Agile Systems Engineering for sub-CubeSat scale spacecraft

Konstantinos Kanavouras
University of Luxembourg, konstantinos.kanavouras@uni.lu

Andreas M. Hein
University of Luxembourg, andreas.hein@uni.lu

Maanasa Sachidanand
ISAE-SUPAERO, France, maanasa.sachidanand@student.isae-supaero.fr

Abstract

Space systems miniaturization has been increasingly popular for the past decades, with over 1600 CubeSats and 300 sub-CubeSat sized spacecraft estimated to have been launched since 1998. This trend towards decreasing size enables the execution of unprecedented missions in terms of quantity, cost and development time, allowing for massively distributed satellite networks, and rapid prototyping of space equipment. Pocket-sized spacecraft can be designed in-house in less than a year and can reach weights of less than 10g, taking away the considerable costs and requirements typically associated with orbital flight. However, while Systems Engineering methodologies have been proposed for missions down to CubeSat size, there is still a gap regarding design approaches for picosatellites and smaller spacecraft, which can exploit their potential for iterative and accelerated development. In this paper, we propose a Systems Engineering methodology that abstains from the classic waterfall-like approach in favor of agile practices, focusing on available capabilities, delivery of features and design "sprints". Shifting away from the typical design-verify-operate model, this method originates from the software engineering discipline, focusing more on short design iterations, team collaboration, and focusing on quickly delivering a minimum viable product. This allows quick adaptation to imposed constraints, changes to requirements and unexpected events (e.g. chip shortages or delays), by making the design flexible to well-defined modifications. Two femtosatellite missions, currently under development and due to be launched in 2023, are used as case studies for our approach, showing how miniature spacecraft can be designed, developed and qualified from scratch in 6 months or less. Both missions involve the attachment of a chip-sized satellite ("ChipSat") into a larger spacecraft, either relying on their host for communications and power or being completely independent. We claim that the proposed method can simultaneously increase confidence in the design and decrease turnaround time for extremely small satellites, allowing novel and unprecedented missions to take shape without the overhead traditionally associated with sending cutting-edge hardware to space.
keywords: systems engineering, femtosatellites, attosatellites, chipsat, agile, scrum

Acronyms

CD	Continuous Delivery
CI	Continuous Integration
COTS	Commercial Off-The-Shelf
IC	Integrated Circuit
MBSE	Model-Based Systems Engineering
PCB	Printed Circuit Board
RF	Radio Frequency
SoC	System on Chip
TDD	Test-Driven Development
UL	University of Luxembourg

1. Introduction

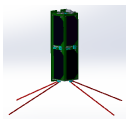
CubeSats are a class of spacecraft categorised as *nanosatellites* [1], which typically weigh less than 10 kg and considerably reduce the effort required to reach orbit. Since the publication of the CubeSat standard [2], more than 1600 CubeSats have been launched into orbit [1].

While CubeSat trends already had a momentous impact in space engineering and are considered of low relative cost and effort in the space industry [3, 4], they still require significant investment from designers and builders. Newcomers in the CubeSat world, such as educational institutions or start-ups, are usually faced with a build and launch cost of more than 200 000 $, and at least some years of development (Table 1). The effort increases further if multiple CubeSat launches are desired.

The considerable cost and effort to create CubeSats has therefore sparked ideas for even smaller classes of spacecraft: ThinSats [9], PocketQubes [6], ChipSats [10] and others. These so-called **"sub-CubeSat" spacecraft** are often categorised, according to their mass, into: [4, 5]

- **pico-satellites** (100 to 1000 g),
- **femto-satellites** (10 to 100 g), and
- **atto-satellites** (1 to 10 g).

Table 1: Comparison between classes of nano-satellites and smaller [5, 6, 7, 8]

	CubeSats	PocketQubes	ChipSats
			
Mass	1 to 10 kg	100 to 1000 g	1 to 100 g
Manufacturing Cost	10^5 $	10^4 $	10^3 $
Launch Cost	10^5 $	10^4 $	10^3 $
Development Time	3 to 5 years	3 to 5 years	4 to 8 months
Power	1 to 10 W	0.5 to 2 W	0.1 to 0.5 W
RF Data Rate	0.1 to 400 Mbps	10 to 250 kbps	10 to 2000 bps

Such smaller spacecraft take advantage of the increasing miniaturisation of off-the-shelf mechanisms and electronics to deliver value by significantly decreasing cost and development effort. The cost of a single flight model for an atto-satellite may be less than 1000 $ [11], while the development time for one unit can be less than 6 months, even with all subsystems being built in-house [5].

As of 2022, more than 350 known pico, femto and atto-satellites have been launched as individual units or as parts of constellations (Table 2). Developers have extrapolated the CubeSat standard into 1/2U, 1/3U and 1/4U form factors [1, 12]. "PocketQubes" [6] are a standardised version of modular picosatellites, which can weigh up to 250 g per cubical unit, and have already started having commercial implementations [13, 14]. Other design classes for sub-CubeSat spacecraft have been proposed, such as SunCubes [15], PCBSats [16, 17], and more [18].

One of the most representative examples of sub-CubeSat missions is the KickSat mission designed in Cornell University [10, 11]. This mission deployed 105 satellites which belong in the "ChipSat" spacecraft class. ChipSats are atto-satellites where all components and subsystems are integrated on a single PCB board, resulting in extremely low mass and size.

Table 2: Number of known nanosatellite (and smaller) launches [1, 19]

CubeSats	1604
Nanosatellites (non-CubeSat)	86
PocketQubes	46
Picosatellites (non-PocketQube)	217
Femtosatellites	4
Attosatellites	105

1.1 Space Systems Lifecycle Models

Space systems development has traditionally followed a top-down, waterfall-like approach [20, 21]. The life cycle of a project typically follows a set of predefined phases, starting from the mission conceptualisation, proceeding with the detailed design definition, and finishing with qualification and then flight [22]. This is often modelled as a V-diagram, showing how the system concept influences component design, leading back to system validation in the end [23, 24]. Development traditionally follows a *stage-gate* process, where a project is evaluated and its continuation determined at specific points during its lifetime [25].

In conjunction with the waterfall model, incremental or iterative practices are often used [26]. **Incremental** development refers to a method where distinct parts of the design are delivered one at a time; **iterative** development refers to an approach where the complete system is being refined after each of many cycles — the goal being that each cycle leads to an improved product, by using the outputs and lessons-learned from the previous cycles [26]. More systems engineering concepts have been extensively explored, such as the spiral model [27], concurrent engineering [28], and Model-Based Systems Engineering (MBSE) [29]. It is also common to use a *hybrid* approach, combining together different methods [30, 31], for example by using different product management techniques in different phases/parts of the project.

These traditional approaches have seen success with large, monolithic missions [20, 26], especially with safety-critical systems [32]. However, smaller spacecraft like CubeSats have characteristics (such as lower complexity, lower costs, shorter schedules, higher risk acceptance, easier integration, smaller teams) which do not always justify the overhead added in terms of cost, time and resources added by naively implementing traditional methods [25,

33, 34, 35].

In CubeSats, different approaches can be followed, depending on the nature of the project (commercial or educational) and the structure of the team:

- **Sequential approaches**: The typical progression of phases is followed (concept, design, assembly, verification and operations), but with modifications or shortening of each project phase. Usually phases 0, A and B (from conception until preliminary design) are combined into a single Phase AB. In some cases, a demonstration of the mission feasibility using a prototype may be required from Phase AB [36, 37, 38].
- **Evolutionary approaches**: Many CubeSat projects follow iterative or incremental approaches, where design, development, and testing may happen concurrently or repeatedly [37, 39, 40]. For example, it is common to manufacture engineering models or other representations of subsystems, before proceeding to system-wide assembly [41]. It is also common to work in an iterative approach for the entire system, by producing different "versions" of a CubeSat [42, 43]. Iterative development can also happen by working on a reduced version of the entire satellite before assembling; this is often implemented through the "FlatSat" approach, where subsystems can be tested long before feature completeness [44, 45]. Software, more specifically, can be tested continuously and independently from the rest of the subsystems, leading to quick verification [46].
- **Emergent approaches**: CubeSat projects have applied approaches fitting the term "agile" [37, 47, 48], where the focus is shifted from requirements compliance to human interactions and customer satisfaction [49]. More approaches again focus on agile software development [50, 51].

Pico-satellites, and especially femto- and atto-satellites, have a number of characteristics and capabilities (notably low cost, small size, fast development, mass production, potential for rapid technology testing, small development teams [5]) that call for new systems engineering methods, specific to them.

It is generally accepted that plan-driven approaches are better suited for large projects and teams, which require low risk and predictability [52, Sec. 2]. In contrast, sub-CubeSat spacecraft are small developments that can benefit from quickly responding to change, reduction of risk through redundancy, and small, knowledgeable teams. In this case, a traditional approach would add costly overhead in information sharing [33], unnecessary bureaucracy [52], high inertia [25], and would not adapt quickly to new technological developments [35].

In this work, we claim that:

a) By creating a tailored methodology, we can produce femto- and atto-satellites in only a few months, with a development cost at least an order of magnitude lower than one of a CubeSat, and with minimal schedule overruns.

b) Already existing systems engineering approaches for simple systems can be easily tailored to a femto- or attosat, in order to satisfy the previous claim.

2. Research Method

To resolve the claims of the previous section, we will propose a systems engineering method [53] tailored to femtosatellites and attosatellites.

In order to gain data and verify our method, we performed a Descriptive Study [54]. The nature of space systems development means that it is difficult to repeat similar conditions in a controlled environment. At the same time, due to the novelty of sub-CubeSat spacecraft, there is not enough available information to perform statistical analyses. Therefore, we applied the *case study* data collection method [54, 55], by following two case studies of miniaturised missions in UL (Section 2.1).

In order to develop our method, we will start by investigating popular frameworks for rapid prototyping and development, which can be easily tailored to the characteristics of sub-CubeSat spacecraft. After selecting a framework that seems best suited to these characteristics, we will apply the lessons learned from the two case studies, as well as suggestions from the literature, in order to create a more well-tailored method (Section 2.2).

2.1 Case Studies

The two case studies selected are missions that, as of 2022, are under manufacturing in the University of Luxembourg (UL) by the SpaSys team. The missions are two femto platforms that are used for in-orbit technology demonstration: a) A 10 × 10 cm payload, used to test Artificial Intelligence for thermography, and b) A 5 × 5 cm chipsat, used to test visible light communication.

While the first mission is a payload and not an independent satellite per se, the characteristics of its development are similar to the ones of an attosatellite, as there is limited functionality that needs to be achieved with low cost, but there are significant constraints in terms of available space, data budget and interfaces.

Our selected case studies are differing in type, technologies used, complexity, fractionation and team structure. This diversity is useful to evaluate if our method can be

generalised to different spacecraft. We can then test the *external validity* [55] of these case studies, i.e. we will use theoretical results to generalise from their specific findings to generic femto- and atto-satellites.

2.1.1 AI4Space

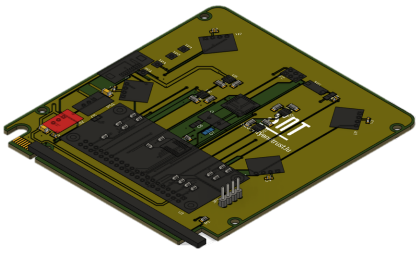


Fig. 1: Render of the AI4Space payload

The objective of the AI4space mission is to launch computer vision algorithms in space and serve as a testbed for the end-to-end development of space systems in UL, in collaboration with the CVI2 research group. The compact-infrared payload of the AI4Space mission aims to detect thermal anomalies on space electronics hardware. The thermal anomalies are detected using Infrared Thermography and Artificial Intelligence. Infrared thermography is a non-invasive method that uses infrared cameras to detect temperature variations of electronic components.

The payload is a PCB (Printed Circuit Board) and has a size of 100 mm × 100 mm × 15 mm and it is hosted by the Skyride payload hosting programme of Skykraft. It has the following subsystems:

- A Raspberry Pi, containing the onboard software developed in Python.
- A "mothercraft" interface, exchanging telecommands and telemetry with the Raspberry Pi.
- A power interface, supplying power to the payload from the mothercraft.
- An Arduino, acting as a target board that shows temperature variation when its clock frequency is modified.
- Heaters, used to increase the temperature significantly to be captured by the infrared cameras.

2.1.2 ChipSat

The "ChipSat" mission consists of a PCB containing 3 independent attosatellite designs, which will be mounted on the side of a microsatellite. The objective of this mission is to investigate the feasibility of free-space visible optical light communication between different satellites of a fractionated system. The "network" consists of two types

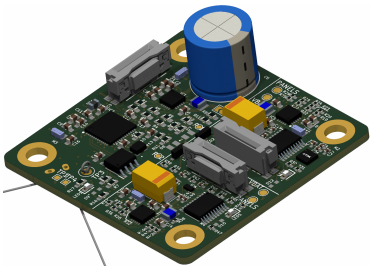


Fig. 2: Render of the ChipSat PCB

of satellites: a "primary", responsible for communication with the ground, and a "secondary", responsible only for data generation. For its maiden flight, the ChipSat will be mounted on a microsatellite, but electrically independent from it. More specifically, the contained components are:

- Microcontrollers, containing the onboard software developed in C++. For the primary, it is an SoC that contains an RF transceiver.
- LEDs and photodiodes, used to perform low-rate visible communication.
- Miniaturised solar panels, connected to an energy harvester IC, generating an average sunlight power of ~200 mW.
- For the primary, a deployable dipole antenna for low data-rate signals.
- For the secondaries, a gyroscope and an ambient light sensor.

2.2 Agile

Out of the methodologies analysed in Section 1.1, we focused on "emergent" approaches, which best suit ChipSat characteristics [35]. These approaches are usually grouped under the term "agile", which originates from the *Manifesto for Agile Software Development* [49], published in 2001.

The Agile manifesto aims to increase productivity by pursuing the following four values: [49]

1. *Individuals and interactions over processes and tools*
2. *Working software over comprehensive documentation*
3. *Customer collaboration over contract negotiation*
4. *Responding to change over following a plan*

Agile approaches therefore often follow an iterative cycle [26], each step of which results in a deliverable, working product.

While Agile originates from Software Engineering and has been observed to improve the effectiveness of engineers [56, 57], it was quickly generalised to systems engineering [58], citing improved engineering efficiency, early Return On Investment, responsiveness to change and increased project control [59, 60, 61].

Practices considered to be agile have already been considered and used, partially or completely, in various space missions [25, 30, 63] and especially in CubeSats [50, 64, 65, 66]. Agile is often used in small teams with limited resources which need turnkey developments and do not have significant risks involved [25].

Agile practices are usually implemented through specific frameworks. In this article, out of the rigorous frameworks defined in Boehm [52], we are focusing on the most popular ones [57], shown in Table 3.

In order to choose a baseline to develop a tailored method, we selected a network out of those based on the following criteria:

- **Easy adaptability to hardware**: Most Agile frameworks are specifically built around software-based practices and tools, such as daily unit testing and instant integration, which cannot be implemented in a larger system. In this article, we consider frameworks that can be easily applied to physical systems.
- **Covering full lifecycle**: Many Agile frameworks cover only certain parts of a product's lifecycle, or provide solutions only for one specific project management aspect. For convenience, we consider frameworks that cover aspects from planning to verification.
- **Small systems**: Some Agile frameworks are built for larger systems with higher complexity and larger teams. In this work, we consider frameworks that claim to work for ~10 maximum people.

From Table 3, it seems that *Scrum* and *Crystal Clear* are the most suitable frameworks for our proposal. In this work we will focus on *Scrum*, because of its very high popularity and available literature [57]. Scrum has also specifically been proposed for use in Systems Engineering, especially for systems that are accepting to rapid changes, by Bott et al. [67].

However, we do note that our search is by no means exhaustive; other frameworks may also present relevant opportunities for tailoring.

3. Applying Agile to sub-CubeSat spacecraft

In order to develop the new method, we will apply the characteristics of sub-CubeSat spacecraft to published findings related to agile systems engineering. We will also use the lessons learned from the two case studies (Section 2.1).

3.1 Distinctive femtosatellite characteristics

To prepare a set of guidelines for the use of Agile in sub-CubeSat developments, we first present some of the most notable distinctive characteristics of such spacecraft [5, 10, 68]:

- Low design complexity
- Low manufacturing cost
- Low launch cost, due to low mass
- Femtosatellites and smaller spacecraft often have little distinction between subsystems in the traditional sense. Using System on Chip (SoC) technology, multiple or even all subsystems can be combined into a single component [69].
- Technology reuse is not yet explored in detail. In contrast to the CubeSat standard and the relevant deployer specifications and component market [2], femtosatellites and smaller are not supported by off-the-shelf specifications or modules as of 2022. However, standardisation for PocketQubes is already underway [6, 70].
- Interface requirements for the deployer and/or launcher are invariants and will usually serve as the main drivers for the design.
- Engineering teams working on sub-CubeSat spacecraft usually consist of few people, and are not spread into different subteams.
- The facilities for manufacturing and testing such spacecraft can often be available in-house [71].

3.2 Agile principles

Douglass (2015) [59] provides seven core ideas for agile methods. In this section, we will restate and analyse their applicability to sub-CubeSat projects.

3.2.1 Work Incrementally

If a product is separable into distinct parts, which can be developed, tested (and preferably deployed) independently, then the work can be easily divided into multiple cycles.

When a system is considered "complete", i.e. it is operational and fulfils the basic objectives of a mission, incremental development can turn into iterative development, where the focus is on the rework and improvement of the existing system, rather than the implementation of critical missing features [26]. This is especially applicable to femto- and attosats, as the time between conception and manufacturing can be very short.

Table 3: Comparison of popular agile frameworks [52, 62]

	Adaptable to hardware	Full lifecyle	Small systems	Comments
Kanban	Yes	No	Yes	Does not explicitly address verification/testing
Scrum	Yes	Yes	Yes	
Lean Development (LD)	Yes	No	Yes	Strategic, risk-driven approach, not focused on systems engineering
Crystal Clear	Yes	Yes	Yes	"Crystal" method for very small teams
eXtreme Programming (XP)	No	No	Yes	
Dynamic Systems Development Method (DSDM)	Yes	Yes	No	Closer to plan-based "traditional" methods, emphasis in management activities
Feature-Driven Development (FDD)	No	No	No	Mostly focused on individual practices for software development

The time between each cycle varies between each projects. While 2 weeks are common in software engineering projects, systems where hardware and physical effort is involved may benefit from longer iterations (4–6 weeks).

3.2.2 Plan Dynamically

A common myth is that Agile practices call for no planning or bureaucracy [59]. Douglass [59] claims that *"planning is important but only if it is accurate"*. Agile principles introduce a degree of uncertainty in all layers of a design, from customer requirements to implementation. The project plan therefore should be defined, but also dynamic: it should reflect this uncertainty and be frequently updated (e.g. every 1–2 months) to improve the assumptions made about the needed amount of work and "velocity" of the team.

3.2.3 Actively Reduce Project Risk

In terms of project planning, uncertainties can introduce risk. Design changes or active actions can help to mitigate risks that are identified early or late in the design.

In our experience, focusing on the critical path in terms of schedule, or the components with the highest impact in the system, will help identifying the most threatening risks. For example, any interaction with an external supplier or provider is coupled with uncertainty and delays, and should hence be started as early as possible during the project. To mitigate against a possible chip shortage, designers should procure critical and irreplaceable components as early as possible (even during the design), and maintain a reliable stock.

3.2.4 Verify Constantly, Integrate Continuously, Validate Frequently

Verification in space systems is usually formal and follows a unit–subsystem–system path [24]. On the other hand, in software systems, verification is done through automated test suites, and can be completed in just a few minutes [72].

While it is easy to test spacecraft software, especially when complemented by a hardware-in-the-loop environment [73], the engineering effort to manufacture and test space hardware can often take years. This reduces the responsiveness to change and the chance to identify errors early in the design [74, 75].

However, when developing picosatellites and smaller spacecraft, the manufacturing and assembly can be made in a few months with little overhead. This means that **the verification of representative hardware can become an important part of the design process, as verification can happen at the end of each development cycle**. The hardware used in these cases could usually resemble a Structural Model, a Development Model or an Engineering Model [76]. Engineering Models in particular should be representative in terms of form, fit and function, without requiring hardware and processes suitable for space.

Especially for femto- and attosattelites, low launch costs mean that there is minimal effort required to reach orbit. **A system can be iteratively verified by sending its different versions into orbit.** Lessons learned from the

actual operation of a satellite will then be used to improve the next iterations. In software engineering, this concept is often referred to as **Continuous Integration/Continuous Delivery (CI/CD)**, and involves constant releases of a product which, while not perfect, are fully functional and operational [77, 78].

Alternatively, as a middle-ground solution, high-altitude balloon flights, or even actual usage of the spacecraft on the ground, could be implemented [79].

Given the above, we can summarise the different verification methods for sub-CubeSat spacecraft: [76]

1. Analysis and Review of Design
2. Testing (software only)
3. Testing (software and hardware)
4. Field operations (on-earth demonstrations, suborbital flights)
5. In-flight operations

We note that, based on the available resources and objectives, a different verification method out of the above can be chosen for different cycles, phases or parts of the same project. We also note that this sequence does not need to be followed in order; for example, after a test flight, the team may continue with hardware testing of the next iterations.

Validation [76], usually in collaboration with the customer, also happens regularly in the context of Agile.

3.2.5 Modeling Is Essential for agile MBSE

Model-Based Systems Engineering (MBSE) practices can be applied to agile systems engineering, especially if they do not need considerable overhead and are easy to implement [59, 80].

For example, an interesting concept in software engineering is that of **Test Driven Development (TDD)**. In TDD, automated tests are written before the development of the code itself [72]. The task of the developer is then to make sure that the developed system passes all the tests. Consequently, in TDD, the test itself serves as a specification or a model for the requirements of a system.

Models can also be used for automated generation of software [81, 82], or for the reduction of the work needed to document a system. However, mechanicals and hardware [83] still require manual effort to integrate into a model.

4. Proposed Methods

In this section, we will propose two agile methods that follow from the analysis performed in the previous paragraphs. The first one is the result of getting inspiration from the traditional V-model, and modifying it with some additional flexibility, so that it meets the definition of agile.

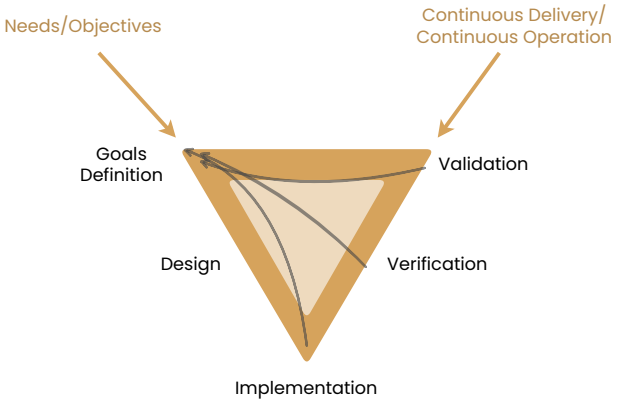


Fig. 3: The triangle model

The second one starts by directly implementing the Scrum agile framework, and applying to it the constraints of a sub-CubeSat spacecraft.

4.1 The Triangle model

In the core ideas described in the previous section, the overarching theme of **iterative development** is dominant. We can therefore adapt the V model into a **triangle model**, where paths exist for **going back to the conceptual stage** at every point during the design. We will use this triangle model as a representation [53] for the proposed model.

Instead of requirements, the triangle model originates from **needs and objectives**. We define these as qualitative targets that a customer or entity has for a mission, such as *creating space heritage within an organisation*, or *measuring temperature variations during reentry*.

Given the needs and objectives, the more specific and quantitative goals are stated later. We still refrain from using the word "requirements", as we want to avoid the rigour and effort connected with defining low-level requirements at the beginning of a project. However, we note that these goals still need to be well-defined, and can refer to the mission, the system, or specific units. For example, a goal would be to *take at least 10 measurements during reentry*, *develop a spacecraft with a* <25g *mass*, or *have a data rate of at least 100*kb/s.

The next steps in the process follow the design and implementation. We deliberately did not distinguish between *high-level* and *low-level* design, or *subsystem-level* and *component-level* design. This is done because pico-scale spacecraft can incorporate different subsystems into single components or units. It is up to each team to determine if they will split the design into multiple levels, based on their size and objectives.

Verification and validation follow after implementation. Validation marks the end of one iterative or in-

cremental cycle. The developers can then proceed with the operation of the validated product (Section 3.2.4), and they can start the next cycle from a refinement of the initial goals.

However, the triangle model does not only follow this circular path: Any point in the triangle can lead to **backtracking**, i.e. going back to the goals definition. This is essentially the **mechanism to respond to changes** during the design. For example, a team might start with a goal of *reaching a data rate of at least* 100 kbit/s *with a* 5×5 cm *attosat*. During the implementation phase, they might find out that their radio transceiver is too large to fit on the attosat's PCB. They will then be forced to backtrack, and change their goals to increase the satellite's size, or support slower data rates with a smaller transceiver.

A significant difference between typical incremental models [35] and the triangle model is that the latter does not require going through all the phases before backtracking: Lessons learned during design or implementation may immediately lead to a redefinition of the goals, without a need to go through the next phases.

An implementation of the triangle model also does not need to begin from the top left. The rapidly-developing COTS component ecosystem means that, for example, a proposed implementation may be available before a specific use case has been prepared. In this case, the system becomes largely **capability-driven** instead of goal-driven.

4.2 Sat-Scrum

While the Triangle Model proposed in Section 4.1 presents significant differences in terms of flexibility in project management, it is still covering a high level of the design process, and, as a tailoring of a more "traditional" process, may not fully take into advantage the capabilities offered by miniaturised spacecraft.

Therefore, continuing the analysis presented in Section 2.2, in this part we will focus on *Scrum*, and how it can be adapted to our target platform.

Scrum is a lightweight agile framework that covers all life cycle activities of a project or a system, and is based on empiricism, focusing on transparency, inspection and adaptation [67, 84].

The definitions of Scrum [84] contain a lot of nuance and terminology that is suggested to be followed by participating projects, to encourage a shift in mentality from a traditional plan-based approach. Here we will focus on the idea of **Sprints**, which correspond to the cycles of an incremental/iterative process. A sprint starts from a backlog (prioritised list of work items) and results in a sprint goal, which might be the next version of a product. A sprint usually includes:

- **Sprint Planning**, a long meeting where a goal and plan is laid out for the next sprint
- **Daily Scrums**, which are 15-minute (timeboxed) meetings that take place to gauge progress and update the short-term planning.
- **Sprint Review**, a technical session at the end of a sprint, where its outcome is reviewed and technical lessons learned are assessed.
- **Sprint Retrospective**, a meeting called after the Review, where the focus is on interactions, processes and tools, and how the team can solve problems that occured in the future.

Scrum also defines the positions of the "Scrum Master", the Product Owner and the Developers, the last two of which are the main responsibles for selecting the backlog work items.

Applying the idea of Sprints to a space project, in the same vein as the triangle model, can result to something similar to what is shown in Figure 4.

In this case, every sprint results in a functional, improved product. During development, verification can be done through one of the five methods listed in Section 3.2.4. In this case, even **in-orbit testing can be part of the normal product development procedure** in each sprint. Flight models do not need to be "feature-complete", only operational; this is aided by the low resources that mainly femto- and attosats need to be launched, essentially if the manufacturer takes advantage of economy-of-scale effects [5].

Due to the complexity of the systems, the duration of a sprint in this case will lean towards the longer end of 4 to 6 weeks. The verification process also needs separate considerations, since it might not be able to be fully automated as in a software system.

In the case of physical hardware, verification might need some days or weeks to be performed, as an activity separate to development. It might include time spent for procurement or assembly. In specific project stages or situations (e.g. breadboard models, or modular assemblies) these activities might be able to be completed in less than a week, in which case they can be a regular part of the sprint.

However, larger developments (e.g. producing Engineering Models) or tests (e.g. test flights) might require significantly more involvement and cannot be completed as part of a single sprint. In this case, the "review & retrospective" parts for every sprint cannot fully reflect the verification done in the sprint. However, they lessons learned and results from this process can be implemented in future sprints.

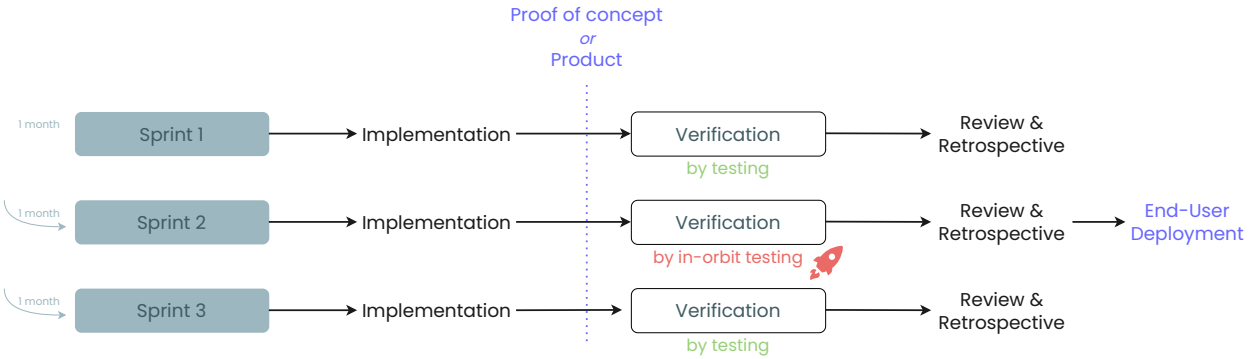


Fig. 4: Example of applying Scrum to sub-CubeSat scale space projects

A few other qualities of Scrum that might be interesting to analyse are:

- In Scrum, there is no clear separation between different functions of the developers: Design, development, quality assurance or analysis. Team members are self-managing and distribute work items based on their discretion.
- Scrum encourages transparency and information sharing for all parts of the work. This can be achieved by using the appropriate project management tools, having a central archive to gather results and information, avoiding personal communications, or even by sharing a common workplace.
- We note the differences between 'incremental' and 'iterative' development, as defined by Heeager et al. [26]. In the beginning of the project, a "scrum increment" will be identical to an "agile increment", as parts of the system will be built for each Sprint until it is functional. However, in the largest part of a project, a "scrum increment" will more closely match an "agile *iteration*", where the focus is on the rework of the system so that it implements more of the defined goals.
- Especially in a space system, the complexity of managing different increments, versions, physical products, test campaigns and even launches, would require significant **tooling** to remove the management overhead and make access to information easy. A combination or adaptation of tools already available in the software engineering, project management and space systems engineering might be useful for this purpose. For example, a Version Control system could be used to store different versions of software, designs or documentation; a Project Management tool built for Scrum can support the work allocation and planning for team members [85]; and Model-Based Systems Engineering can be used to formalise the most crucial and high-level parts of the process [59, 86]. However, it is important that the tooling used does not act as an impediment to the daily work of the developers.

5. Case Studies

In this section, we will provide details on the Case Studies presented in Section 2.1. We will start by presenting the development flow of these missions, as well as some practices that were followed and lessons learned. We will finally emulate the application of the methods of Section 4, following a "counter-factual" approach, since both missions already started before these methods were developed [87].

5.1 AI4Space

The project began with understanding the needs and objectives of the AI-driven payload. Then, the software and hardware architecture were documented to satisfy the space mission objectives. While we initially used MBSE to model requirements during the development phase, there was not enough time to complete and utilise this formalisation. COTS components were chosen instead of space-grade products since they were easily available, less expensive and reduced software and hardware development time. Additionally, ready-to-use development platforms such as Edge Impulse were used to embed AI algorithms on the computer module to reduce development time. Finally, the functionalities of the payload were tested using a breadboard model to validate the proof of concept, before building the qualification and flight models.

For the development of AI4Space, we experimented with a tailored approach that incorporated various iterative practices adhering to the agile manifesto [49], but without belonging in a concrete framework:

- We designed a system that was **shippable from the early stages**. We implemented the basic operational functionality with high priority in the first few months. This included basic operational functions, such as telemetry and telecommands, downlinking science data, or command scheduling ("time-tagging"). It also included at least a proof-of-concept for the payload, with the basic operations and sensor code being written only over the first 2 weeks after development started.
- **Multiple functional iterations** were made during development. We roughly followed the "Sprint" pattern, performing one iteration every 2 weeks, which was concluded with a review and planning meeting at the end of the sprint.

In order to manage all of the above, a hybrid method was used:

- The major mission milestones were managed in a traditional way, following a **Gantt chart**, and splitting the mission into specific large work packages, starting from *Preparation* until *Scientific Exploitation* of the results.
- For the design and development, **GitLab** as a tool for project management, work item tracking, and result sharing.
- For the assembly, we created a standard assembly procedure which included the list of the required components. Hardware stock tracking was done using Inventree to monitor the components purchased, current stock and components to order.

While we could easily adopt agile practices for the payload's software, the constraints imposed by hardware (mainly cost, time and part availability) are harder to work with [74, 75]. In the project's context, we worked simultaneously on a functional **Breadboard Model** that was developed incrementally, and the PCB designs for the **Qualification/Flight** models. The following principles were followed throughout development:

- "Stand-in" parts were used when a component or hardware function was not readily available. We deemed it more important to have a *functional* instead of a *representative* model through the early stages. For example, we used a Raspberry Pi 3 instead of a Raspberry Pi 0 for the early days of development.

  It is important to note that the loss of representativeness has to be recorded and known in the team, so that fewer surprises show up later in the process. In some cases, it might even be possible to emulate the differences in the configuration, for example by limiting the amount of memory available to the Raspberry Pi in software.
- The software was implemented in conjunction with the hardware.

  Specifically for AI4Space, our main platform was a Linux distribution on the Raspberry Pi. While it was possible to develop most parts of the software independently from the hardware ("off-line"), we opted to validate the written software on the hardware models immediately, as part of a CI/CD process.
- MBSE was used for the definition of interfaces. The "model" in this case was Python code, which included the formal definition of the interfaces, along with documentation. Python's `construct` library [88] could then immediately parse and generate telemetry & telecommands, using only this single source of data. We also prepared a script that automatically created human-readable documentation for this interface.
- Parts of the system were tested automatically, using the Robot automation framework [89]. The specific framework is not software-centred and provides a user-friendly test format, inspired by the concept of "user stories" in Behavior-Driven Development [90]. This means that:
  - Testing is not limited to software functions only; An appropriately configured system can also perform hardware verification.
  - Testers can be independent from the developers, since our tests were reasonably decoupled from the code they test.
  - Test stories could also be used for nominal procedures, such as clearing spacecraft logs.
  - There can be an additional slight learning curve, since developers need to become familiar with the domain-specific language introduced by the framework.

During development, we also observed that:

- The nature of the team meant that members had to spend a significant amount of their time in tasks unrelated to the project.
- Not enough time was dedicated to documentation, which steepened the learning curve for new members. However, the fast pace of development meant that any written documentation would very quickly become obsolete, even if it was written at a high level

only. Ideally, the appropriate level of documentation would be written as a required part of development, or the use of automated documentation generation tools could be extended.

Documentation need not only explain the function of a system; it can also contain information about the design justification. In our cases, this was done on a rolling bases: When opening and closing an issue on Gitlab, we added at least a short sentence explaining the reason for this action.

- Some unplanned events caused "road-blocks" that had to halt development and break momentum, such as procurement delays, supplier stock issues, and equipment or network issues. After these, we devised a "Minimum Working Environment" that required only a personal computer for a developer to work towards their goal.
- For software development, we followed an empirical approach instead of laying out a detailed plan beforehand, and we did not over-design, meaning we did not add more functionality, modularity or levels of abstraction, apart from what was needed in each iteration. This meant that parts of the code had to be **refactored** in some situations.

  While detailed planning might prevent time spent on refactoring, the Scrum Guide [84] suggests that empricicism and lean thinking reduce wasted time. In our case, many of those 'adaptations' resulted only from the direct application of lessons learned during the project, that would be harder to recognise earlier in the project.
- Team members were involved in all aspects of the project, including development, verification and manufacturing. However, in the case of AI4Space it would be ideal to allocate at least manufacturing to another team or an external contractor, so as to not impede the momentum and progress of every iteration.

The approach taken in the AI4Space fits the most basic approach of the triangle model, as described in Section 4.1. While the original goals of the mission were defined, after some iterations of design, proof of concepts and rudimentary testing, they were redefined so that the process could continue again; the team followed this cycle three times, back-tracking after the first breadboard proof-of-concept, and after the implementation of the AI algorithm.

As mentioned in Section 2.1, we can theorise the effects of applying the "Sat-Scrum" method of Section 4.2 to AI4Space. This could lead to a more structured workflow, as shown in Figure 5. In this case, two *in-orbit validation*

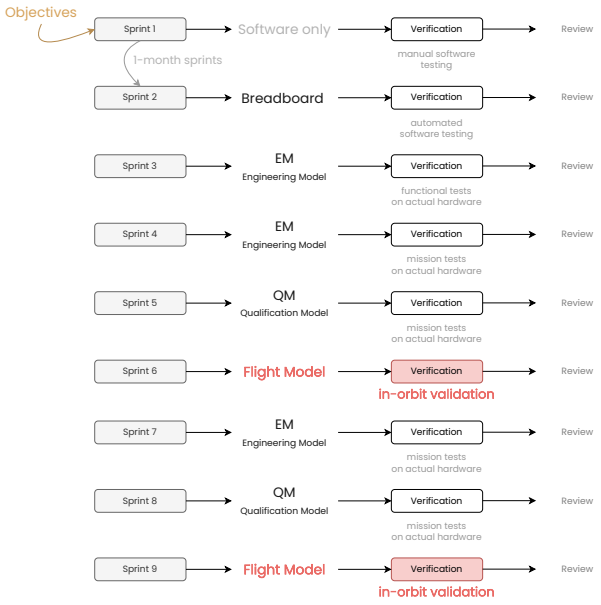


Fig. 5: Theoretical example of AI4Space scrum schedule

phases are implemented: The first is used as a technology demonstrator of the platform, only with some rudimentary functionality; the second would include a complete implementation of the original need. Some advantages of applying this method over a simple triangle would be:

- Clearer project structure and task allocation, using well-defined procedures and tools, and streamlined interactions between developers
- "Forced" communication between members: encourages information sharing and reduces misunderstandings
- Verification and documentation would be required for each sprint. Documentation would only need to be updated every month, and not at very fast intervals that would render it obsolete immediately.
- The inclusion of more Engineering Models would prevent last-minute issues from blocking the production of Qualification Models, at the expense of some administrative overhead and cost.

However, we note that the structure of Figure 5 is not a 'plan' in the conventional sense: Since Scrum is derived from empiricism and emergence, the actual decisions would be taken during the development cycle. Nonetheless, milestones such as launches cannot be changed, and have to be considered as invariants by the Scrum team.

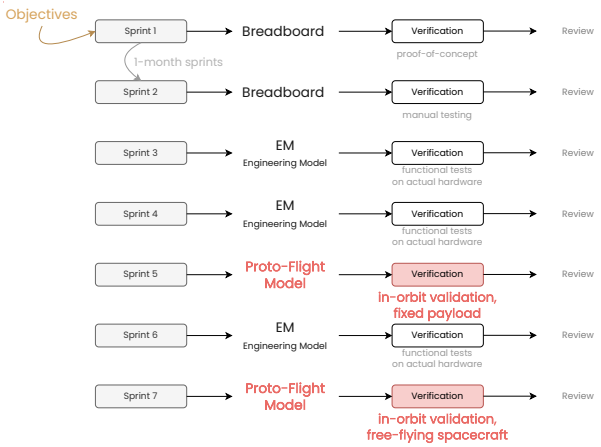


Fig. 6: Theoretical example of the ChipSat scrum schedule

5.2 ChipSat

The ChipSat project also followed some agile principles, in combination with some traditional planning:

- A **Gantt chart** was made using assumptions early in the project. However, it was seen that frequent updates to the Gantt chart were inevitable, but it is regularly updated after any new development.
- Continuous changes in the requirements, capabilities and results meant that the project had to be **easily adaptable** to any changes. For example, the PCB design had to be updated a few times in all stages of the project to accomodate new geometry restrictions.
- The team did not set in stone the number of iterations or designs that would be manufactured. Rather, since the procurement and manufacturing cost was low, a complete chipsat board was procured after every design iteration.
- Software and hardware development both started from day 0. This produced invaluable information regarding the feasibility of different design choices, and provided an approximate (yet adequate) reference model for internal use, but also for all external collaborators.
- **Project risk** was identified and attempted to be reduced. Especially regarding chip shortages, the procurement process for critical ICs was initiated immediately after a relevant design decision is taken, to prevent chip shortages from delaying the project by many months.
- Due to uncertainties, different "branches" or options of the project had to be worked on at the same time until resolution (e.g. version with/without batteries, with deployable/fixed antenna, etc.)

Modelling the design process with the triangle model of Section 4.1 reveals that the definition of goals in this project (in terms of payload, features and performance) began after building the physical proof-of-concept — i.e. the model did not start from the top, but started from the implementation, and then back-tracked to the goals definition.

Figure 6 demonstrates a hypothetical Scrum approach of the ChipSat development. Here we note that in-orbit validation can first happen as an integrated payload (or even a high-altitude baloon flight), while the ChipSat can be released as an independent spacecraft on a second mission.

The advantages of this approach would be similar to the ones for AI4Space. However, the smaller size of the system, combined with a mass of <20 g for the freestanding board, would make it more convenient to produce multiple models and test in real-world conditions. These produced functional ChipSat models could even be used on the ground for outreach, or as wireless, self-sufficient Internet of Things (IoT) nodes [79].

6. Conclusion

In this paper, we explored how the unique characteristics of sub-CubeSat spacecraft can provide opportunities for tailored design methods, which take advantage of their low cost and potential for rapid manufacturing and testing. To develop a tailored model, we received inspiration from two case studies in the University of Luxembourg, and from the principles of Agile systems engineering. We then devised two lifecycle models for the development of sub-CubeSat spacecraft, and we verified their theoretical application to the two case studies. We observed that by following one of the proposed models, we could keep development time for such missions down to 6–9 months, while iterating on physical builds of the system every ~1 month. This approach could reduce project risk by giving almost instant feedback to the designers about their design choices. By taking advantage of economy-of-scale effects, manufacturers can launch sub-CubeSat spacecraft as part of the development process, but keep manufacturing and testing costs in proportion to the spacecraft's mass. For future work, we propose more research on how other agile frameworks tailored to systems engineering (such as Crystal, SAFe or RUP [52, 67]) and MBSE could be applied to gram-scale spacecraft. We also propose investigation or development of specific tools to aid teams in their day-to-day task management. Finally, we propose further validating proposed methods for these spacecraft, by comparing the application of different methods to actual case studies.

References

[1] Erik Kulu. *Nanosats Database*. Nanosats Database. 4th Apr. 2021. URL: https://www.nanosats.eu/index.html (visited on 29/06/2021).

[2] California Polytechnic State University. *CubeSat Design Specification Rev. 14.1*. CP-CDS-R14.1. Feb. 2022.

[3] Armen Poghosyan et al. "CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions". In: *Progress in Aerospace Sciences* 88 (1st Jan. 2017), pp. 59–83. ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2016.11.002.

[4] Martin N. Sweeting. "Modern Small Satellites-Changing the Economics of Space". In: *Proceedings of the IEEE* 106.3 (Mar. 2018), pp. 343–361. ISSN: 1558-2256. DOI: 10.1109/JPROC.2018.2806218.

[5] Andreas M. Hein et al. *AttoSats: ChipSats, other Gram-Scale Spacecraft, and Beyond*. 31st Dec. 2019. DOI: 10.48550/arXiv.1910.12559.

[6] S Radu et al. *The PocketQube Standard*. Alba Orbital, TU Delft, G.A.U.S.S. Srl, 7th June 2018.

[7] Stefano Speretta et al. "CubeSats to PocketQubes: Opportunities and Challenges". In: 67th International Astronautical Congress. 26th Sept. 2016.

[8] Nasir Saeed et al. "CubeSat Communications: Recent Advances and Future Challenges". In: *IEEE Communications Surveys & Tutorials* 22.3 (2020). Conference Name: IEEE Communications Surveys & Tutorials, pp. 1839–1862. ISSN: 1553-877X. DOI: 10.1109/COMST.2020.2990499.

[9] Robert Twiggs et al. "The ThinSat Program: Flight Opportunities for Education, Research and Industry". In: *Small Satellite Conference* (6th Aug. 2018).

[10] Zachary Manchester. "Centimeter-Scale Spacecraft: Design, Fabrication, And Deployment". PhD thesis. 17th Aug. 2015.

[11] Zachary Manchester et al. "KickSat: A Crowd-Funded Mission to Demonstrate the World's Smallest Spacecraft". In: *Small Satellite Conference* (14th Aug. 2013).

[12] Joseph Gangestad et al. "Flight Results from AeroCube-6: A Radiation Dosimeter Mission in the 0.5 U Form Factor". In: *12th Annual CubeSat Developers Workshop, San Luis Obispo, CA*. 2015.

[13] *Unicorn 2 Platform*. Alba Orbital. URL: http://www.albaorbital.com/unicorn-2 (visited on 09/09/2022).

[14] *FOSSA Systems - Our dedicated picosatellite platforms for IoT*. 27th May 2021. URL: https://fossa.systems/satellites/ (visited on 09/09/2022).

[15] A. S. U. News. *SunCube miniature satellites*. 5th Apr. 2016.

[16] David Barnhart et al. "Enabling Space Sensor Networks with PCBSat". In: AIAA/USU Small Satellite Conference 2007. Vol. SSC07-IV-4. Logan, Utah, 13th Aug. 2007.

[17] Haoran Gong et al. "Design of foldable PCBSat enabling three-axis attitude control". In: *Acta Astronautica* 192 (1st Mar. 2022), pp. 291–300. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2021.12.004.

[18] Tracie R. Perez et al. "A Survey of Current Femtosatellite Designs, Technologies, and Mission Concepts". In: *Journal of Small Satellites* 5 (1st Oct. 2016), pp. 467–482.

[19] Tom Abate. *Inexpensive chip-size satellites orbit Earth*. Stanford News. In collab. with Zachary Manchester. Section: Science & Technology. 3rd June 2019. URL: https://news.stanford.edu/2019/06/03/chip-size-satellites-orbit-earth/ (visited on 29/08/2022).

[20] Garrett Shea. *NASA Systems Engineering Handbook Revision 2*. NASA. 20th June 2017. URL: http://www.nasa.gov/connect/ebooks/nasa-systems-engineering-handbook (visited on 05/07/2022).

[21] ECSS Secretariat. *ECSS-E-ST-10C Rev.1 – System engineering general requirements*. European Space Agency, 15th Feb. 2017.

[22] Miguel A. Aguirre. *Introduction to Space Systems: Design and Synthesis*. Space Technology Library. New York: Springer-Verlag, 2013. ISBN: 978-1-4614-3757-4. DOI: 10.1007/978-1-4614-3758-1.

[23] John O. Clark. "System of Systems Engineering and Family of Systems Engineering from a standards, V-Model, and Dual-V Model perspective". In: *2009 3rd Annual IEEE Systems Conference*. 2009 3rd Annual IEEE Systems Conference. Mar. 2009, pp. 381–387. DOI: 10.1109/SYSTEMS.2009.4815831.

[24] Bundesrepublik Deutschland. *V-Modell XT*. 31st Jan. 2006.

[25] Ronald S. Carson. "4.2.1 Can Systems Engineering be Agile? Development Lifecycles for Systems, Hardware, and Software". In: *INCOSE International Symposium* 23.1 (2013), pp. 16–28. ISSN: 2334-5837. DOI: 10.1002/j.2334-5837.2013.tb03001.x.

[26] Lise Tordrup Heeager et al. "A conceptual model of agile software development in a safety-critical context: A systematic literature review". In: *Information and Software Technology* 103 (2018), pp. 22–39. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.06.004.

[27] NASA Engineering & Safety Center. *Aligning System Development Models with Insight Approaches*. NASA Lessons Learned. 23rd Aug. 2018. URL: https://llis.nasa.gov/lesson/24502 (visited on 30/08/2022).

[28] Massimo Bandecchi et al. "Concurrent engineering applied to space mission assessment and design". In: *ESA Bulletin* 99 (Sept. 1999).

[29] P. M. Fischer et al. "Implementing model-based system engineering for the whole lifecycle of a spacecraft". In: *CEAS Space Journal* 9.3 (1st Sept. 2017), pp. 351–365. ISSN: 1868-2510. DOI: 10.1007/s12567-017-0166-4.

[30] Scott E. Carpenter et al. "Is Agile Too Fragile for Space-Based Systems Engineering?" In: 2014 IEEE International Conference on Space Mission Challenges for Information Technology. Sept. 2014, pp. 38–45. DOI: 10.1109/SMC-IT.2014.13.

[31] Nicola Garzaniti et al. "Toward a Hybrid Agile Product Development Process". In: 1st Feb. 2020, pp. 191–200. ISBN: 978-3-030-42249-3. DOI: 10.1007/978-3-030-42250-9_18.

[32] Rashidah Kasauli et al. "Safety-Critical Systems and Agile Development: A Mapping Study". In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Aug. 2018, pp. 470–477. DOI: 10.1109/SEAA.2018.00082.

[33] Ali Yassine et al. "Information hiding in product development: the design churn effect". In: *Research in Engineering Design* 14.3 (1st Nov. 2003), pp. 145–161. ISSN: 1435-6066. DOI: 10.1007/s00163-003-0036-2.

[34] Winston W. Royce. "Managing the Development of Large Software Systems". In: *Technical Papers of Western Electronic Show and Convention*. Los Angeles, USA, Aug. 1970.

[35] SEBoK Editorial Board, ed. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*. v. 2.6. Hoboken, NJ, 20th May 2022.

[36] Cristóbal Nieto-Peroy et al. "CubeSat Mission: From Design to Operation". In: *Applied Sciences* 9.15 (Jan. 2019). Number: 15 Publisher: Multidisciplinary Digital Publishing Institute, p. 3110. ISSN: 2076-3417. DOI: 10.3390/app9153110.

[37] Daniel Lubián-Arenillas et al. "Nanosatellite development methodology and preliminary design guides for the NANOSTAR Project". In: European Conference for Aeronautics and Space Sciences (EUCASS 2019). Madrid: E.T.S. de Ingeniería Aeronáutica y del Espacio (UPM), July 2019, pp. 1–11.

[38] Tyvak. *Trestles 6U/12U Platform Specification Booklet*. 19th Jan. 2021. URL: https://rsdo.gsfc.nasa.gov/images/catalog-rapidIV/Tyvak_Trestles_6U-12U_Catalog_Brochure.pdf (visited on 08/09/2022).

[39] Abdulaziz Alanazi et al. "Engineering Methodology for Student-Driven CubeSats". In: *Aerospace* 6.5 (May 2019). Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, p. 54. ISSN: 2226-4310. DOI: 10.3390/aerospace6050054.

[40] William Sousa. "CubeSat Development Framework". PhD thesis. Air Force Institute of Technology, 1st Mar. 2021.

[41] Pauline Faure et al. "Toward lean satellites reliability improvement using HORYU-IV project as case study". In: *Acta Astronautica* 133 (1st Apr. 2017), pp. 33–49. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2016.12.030.

[42] Zachary Scott Decker. "A systems-engineering assessment of multiple CubeSat build approaches". Accepted: 2016-12-05T19:10:36Z. Thesis. Massachusetts Institute of Technology, 2016.

[43] Jeroen Cappaert. "Building, Deploying and Operating a Cubesat Constellation - Exploring the Less Obvious Reasons Space is Hard". In: *Small Satellite Conference* (7th Aug. 2018).

[44] NASA CubeSat Launch Initiative. *CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers*. Oct. 2017.

[45] AcubeSAT Team. *AcubeSAT Manufacturing, Assembly, Integration and Verification File*. 17th May 2021.

[46] Jonis Kiesbye et al. "Hardware-In-The-Loop and Software-In-The-Loop Testing of the MOVE-II CubeSat". In: *Aerospace* 6.12 (Dec. 2019). Number: 12 Publisher: Multidisciplinary Digital Publishing Institute, p. 130. ISSN: 2226-4310. DOI: 10.3390/aerospace6120130.

[47] Boris Segret et al. "The Paving Stones: initial feed-back on an attempt to apply the AGILE principles for the development of a CubeSat space mission to Mars". In: SPIE Astronomical Telescopes + Instrumentation. Ed. by George Z. Angeli et al. Montréal, Quebec, Canada, 4th Aug. 2014, 91500W. DOI: 10.1117/12.2056377.

[48] Ralph LaBarge. "CubeSat – An Agile System Architecture?" In: *INSIGHT* 17.2 (2014), pp. 27–30. ISSN: 2156-4868. DOI: 10.1002/inst.201417227.

[49] Kent Beck et al. *The Agile Manifesto*. Feb. 2001.

[50] Alexander Lill et al. "Agile Mission Operations in the CubeSat Project MOVE-II". In: *2018 SpaceOps conference*. 25th May 2018. DOI: 10.2514/6.2018-2635.

[51] Sean Coyle et al. "EECSat: CubeSat Development". In: 2020 Annual General Donald R. Keith Memorial Capstone Conference. West Point, New York, USA, 30th Apr. 2020, p. 6.

[52] Barry W. Boehm. *Balancing agility and discipline: a guide for the perplexed*. In collab. with Richard Turner. Boston: Addison-Wesley, 2004. ISBN: 978-0-321-62388-1.

[53] Kilian Gerrike et al. "What do we need to say about a design method?" In: 21th International Conference on Engineering Design (ICED 2015). Vancouver, Canada, 2017.

[54] Lucienne T. M. Blessing et al. *DRM, a Design Research Methodology*. Springer Science & Business Media, 13th June 2009. 411 pp. ISBN: 978-1-84882-587-1.

[55] Robert K. Yin. *Case Study Research: Design and Methods*. 5th ed. SAGE Publications, 2014. 313 pp. ISBN: 978-1-4522-4256-9.

[56] *15th Annual State Of Agile Report | Digital.ai*. URL: https://digital.ai/resource-center/analyst-reports/state-of-agile-report (visited on 25/07/2022).

[57] *Status Quo (Scaled) Agile 2020*. Process-and-Project.net. URL: http://www.process-and-project.net/studien/studienunterseiten/status-quo-scaled-agile-2020-en/ (visited on 25/07/2022).

[58] Reinhard Haberfellner et al. "Agile Systems-Engineering versus Agile-Systems Engineering". In: *INCOSE International Symposium* 15 (1st July 2005). DOI: 10.1002/j.2334-5837.2005.tb00762.x.

[59] Bruce Douglass. *Agile Systems Engineering*. 1st edition. Morgan Kaufmann, 2015. ISBN: 978-0-12-802349-5.

[60] Markus Kohlbacher et al. "Do agile software development practices increase customer satisfaction in Systems Engineering projects?" In: (1st Apr. 2011). DOI: 10.1109/SYSCON.2011.5929091.

[61] M. Ann Garrison Darrin et al. "The Agile Manifesto, design thinking and systems engineering". In: 2017 Annual IEEE International Systems Conference (SysCon). ISSN: 2472-9647. Apr. 2017, pp. 1–5. DOI: 10.1109/SYSCON.2017.7934765.

[62] Torgeir Dingsøyr et al. "A decade of agile methodologies: Towards explaining agile software development". In: *Journal of Systems and Software*. Special Issue: Agile Development 85.6 (1st June 2012), pp. 1213–1221. ISSN: 0164-1212. DOI: 10.1016/j.jss.2012.02.033.

[63] Robin L. Dillon et al. "Faster-Better-Cheaper Projects: Too Much Risk or Overreaction to Perceived Failure?" In: *IEEE Transactions on Engineering Management* 62.2 (May 2015), pp. 141–149. ISSN: 1558-0040. DOI: 10.1109/TEM.2015.2404295.

[64] Evelyn Honoré-Livermore et al. "An Agile Systems Engineering Analysis of a University CubeSat Project Organization". In: *INCOSE International Symposium* 31.1 (2021), pp. 1334–1348. ISSN: 2334-5837. DOI: 10.1002/j.2334-5837.2021.00904.x.

[65] Nicholas Dallmann et al. "An Agile Space Paradigm and the Prometheus CubeSat System". In: *Small Satellite Conference* (11th Aug. 2015).

[66] Lucy Berthoud et al. "University CubeSat Project Management for Success". In: *33rd Annual AIAA/USUConference on Small Satellites SSC19-WKIII-07* (3rd Aug. 2019). Publisher: Utah State University.

[67] Mitch Bott et al. "An Analysis of Theories Supporting Agile Scrum and the Use of Scrum in Systems Engineering". In: *Engineering Management Journal* 32 (20th Sept. 2019), pp. 1–10. DOI: 10.1080/10429247.2019.1659701.

[68] David J. Barnhart et al. "A low-cost femtosatellite to enable distributed space missions". In: *Acta Astronautica* 64.11 (1st June 2009), pp. 1123–1143. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2009.01.025.

[69] Wayne Wolf et al. "Multiprocessor System-on-Chip (MPSoC) Technology". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.10 (Oct. 2008), pp. 1701–1713. ISSN: 1937-4151. DOI: 10.1109/TCAD.2008.923415.

[70] Jasper Bouwmeester et al. *PQ9 and CS14 Electrical and Mechanical Subsystem Interface Standard for PocketQubes and CubeSats*. DataverseNL, 4th July 2018. DOI: 10.34894/6MVBCZ.

[71] Nikoletta Triantafyllopoulou. *The QUBIK Project: Ready for orbit*. Libre Space Foundation. 19th Nov. 2020. URL: https://libre.space/2020/11/19/the-qubik-project-ready-for-orbit/ (visited on 30/08/2022).

[72] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Signature Series. Addison Wesley Professional, Pearson Education distributor, 2002. ISBN: 978-0-321-14653-3.

[73] Michael Fritz et al. "Hardware-in-the-loop environment for verification of a small satellite's on-board software". In: *Aerospace Science and Technology* 47 (1st Dec. 2015), pp. 388–395. ISSN: 1270-9638. DOI: 10.1016/j.ast.2015.09.020.

[74] Nicola Garzaniti et al. "Effectiveness of the Scrum Methodology for Agile Development of Space Hardware". In: 2019 IEEE Aerospace Conference. ISSN: 1095-323X. Mar. 2019, pp. 1–8. DOI: 10.1109/AERO.2019.8741892.

[75] Matthew Peterson et al. "When Worlds Collide — A comparative analysis of issues impeding adoption of Agile for hardware". In: *Proceedings of the Design Society* 1 (Aug. 2021). Publisher: Cambridge University Press, pp. 3451–3460. ISSN: 2732-527X. DOI: 10.1017/pds.2021.606.

[76] ECSS Secretariat. *ECSS-E-HB-10-02A – Verification guidelines*. European Space Agency, 17th Dec. 2020.

[77] Kent Beck. *Extreme programming explained: embrace change*. In collab. with Cynthia Andres. 2nd ed. The XP series. Boston: Addison-Wesley Professional, 2008. ISBN: 978-0-321-27865-4.

[78] Mojtaba Shahin et al. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". In: *IEEE Access* 5 (2017). Conference Name: IEEE Access, pp. 3909–3943. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2685629.

[79] Van Hunter Adams. "Theory and Applications of Gram-Scale Spacecraft". PhD thesis. Cornell University, May 2020. DOI: 10.7298/b0bt-8v62.

[80] Jian Tang et al. "An MBSE framework to support agile functional definition of an avionics system". In: *International Conference on Complex Systems Design & Management*. Springer, 2018, pp. 168–178. DOI: 10.1007/978-3-030-04209-7_14.

[81] Maxime Perrotin et al. "TASTE: A Real-Time Software Engineering Tool-Chain Overview, Status, and Future". In: *SDL 2011: Integrating System and Software Modeling*. Ed. by Iulian Ober et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 26–37. ISBN: 978-3-642-25264-8. DOI: 10.1007/978-3-642-25264-8_4.

[82] I. Bychkov et al. *Using Binary File Format Description Languages for Documenting, Parsing, and Verifying Raw Data in TAIGA Experiment*. 4th Dec. 2018. DOI: 10.48550/arXiv.1812.01324.

[83] Jerry de Vos et al. *Documentation of Open Hardware*. Delft Open Hardware Academy. 15th Aug. 2022. URL: https://hackmd.io/@Oggo2XIlRZ6wwlsXi_vc8Q/By3DNodtq (visited on 30/08/2022).

[84] Ken Schwaver et al. *The Definitive Guide to Scrum: The Rules of the Game*. 9th Nov. 2020.

[85] Paolo Ciancarini et al. "An Open Source Environment for an Agile Development Model". In: *Open Source Systems*. Ed. by Vladimir Ivanov et al. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2020, pp. 148–162. ISBN: 978-3-030-47240-5. DOI: 10.1007/978-3-030-47240-5_15.

[86] Luca Boggero et al. "An MBSE Architectural Framework for the Agile Definition of System Stakeholders, Needs and Requirements". In: *AIAA Aviation 2021 Forum*. 2021, p. 3076.

[87] A. M. Hein et al. "Evaluating engineering design methods: taking inspiration from software engineering and the health sciences". In: *Proceedings of the Design Society: DESIGN Conference* 1 (May 2020). Publisher: Cambridge University Press, pp. 1901–1910. ISSN: 2633-7762. DOI: 10.1017/dsd.2020.317.

[88] Arkadiusz Bulski et al. *Construct*. Jan. 2020.

[89] Robot Framework Foundation. *Robot Framework*.

[90] Dan North. *What's in a Story?* 11th Feb. 2007. URL: https://dannorth.net/whats-in-a-story/ (visited on 14/09/2022).