our experience with on-orbit operations.

Software Rollouts

While the focus prior to launch is on launch-MVP and its associated milestones, after the satellite is shipped to the launch base, the focus shifts to the "next MVP" and associated milestones. Milestone work is well-scoped and generally we have many different groups of people working on different milestones in parallel. A milestone may touch a single application, or may touch many applications in the system. Whatever the affected software set of that milestone is, as a milestone reaches completion, we begin testing and accepting software candidates for each piece of the puzzle.

Software Rollout Process

The software rollout process described by Doan, et al[10] is detailed as follows:

1. A software release candidate is loaded into a flight-representative hardware-in-the-loop test environment that mirrors the current nominal flight configuration. This allows us to have confidence that any testing we do in this environment should mirror what we will see on orbit when we deploy to the real spacecraft.
2. The candidate undergoes the nominal software update procedure. This gives us confidence that the upgrade path from the current application to the new application does not make any breaking assumptions and that the upgrade path is safe.
3. The candidate is then used in a set of nominal/off-nominal automated procedures that provide confidence that we have not introduced a regression and that the desired feature is available to unlock the feature from the milestone.
4. The candidate is then cleared for load to flight assets.
5. The release is then loaded onto a single image of a spacecraft. It is again run through a set of nominal/off-nominal automated procedures.
6. The release is then allowed to run for several orbits of use.

7. The release is then greenlit to be installed in all relevant software image slots on the spacecraft.
8. The release is then greenlit to be installed on more spacecraft.

If at any point the candidate behavior is not desired, we stop the rollout process, assess the issue, and decide how to move forward. This process makes it difficult to roll out a breaking change to an entire spacecraft, let alone an entire constellation.

Software Rollouts To-Date

The Agile Aerospace approach to spacecraft delivery means that we should expect to see many software updates early in the program, but that the rate of software updates should reach a steady state once all features are implemented and the system reaches maturity. This steady state reflects resolutions for novel anomalies encountered in the space environment as well as software changes to support different product offerings and/or changes to the concept of operations.

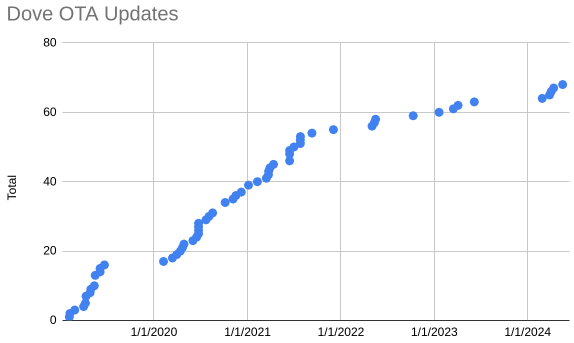

Figure 3: Dove Software Updates

Figure 3 shows software updates on the Dove program over the past few years. As expected, we see an initial spike of software updates followed by two distinct periods of relatively stable software updates. The initial spike reflects a period of rapid iteration and application of lessons-learned. The next period from 2020 to mid-2021 of about 20 updates/year reflects a period of a priori known platform improvements. The last period from mid-2021 on of about 5 updates/year reflects a period of continued optimization.

The Smallsat platform has an order of magnitude more compute nodes than the Dove platform, so while there are general similarities between the programs' update cadence, the Smallsat program has been updating software much more frequently.

Since the launch of Pelican-1 in November 2023, we have learned a lot about the Smallsat platform and have also unlocked a lot of features. Here are some notable statistics on the software updates we have performed on Pelican-1:

- 0 bootloader updates
- 1 kernel update
- 1 device tree update
- 71 unique updates (each applied to multiple slots)
- 28 unique update targets

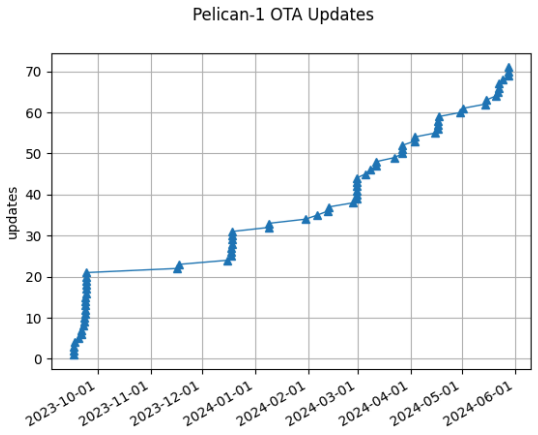

Figure 4: Pelican-1 Software Updates

Some of the application updates double-dip into one another (e.g. we may only have a single release to orbit that covers both a thermal change and a power system performance change). Here is a mapping of most of those 71 updates to different subsystems:

- 8 updates to our TTC radios
- 7 updates related to attitude determination and control
- 6 updates related to thermal/heaters
- 6 updates to fault protection and recovery
- 5 updates to the power system
- 5 updates to routing
- 5 updates to telemetry and event handling
- 4 updates to security
- 4 updates to OTA updating
- 4 updates related to propulsion
- 4 updates to our payload
- 4 changes to scheduling/onboard sequence execution
- 3 changes to time knowledge and distribution
- 2 updates to enable onboard scripting and better debugging

There have been many software updates, and there will continue to be more software updates as we move from

feature build-out through feature completeness and then into performance improvement.

Figure 4 depicts the set of software updates plotted in time. Some notable features are visible in the plot. The flurry of updates before November 2023 correspond to our pre-launch campaign in which we ensured all the software was ready for launch. The first software updates after launch were minor tweaks to help with radio link reliability. The spike in December 2023 was in support of solar array deployment and associated spacecraft attitude control mode changes as we transitioned into a new phase of the mission. The spike towards the end of February 2024 was related to a systemic software bug in telemetry flow related to high water marks on sockets. The subsequent change in slope starting in March 2024 reflects an operational change wherein flight software updates were accepted weekly (more or less as the software became available); prior to that, software was accepted as-needed or in support of specific activities.

The expectation is that, in the near future, as the Smallsat platform approaches feature-completeness, the update cadence will slow to something similar to the steady state of the Doves.

Summary

At Planet, we champion Agile Aerospace development and have applied these philosophies and processes to a nascent spacecraft program: the Smallsat program that both Pelican and Tanager constellations are built upon. For Pelican-1, a first-of-a-generation prototype spacecraft, the first of the larger Smallsat program, we placed a large emphasis on delivering a working, tested, albeit feature-incomplete platform to the launch base. We focused on and continue to focus on minimum viable product (MVP) to ensure the team is working on the right features and bug-fixes at the right time. We try to integrate early and iterate often. Software and hardware updates are seen as a natural part of the process. Our software update process is safe, well tested, and thanks to our Missions, Operations, and Flight Software teams, fairly automated. Pelican-1 and the larger Smallsat program as a whole have benefitted from many, many software updates, helping us iterate to a reliable, robust system.

Agile Aerospace allowed us to deliver Pelican-1 in 2 years from conception to reality, prove out the hardware on orbit, and fold all the lessons learned into the next spacecraft. The rest of the Smallsat program, including all subsequent Pelican and Tanager spacecraft, will benefit from both the software and hardware iterations.

References

1. DOD-STD-2167A
2. Petersen, K; Wohlin, C.; Baca, D. "The Waterfall Model in Large-Scale Development" 10th International Conference on Product-Focused Software Process Improvement 2009
3. Stwartwout, D. “Agile Software Engineering in NASA’s Waterfall World” 17th Annual Workshop on Spacecraft Flight Software 2024
4. Howard, B. “What is Agile Aerospace? Learn Planet’s Approach” 2019.
5. Beck, K.; Beedle, M.; Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; Kern, J.; Marick, B.; Martin, R. C.; ; Mellor, S.; Schwaber, K.; Sutherland, J.; Thomas, D.; "Manifesto for Agile Software Development". Agile Alliance. 2001
6. Devaraj, K. “Planet’s Modular, Extensible Smallsat Platform Enables New Missions Including Pelican And Tanager” 2024
7. Planet “Our Next-Generation Satellite Constellation Pelican Is Expected To Deliver Very-High-Resolution And Rapid-Revisit Capabilities” 2022
8. Planet “The Hyperspectral Constellation” 2024
9. Mason, J. “Agile Aerospace: Rapid Development Of Earth Observation Tech For A Rapidly Changing World” 2023
10. Doan, D.; Flores-Pozo; K., Mankar, A.; Mcgill, L. “Agile Aerospace: Lessons Learned from Planet Mission Operations” SpaceOps Conference 2021