Software Product Assurance at the German Space Agency

Christian R. Prause*[1], Markus Bibus[1], Carsten Dietrich[1], Wolfgang Jobi[1]

[1]*DLR Raumfahrtmanagement, Königswinterer Straße 522-524, 53227 Bonn, Germany*

SUMMARY

The DLR Space Administration designs and implements the German space program. While project management rests with the agency, suppliers are contracted for building the space devices and their software. As opposed to many other domains, these are often unique devices with uncommon and custom-built peripherals. Its software is specifically developed for a single mission only and controls critical functionality. A small coding error can mean the loss of a mission. For this reason, customer and supplier collaborate to improve products and processes to achieve better software at the end of a project. We discuss quality in the context of space projects, and report a cross-sectional view from a customer's perspective on various management tools for influencing suppliers' processes and product quality: standards, single-source tailoring and cross-company product assurance. Copyright © 0000 John Wiley & Sons, Ltd.

Received ...

1. INTRODUCTION

The DLR is the national aeronautics and space research center of the Federal Republic of Germany. Its many research institutes are distributed over Germany and conduct research in the areas of transportation, energy, flight, security and spaceflight. The DLR is also Germany's space agency that — in addition to its own research — has been given responsibility by the government for the planning and implementation of the national space program. For this purpose, the DLR Space Administration acts as customer and project manager for its space missions during the making of space devices, while contractors perform the actual work of making. It commissions devices (e.g., spacecraft) for its missions from a diverse range of suppliers including industrial and academic partners. It invites tenders, awards contracts for projects, supervises them afterward, and promotes innovative ideas in research and industry.

Other European nations have their own space agencies as well. The different agencies sometimes collaborate with each other in projects, however, they remain independent entities with their own space programs. Furthermore, there is the ESA, which is the European Space Agency. It is an international organization with currently 22 member states including Germany. DLR and ESA collaborate closely, and ESA committees include DLR representatives. Yet, they are separate organizations, both doing their own projects, having their own research divisions and mission operations, and often procuring devices by external entities.

The space domain is peculiar with respect to the fact that many spacecraft are one of a kind devices with uncommon and custom-built hardware and software. As opposed to consumer products,

*Correspondence to: Christian Prause, DLR Raumfahrtmanagement, Königswinterer Straße 522-524, 53227 Bonn, Germany. E-mail: christian.prause@dlr.de

scientific space missions are expensive, one-of-a-kind devices with high technical uncertainty. For most missions, a second unit is never built. If the mission goal is not reached, for whatever reason, there is no second chance. Preparing a mission and subsequent production of the spacecraft can take decades. While software can be updated in flight, it often controls critical functionality. A single failure can mean the loss of a spacecraft and its mission, for example: Ariane Flight 501 [1] or Mars Climate Orbiter [2]. Additionally, due to limited contact times with ground stations, uploading new software versions can take days. Therefore very high quality of the software is needed, also justifying higher efforts in avoiding coding errors [3, 4].

When the NASA (United States' National Aeronautics and Space Administration) started its "Faster, Better, Cheaper" initiative in the early 1990s, the goal was to enable more and parallel projects. The program capped maximum project cost, simplified organizational structures, reduced the workforce by a third, and transferred day-to-day space operations to a single private contractor, losing an inherent quality assurance checks and balances system. Also lost were critical engineering and management skills through reduction-in-force layoffs and when experienced personnel took advantage of early retirement offers [5]. The program also put a high cost pressure on projects. At the same time, software became important as a sponge for complexity, to grant autonomy to free-flying devices, and as "band-aid" for hardware design compromises that allow reducing hardware costs [6, 3, 7]. When NASA missions like the Mars Climate Orbiter and Polar Lander failed [3], and the Columbia shuttle was destroyed [5], it became clear that processes had to be reconsidered. In Germany, the Space Administration reacted to similar experiences by significantly increasing dedication to and efforts in quality assurance activities [8, 9]. Its *product assurance department* became ingrained in the management processes and has since been responsible for quality management support in all major projects.

This article makes a cross-section through the principal management tools that are employed for ensuring product quality from the customer perspective, and for managing the evolution of supplier software processes from a customer's perspective and on customer initiative. The following sections are structured as follows: First, an introduction is given to the role of product assurance, and software quality is discussed in the context of space flight to further motivate product assurance activities (see Section 2). The basis of product assurance work is a national catalog of process requirements derived from several standards, first and foremost the *European Cooperation for Space Standardization* (ECSS) standards (see Section 3). The national catalog respects the historical, legal and cultural peculiarities of the German space program, while converging towards the ECSS standards (see Section 4). These requirements are further *tailored* to each individual project, anchoring product assurance work in project execution (see Section 5). Continuous customer (i.e., DLR) involvement in the process of making is necessary to provide confidence that goals will be reached effectively and efficiently, and to detect potential problems as early as possible. The Space Administration's product assurance therefore stays in close contact with project management, the supplier and the supplier's product assurance to ensure that development processes fulfill the requirements and are suitable to ensure the quality of the product (see Section 6). Section 7 gives an experience report on how advanced static analysis for critical software was introduced in German missions. In particular, this had to happen against the initial resistance of industrial suppliers, and was only made possible by exploiting the above management tools. Section 8 reports on related work, Section 9 summarizes lessons learned, and Section 10 concludes this article.

2. SOFTWARE PRODUCT ASSURANCE AND COST OF QUALITY

Product assurance is one of three primary project functions in a space project. The other two are *project management* and *engineering*. Project management is concerned with the more non-technical management of the project like project planning, resource allocation, and budgets. A colleague once said about the difference between engineering and product assurance: *The task of engineering is to make it work once, while the task of product assurance is to make it work all the time.*

Product assurance is a management discipline that assumes the customers' viewpoint on product quality. It supports project management in steering the product life cycle and controlling production according to technical and programmatic requirements, while building on experience and lessons learned. Software product assurance disciplines include quality assurance with sub-ordinate quality control, safety and dependability assurance, project planning, (independent) validation and verification, testing and evaluation, configuration management, and software measurement. Its functions are to observe, witness tests, analyze and recommend, but not to develop or test, manage people or set product requirements. Instead, it has organizational, budgetary and product developmental independence meaning that it reports to highest management only, has its own budget, and does not expend labor to help building a product [10, 11, 12].

It is important to note that in a typical space project, the customer and the seller both have personnel in place that fulfills the role of product assurance, and both maintain direct communication with each other. The customer's product assurance mirrors the sellers' own product assurance function, acting as reviewer of contractors and technology providers. It assesses organizations and how they perform development activities in order to obtain software products that are fit for use and built in accordance with applicable project requirements [13]. At the DLR Space Administration, product assurance accompanies technical processes in order to ensure product quality and successful completion of the project. It acts as an external reviewer for contractors and technology providers ranging from large companies to small and medium enterprises, research institutes and universities. While being smaller, it is comparable to NASA's *Software Assurance Technology Center* (SATC, cf. [14]). A key aspect to achieving high quality software products are the suppliers' development processes. It is hence necessary to influence suppliers to improve their processes, and to continuously evolve methods and tools of software quality assurance for the future.

2.1. The Rise of On-Board Software

As opposed to most commercial software available for general business, software developed for technical purposes in the space domain is intended for specific applications with special needs and characteristics [15]. In general, there are four types of such software: pre-pass, real-time, post-pass and on-board software. The first three types are associated with the so-called *ground segment*, while the fourth is part of the *space segment*. Both segments form a spacecraft system.

Pre-pass software is run on the ground before a spacecraft rises over the horizon of a ground station. Typical tasks of such software are orbit determination and prediction, planning and scheduling, command list generation and simulation. *Real-time* software is operated while the spacecraft is visible from the ground station. It includes controlling of the antenna tracking, command uplink and verification, data reception, and status checking. *Post-pass* software operates after a pass, when the spacecraft is no longer visible and its data has been downloaded. Tasks include health assessments of the device, data processing and data analysis. Finally, the fourth type of software, *on-board* or *flight* software is executed on-board the spacecraft. It is responsible for telemetry and telecommand, on-board data handling and processing, fault recovery, and all the like.

In the 1970s, software started to become "big business", when the ratio between hardware and software expenditures first started tipping towards software. NASA software expenditures grew to an amount twice as high as hardware expenditures [16]. For scientific space missions, the ratio between costs for hardware and software has since then changed from 10:1 to 1:2. Software has also become an important design tool for cutting on hardware costs [7]. The complexity of on-board software has increased similarly: In the automotive domain, electronics are already accountable for 90% of the innovations, and 80% out of that in the domain of software [17]. In military aircraft, the percentage of functionality provided through software grew from 8% to 80%: The F-4A had 1000 lines of source code, the F-22 already had 1.7 million lines, and the F-34 is estimated to have almost 6 million lines of code [6]. Regarding the on-board software for spacecraft, software today has not yet achieved a similarly high share of the total cost. But its importance is growing, too: Experts gauge that flight software costs can be up to almost 20% or even 30% of spacecraft costs. In ESA projects, the 10% mark was surpassed about ten years ago (cf. [18, 15]).

The reasons why the rise of space flight software did not keep up with that in other domains are serious limitations put on flight computing hardware through weight limits, smaller markets, long development times, reliability needs and, in particular, radiation hardness requirements. In the ground segment, where modern computing equipment can be used, software development makes up for over 40% of development costs [15]. But flight software is not only limited through the availability of computing hardware, it is embedded software with some hard real-time functions that has a much higher criticality than ground software. As a consequence, developing ground software costs about three hours per line of code (including requirements analysis, design, documentation, and verification and validation), resulting in an approximate price of over 200 EUR per line of code, whereas developing flight software costs twice as much, about six hours or more than 400 EUR per line (numbers for Ada code; C code is considered to be more than 50% more expensive per line of code.) [18].

Product assurance work at the Space Administration and this article focus on flight software because of its higher criticality. Depending on the kind of device and mission in which the software is to be used, reducing the potential risks to the device and mission are much more important than costs. From a cost perspective, however, focusing on ground software might be more efficient because its share of development costs of its respective system segment is four times higher.

2.2. The Cost of Software Quality

In order to explain the cost difference between ground and flight software, Figure 1 depicts positive and negative quality costs, and shows where their respective sweet spot for quality cost is (cf. [19]). The falling dashed line shows the negative quality costs, i.e., the costs (taking into account expectancy values from potential risks) that are caused by too low a quality level. A lower software quality level leads to higher risks of unfavorable and fatal events due to software failures, and therefore to higher expected losses. (Lower quality levels not only lead to the mentioned external failure costs, but also to internal failure costs from scrap, rework, failure analysis, etc. but these are less relevant in the current discussion.) With more efforts put into achieving a higher quality level, the costs through risk can be reduced. However, the cost function is not linear; small improvements in risk reduction require ever higher quality levels. It is practically impossible to achieve zero risk.

The rising dashed line shows the positive quality costs, i.e., the costs that are caused by the measures that are leading to higher software quality levels. Positive costs are prevention and appraisal costs. Prevention measures, for example, are quality planning and assurance, and personnel training. Appraisal measures include audits, testing, verification, and supplier assessment.

Both costs together add up to the *total cost of quality* (black curve above the two dashed ones). This discussion of costs, of course, is rather theoretical, because reliable and quantifiable values for positive as well as negative costs rarely exist. The cost of individual quality level improvement measures is difficult to determine, as well as their benefit or resulting risk reductions. Likewise, for example, the loss of a mission in terms of missed scientific chances, delays, bad reputation, the potential loss of life, or even the long-term damage of radioactive material released into earth's atmosphere through a potential accident (e.g., NASA Cassini controversy) are hard to estimate. Furthermore, the software-related costs have to be considered within the context of their larger hardware-software system. Still, because both functions are non-linear, there is a point where the sum of both costs is minimal. This is the sweet spot of software quality.

The topmost, red curve shows the total cost of quality for an adjusted negative quality curve for worse risk costs (corresponding negative cost curve not shown). Regardless of the quality level, critical software is generally more expensive than non-critical software. But furthermore, the critical software sweet spot is achieved at a higher quality level. The main reason why flight software is more expensive in practice than ground software is that higher efforts in its quality are actually cheaper. The tailoring activities described in this article rely on heuristics and experience to assess how much effort should be put into achieving a quality level that is adequate for the respective project. It tries to balance positive against negative quality costs (cf. [20]).

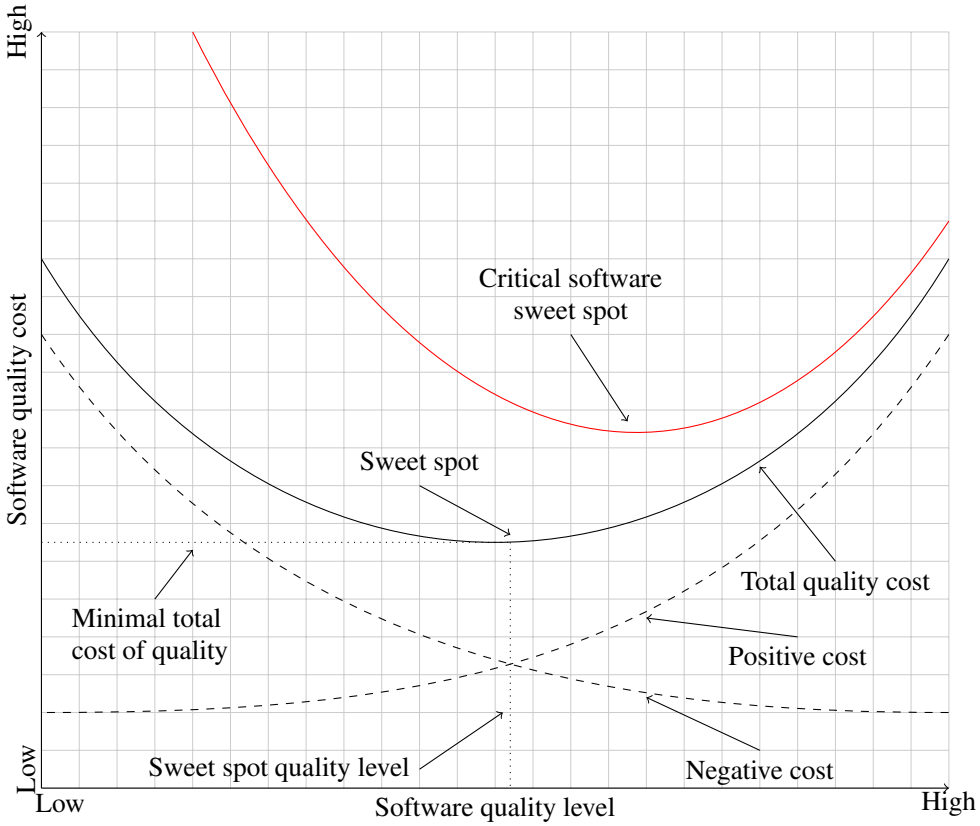

Figure 1. Software quality level and cost of quality adapted (cf. [19, 20])

2.3. Economic Considerations regarding Software Product Assurance

Software product assurance deals with the quality of the product, and improvement of the development processes. Let us look at two grounded but fictive example cases for quality cost considerations with respect to software product assurance. The examples here rely on publicly available data only.

An important aspect in space projects is risk minimization. Although estimating risks is difficult because risk probabilities are mostly unobservable, we could make the simplified assumption that if 10% of space system cost are spent on software, then also 10% of the risks arise from software. That means that one out of ten mission failures would be caused by software. Actually, the number is probably higher because many critical functions are assumed by software. The success rates for space missions are somewhere between 60% (missions to Mars, 23 of 43 failed or were only partially successful) and 95% (for launchers) [21]. The success rate for a one-of-a-kind, first-time-ever scientific mission could therefore be guessed to be 80%. Hence, 20% of the missions fail. A fictive major mission (like [22]) may have a cost of about 200 million USD. This assumption is based on publicly available project data from one of currently several major missions. The project cost was estimated with ≈ 150 million EUR (year 2016 equivalent of 140 million EUR in fiscal-year-2009-Euros), translating to 170 million USD. As it was due to start operating nominally in 2013 but was not yet launched, [22] some cost increment is plausible.

The risk costs of a major mission failure due to software are then 10% (software risk) $\times$ 20% (science mission risk) $\times$ 200 million USD (mission cost), or 4 million USD software risk cost. Reducing the risk of fatal software failures by 10% means saving 400.000 USD per major mission.

As the second example, we look at gains from software process improvements in general. The total German expenditures in the national space program are about one and a half billion USD.

However, almost half of that amount is Germany's contribution to the ESA [23]. So the upper bound for development costs is 750 million USD. Major missions are (an important) part of the national space program. A mission's system consists of space, ground and launch segment; i.e., the actual satellite is only a part of the whole project, however, it is by far the most expensive part (cf. [18]). There are currently a few development projects for major missions running in parallel. If the typical development project takes five years (e.g., [22]), this would statistically equal about one major mission every one to two years, or about 100 to 200 million USD per year (lower bound). From the upper and lower bound, we therefore presume here that the annual budget for space segments of major and smaller missions is 250 million USD per year.

As introduced earlier, a conservative assumption is that 10% of the space segment costs are software development costs. The annual expenditures in flight software development activities are therefore 25 million USD. An increase in development effectiveness through improved processes (e.g., improved validation and verification activities, or less scrap or rework) of 1% means savings of 250.000 USD per year. Gains from process improvements of more than one percent do not seem obnoxious because, firstly, the national space market is rather small but complex, limiting the effects of open competition on the need for productivity optimization and, secondly, because some work is also done by research organizations and small enterprises that often do not invest very much in their process maturity (cf. [24]). Basili et al. [14] report that over the time of several years, through their efforts in the domain of software engineering and process improvement in the NASA, they achieved improvements of 15%, which would translate to an annual 3.75 million USD savings.

These two very much simplified examples do not take into account indirect costs, follow-up costs, the costs of schedule overruns, and much more. Furthermore, they are based on conservative assumptions about the role of software in space projects, and hence rather represent lower bounds for benefits from process and product quality improvements. Other positive side-effects like knowledge gains are also not considered. Since software engineering is a rather young scientific field that is quickly evolving, the described improvements in process improvements and risk reduction should be possible to realize. It should also be noted that while several spectacular mission failures have occurred due to software defects over the years, many more mission losses were averted through post-launch work-arounds that could only be implemented in software [25].

3. THE ECSS STANDARDS

Many organizations are usually cooperating in the production of a space device. They are bound by legal contracts in the roles of customer, and suppliers which in turn act as customers to their lower-tier suppliers. The ECSS is a system of standards that European space agencies and industry have given themselves to put their work on a more rigorous basis. While the ECSS standards have no legal standing by themselves, they are made applicable by invoking them in the business agreements [26]. They provide a collection of what we call *process requirements* here (another term would be *software standards*). These are not to be confused with product requirements that describe what the product should do.

3.1. Inception of the ECSS

The space flight domain is signified by a high complexity in various dimensions. Technical difficulties arise from the fact that every device is a unique specimen, harsh conditions in which devices have to operate and the difficult access to devices in flight. Hardware cannot be tested in its real target environment before but the environment can only be simulated. It is exposed to extreme physical conditions like the structural stress of launches, steep temperature changes, high-energy radiation and solar storms, vacuum, and particles and space debris, while hardware is almost impossible to maintain once launched. The software is developed for a single mission and as a controller of a unique hardware configuration. It must be capable of dealing with unfavorable events like particle impacts and evasive maneuvers, or cope with degradation of hardware functionality by detecting, isolating and recovering from (potentially permanent) hardware

failures. At the same time, it must for example, always keep damageable hardware at acceptable temperatures through commanding temperature control systems, use fuel sparingly, maintain flight orbits, manage attitudes to support communication links, heat input or solar panel utilization, avoid space debris, conduct energy management, and control diverse scientific payload. Particular managerial difficulties result from very long lasting projects, complex tasks and the need to coordinate diverse partners including industry, research organizations, national and transnational agencies from different countries in a wide range of different contractual relationships including commercial procurement contracts, science grants, or co-operations mandated by politics.

In order to mitigate difficulties, the ESA developed standards to be applied in their projects. But also national agencies and the commercial sector developed own standards. The individual standards did not achieve legislative authority. Consequently, the use of standards had to be negotiated individually in projects, and their use could differ strongly between two projects. However, the need for cooperation between the different entities made this approach ever more ineffective. When the ESA recognized this, they founded the European Cooperation for Space Standardization (ECSS) in 1993 to create a single and coherent set of space standards [27]. For the development and enhancement of the standards, the major players in the European space sector — agencies and industry — collaborate. The ECSS aims to improve development processes with the objectives to:

- cut costs,
- increase quality,
- reduce risks,
- improve competitiveness,
- enhance safety and reliability,
- improve collaboration, and
- develop and disseminate fresh knowledge [28].

The standards are made available for free[†] to promote their wider usage [28]. Several standards have also been officially passed on to other countries including Japan, Ukraine, and Kazakhstan for use in their own projects. The international collaboration of the ECSS is expanding, for example, through collaboration with the international and European standardization organizations ISO and CEN.

3.2. Organization of the standards

Most ECSS standards documents follow a systematic naming approach [26]:

```
ECSS-[branch]-[type]-[major][-minor][version]
```

The basic structure of the ECSS standards system is divided into five branches that denote the thematic area (or *branch*) of the standard: The first branch (ECSS-P and ECSS-S standards) deals with the ECSS itself and describes how standards are developed, contains a glossary of aerospace terms, and the like. ECSS-M standards address management topics. Standards in of the ECSS-U branch address sustainability like space debris or protecting alien planets from terrestrial life forms. The topic of ECSS-E and ECSS-Q standards is engineering, and product assurance, respectively.

In each branch there are several top-level standards (denoted by a number that is a multiple of 10) that address a specific *discipline*. Examples of disciplines in the engineering are *system engineering* (ECSS-E-ST-*10*) and *software engineering* (ECSS-E-ST-*40*). Likewise, the Q branch contains disciplines like *dependability* (ECSS-Q-ST-*30*), *safety* (ECSS-Q-ST-*40*), or *software product assurance* (ECSS-Q-ST-*80*).

Each discipline has at least its main standard that is of normative nature, i.e., it contains prescriptive information on how to comply. The standards document — signified by the *type* "ST" in its name — describes a set of requirements to be applied to processes. These requirements focus

[†]Visit `http://www.ecss.nl/` for online access to the standards.

on product aspects and primarily on what has to be accomplished (e.g., 100% test coverage) rather than on how to organize and perform the work [26].

Additionally, there may also be sub-ordinate standards or handbooks (which have a type "HB" instead of type "ST" designation). Sub-ordinate standards further detail certain aspects of a discipline. Handbooks are of an informative, descriptive nature and written as prose texts. They explain a standard, provide interpretive help and explain the intentions behind the requirements in the standard. For example, in the software product assurance discipline, ECSS-Q-HB-80-01A addresses the reuse of existing software.

Every document usually has a revision letter appended to it, starting with "A". After a major revision to the document, the next letter from the alphabet is appended instead. The current major revision of standards is "C". In ECSS nomenclature, the revision is also called *issue*.

The ECSS standardization body is organized into several working levels. ECSS objectives, policy, strategy are defined by the ECSS *Steering Board* at the top which also endorses the yearly work plan. Below the Steering Board operates the *Technical Authority* (TA) that puts the the objectives, policy and strategy into action by setting up, approving, implementing and monitoring the work plan. *Working Groups* perform the actual writing and maintaining of standards. *Networks of Experts* are on the lowest level and represent focal points for disciplines to provide support with technical expertise. The *Executive Secretariat* supports the TA and Working Groups in doing their work by ensuring that drawting rules are followed, by support on administrative issues, and as an interface to other standards organizations [27].

3.3. Software Standards in the ECSS System

The principle requirements that are applicable to software development for space applications are laid out in ECSS-E-ST-40 (*Space engineering – Software*), which is a specific adaptation of ISO/IEC 12207 and gives consideration to standards like the ISO 9000 family, EN 50128 for railway applications, and the DO-178 standard for airborne systems and equipment. The life cycle elements covered are requirements engineering, architectural design, software design, implementation, validation and verification, delivery and acceptance, operation, maintenance, software system engineering, and the management of software project. The standard also applies to non-deliverable software which affects the quality of products and services.

It has a historically close relationship with the software product assurance standard ECSS-Q-ST-80C [29]. Both standards used to share many aspects and still address the full spectrum of software-activities for space devices: launchers, manned and unmanned spacecraft, platforms, payloads, experiments and associated ground equipment and facilities, and services implemented by software. Their interrelation and partial duplication, however, was resolved with issue "C" of the standards, which draws a clearer border between both standards. Since then, ECSS-Q-ST-80C focuses more on quality assurance activities, auditing, problem resolution, infrastructure aspects, training and software process improvement. It interfaces with space engineering and management branches of the ECSS, and explains how they relate to product assurance processes.

ECSS-Q-ST-80C is supplemented by five handbooks: ECSS-Q-HB-80-01A addresses the *Reuse of Existing Software*. The two volumes *Framework* and *Assessor Instrument* of the ECSS-Q-HB-80-02A *Software Process Assessment and Improvement* are known as *SPiCE for Space*, a software process maturity model derived from the SPiCE (Software Process Capability dEtermination) framework based on ISO/IEC 15504. Requirements regarding *Software Dependability and Safety* are further explained in ECSS-Q-HB-80-03A because dependability and safety are issues of paramount importance in the development and operations of space systems [30]. Finally, guidelines for *Software Metrication Programme Definition and Implementation* are provided through ECSS-Q-HB-80-04A [31].

The ECSS-E-HB-40A *Software Engineering Handbook* motivates the use of ECSS-E-ST-40 and its requirements. It provides advice, interpretations, elaborations and best practices for its implementation.

Recently, the ECSS started to work on a handbook for agile development. In industry, there is a growing interest in agile life cycle models like Scrum and they promoted the idea of an agile

standard. However, the ECSS technical authority did not want to potentially contradict any of the existing and proven standards with a new standard that promotes a new development model that was not yet proven to work for space applications. Therefore, the ECSS opted to let a working group create a document of type handbook instead. In case of a potential conflict, the requirements from the existing standards would hold untouched in any project.

In addition to these software-specific standards, there are many more that also have an influence on how software developed. For instance, ECSS-M-ST-40C defines requirements for configuration and information management that are applicable to software configuration management as well. Other such life cycle processes are the processes of acquisition, supply, documentation, or more generic management requirements. It is common for ECSS standards to make other standards applicable.

Software is an element that nowadays pervades any space program and its product tree (see also 2). A short introduction to software development according to ECSS can be found in [32].

4. NATIONAL REQUIREMENTS CATALOG

The development and manufacturing of space systems in Europe has been influenced by the cooperation of different national space agencies, the ESA, and industry since the beginning. However, European countries also maintain their own space activities.

Traditionally, the agencies developed their own standards independently and applied them to their projects. Yet rising demands for collaboration between the different stakeholders led to the founding of the ECSS (see Section 3). While its goal is to harmonize existing standards for space projects, and to develop and maintain a single, coherent set of standards to continually improve the quality, reliability, functional integrity and compatibility of all project elements, this process is still ongoing. International harmonization is not yet fully reached. A sudden transition from the traditional ways of working to ECSS cannot be easily implemented. Therefore, a national product assurance requirements catalog that is suited to its needs is maintained by the DLR.

4.1. Pre-tailoring ECSS into the National Catalog

With *pre-tailoring* we denote a process that takes the ECSS standards to turn them into a single coherent set for use in the German space program. This pre-tailoring is necessary to adapt the complex system of standards to national interests (e.g., no manned flight), agency policies, and established practices in national industry. The process is analogous to ECSS-S-ST-00C [26] tailoring.

The first step is to extract distinguishing characteristics of the national space program. Relying on experience, we selected the eight characteristics *Space Flight Type* C_{Sft} (e.g., satellite, lander), *Launcher Type* C_{Lt} (e.g., expendable, reusable), *Utilization* C_U (e.g., free-flyer, ISS internal), *Objective* C_O (spacecraft, payload), *Maintainability Requirements* level C_{Mr}, *Environment Conditions* C_{Ec} (atmosphere, space), *Safety Requirements* level C_{Sr}, and *Manned Flight Authority* C_{Mfa} (ESA, NASA, Roscosmos).

In Step 2, these characteristics are complemented with refinement analyzes regarding *Lifetime* R_L, *Budget* class R_B, *Complexity* R_C, *Technology Risk* R_{Tr}, and *Risk Policy* R_{Rp} levels (high, low). Characteristics plus refinements form the dimensions of a nominal scale vector space for projects p:

$$p \in C = C_{\{Sft,Lt,U,O,Mr,Sr,Mfa\}} \times R_{\{L,B,C,Tr,Rp\}}$$

Step 3 is to identify standards relevant for software quality. First and foremost, this is the ECSS software product assurance standard ECSS-Q-ST-80C [29] and parts of its complementary software engineering standard ECSS-E-ST-40C [33]. However, also requirements from other standards are integrated, for instance, configuration management (ECSS-M-ST-40C) or non-conformance reporting (ECSS-Q-ST-10C).

In Step 4, every requirement is checked for its applicability to the national space program. For example, several requirements regarding software maintenance are out of scope of the Space Administration's product assurance.

Step 5 allows to add requirements that are not part of ECSS. For example, NASA or Roskosmos standards have to be obeyed for ISS (International Space Station) missions as these are the safety authorities. This step also allows to introduce further processes (cf. Section 7).

Step 6: After requirements have been gathered from the relevant standards, they are checked for internal coherence and consistency. Furthermore, it is necessary for later tailoring to discriminate more or less demanding and expensive process requirements. Therefore, one or more requirement level tags are assigned to each requirement. Requirements imposed on software-specific processes are classified with one of four levels $w \in W_? = \{W_1, ..., W_4\}$ where higher level requirements are less demanding or expensive, and are included in all lower levels: $W_1 \subseteq W_2 \subseteq W_3 \subseteq W_4$. For example, Independent Software Verification and Validation (ISVV) by a third party is at level W_1 due to its high cost. As opposed to this, software configuration management (W_4) is considered basic engineering rigor. Cross-domain requirements from generic quality management and safety rely on an analogous scheme (using four Q and three S levels, respectively) and further influence the software process requirements.

Finally, as Step 7, the results are documented in the product assurance requirements catalog [34]. The requirements are stored alongside their classifications in a database for automated tailoring (see Section 5).

4.2. Industry Expert Group

Step 6 of pre-tailoring is very work intensive and requires deep knowledge of the various requirements in order to assess their benefits, costs, effects and cross-relations. Decisions made here severely influence the cost of later projects, and the efforts of suppliers. This work is therefore done by a group of experts, which is convened every few years to update the requirements catalog. The expert group officially consists of one representative of every major stakeholder in the national program, i.e., from Space Administration and national industry. These formal representatives commonly have a *Head of quality*-role or equivalent in their organization, and are supported by own domain/software experts.

The expert group starts its work by assigning so-called *field captains* to thematic subsets of the requirements. The field captains first write a short assessment for each requirement in their responsibility. They see the item's ID, source, title, descriptive text, and requirement level tags ($Q_?$, $W_?$, $S_?$), and take into account aspects like cross-relations (duplications, contradictions, ...) or practical impact. They attach their comment to the requirement and propose a resolution of *accept*, *reject* or *modify*, i.e., to either accept the requirement for inclusion in the catalog, to reject it, or to accept it with the modifications given in the comment.

Next, all representatives cast their vote on the proposed resolution (*accept* or *reject*) and may add a comment. If at least one representative rejects the resolution, the field captain has to make a new proposal, which consists of an explanatory comment and a revised resolution. The representatives now again vote with *accept* or *reject*. The resolution is accepted if it is supported by the majority, otherwise final resolution is postponed for a round table discussion.

The expert group meets physically once when it constitutes and once it finishes. The work between the two meetings is supported by a web-based tool (see Figure 2) that lets experts handle all reviewing and voting activities asynchronously, and maintains the database that forms the catalog of requirements. In the end, a document version of the catalog is printed as book and signed off (cf. [34]) by all representatives to support its symbolic value.

5. PER-PROJECT TAILORING OF PRODUCT ASSURANCE REQUIREMENTS

The ECSS is a system of coherent standards that supports a wide range of diverse space projects. In its original form, it might therefore not yet suit the individual project very much. This can result

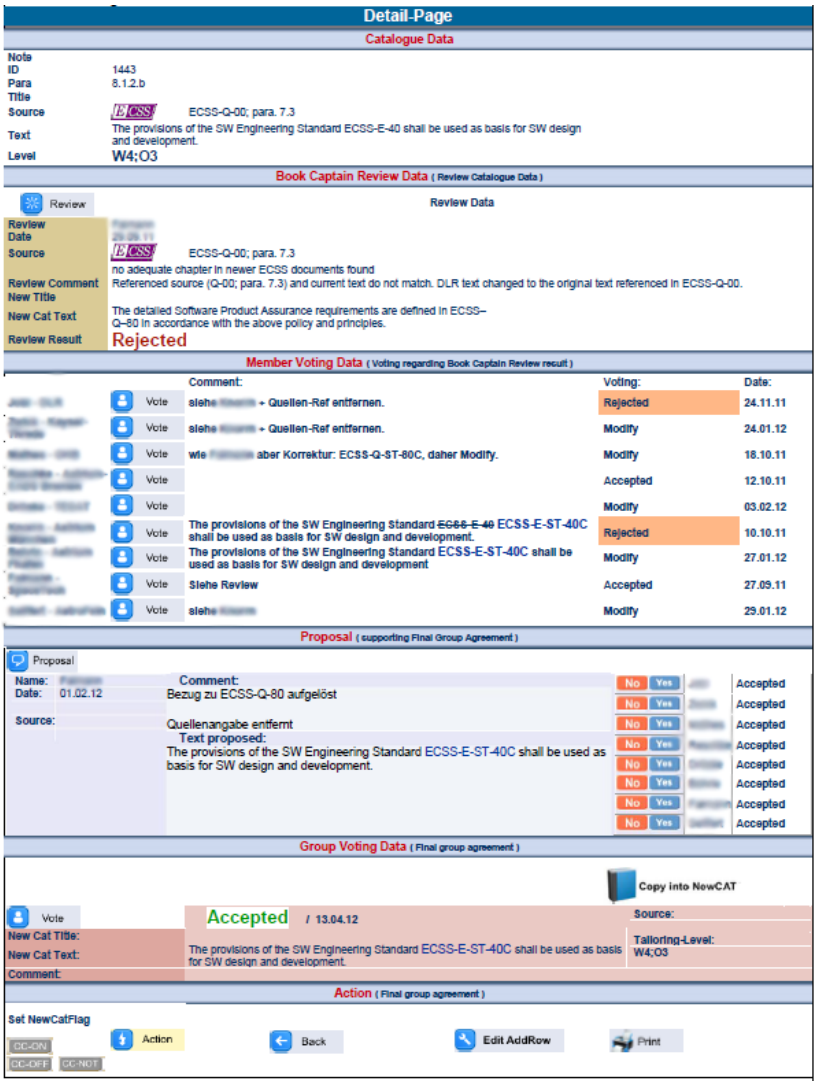

Figure 2. Screenshot of industry expert group discussion about a requirement in the web-based tool.

in reduced project performance in terms of technical performance, life cycle cost-effectiveness, or timeliness of deliveries. *Tailoring* is the process of fitting requirements placed on the process to the specifics of individual projects [26].

The basis for tailoring is the national catalog of product assurance requirements. For tailoring, the three functions $f_W : C \to W_?$, $f_Q : C \to Q_?$ and $f_S : C \to S_?$ process the project vector $p \in C$ in order to obtain the applicable requirement levels. The requirement levels then select or deselect the individual requirements, resulting in the set of requirements applicable to the software development process. This tailoring is, for the most part, automated through a software tool, which interactively prompts the project's basic characteristics and complementing analysis from the user. The selections made here lead to the applicable requirement levels for $W_?$, $Q_?$ and $S_?$ (see Figure 3).

The tailoring tool then automatically picks the requirements for inclusion in the product assurance requirements document that becomes part of the procurement contract for the space device. As described before, all requirements in the national catalog were tagged during pre-tailoring with one or more requirement level tags. For example, the requirement that a hardware-software interaction analysis should be conducted, is tagged with S_2 and W_3. It means that the requirement is included if

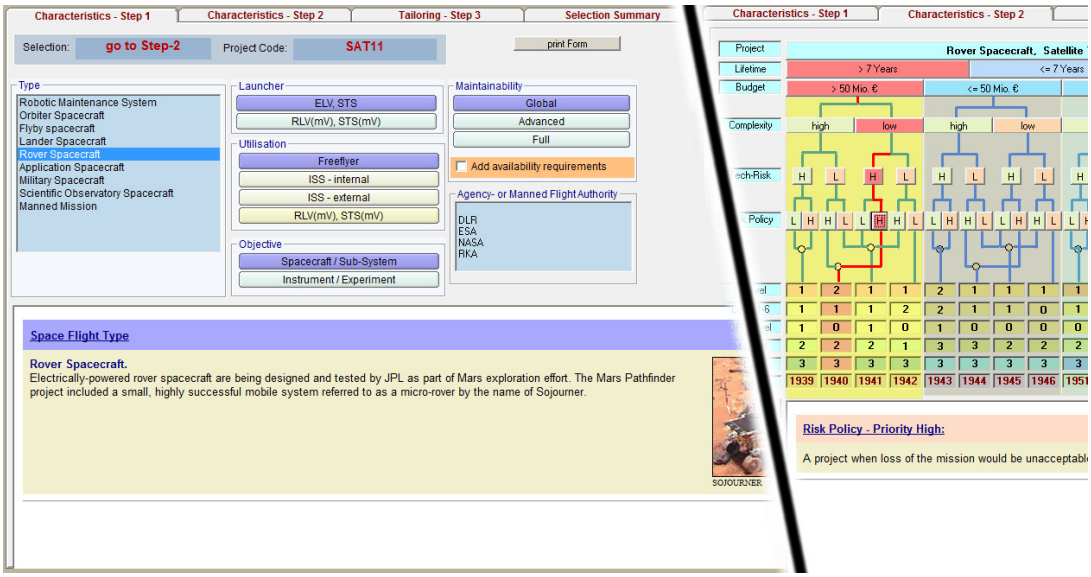

Figure 3. Screenshot of tailoring Step 1 and Step 2 in the QMExpert application. Left part: Initial tailoring parameters. Right part: Refinement parameters highlight path in red to resulting requirements levels.

the safety level is $s \leq 2$ or if the software level is $w \leq 3$, which is in all applications that are rather safety critical and in all applications where software has an important role.

Next, the user is presented with a preview, and an overview of which requirements were selected and deselected. He can select additional requirements, or deselect requirements that he wants to be removed for the project. The tailoring tool also ensures consistency by making sure that referenced requirements that are not themselves selected are included as well. Finally, the tool exports to a Word document for further processing, e.g., inclusion in contractual documents. Front matters, table of contents, abbreviation lists, indexes, chapters and the like are automatically generated.

6. PRODUCT ASSURANCE PROGRAM

The Space Administration does not itself perform engineering and manufacturing of space devices. Instead, it acts as customer and project manager. For ensuring product quality and successful project completion, technical controlling is necessary. Through it, customer and supplier seek *visibility* for mitigating high risks of, for instance, untested technologies, large sums of money, or loss of life. The two ways to achieve visibility are (i) to negotiate contracts with intermediate products and partial payments, and (ii) to increase customer involvement in the development process [11]:

Firstly, missions are divided into several phases from 0, A to F, e.g., early concepts and feasibility studies (Phases 0 and A), operation (E) and disposal (F). Each of the phases includes end milestones in the form of project reviews, the outcome of which determine payments and readiness of the project to move forward to the next phase. The reviews are main interaction points and provide the intermediate products. However, the largest portion of costs is spent in the design and production phases B to D, which, for practical reasons, are hard to subdivide, and so are often conducted in one piece. Secondly then, tight customer involvement is essential. All three disciplines of project management, engineering and product assurance are therefore mirrored on the customer side as interfaces to supplier counter-parts.

Product assurance plays the role of a devil's advocate in a constructive and non-confrontational way. It benevolently probes the product, has organizational and budgetary independence, and helps shaping but not building the product [11]. It interacts with supplier-side quality functions, and primarily with quality improvement in order to trigger improvements in the suppliers' development processes. Yet, this interaction is not a one-way street: feedback, experiences, and knowledge

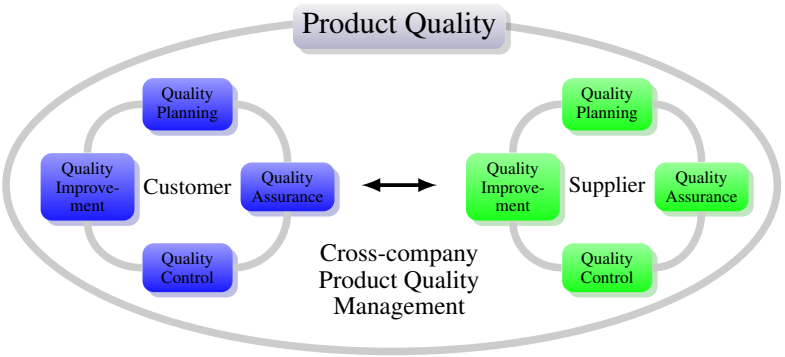

Figure 4. Cross-company product quality assurance

generated from projects lead to improved product assurance also on the customer's side (Figure 4). As a member of ECSS, DLR forwards knowledge further upstream so that it may eventually find its way into standards. Further, the toolbox of processes, methods and tools for product assurance is continuously evolved.

Members of the customer product assurance are involved in project activities from the beginning: As part of the contractual negotiations, the supplier has to state his compliance to the tailored process requirements. The *statement of compliance* is a matrix indicating for each requirement either *fully compliant*, *partially compliant*, *non-compliant*, or *not applicable*. During the project, the supplier adapts its processes in order to comply with the agreed requirements.

For example, there is the general requirement of having established product assurance functions. While having not done so has to be considered as a serious threat to the capabilities of a supplier [11], small companies, universities and research institutes are often missing product assurance. So, one of the first improvements is to promote the establishment of product assurance.

Our requirements prescribe what should be achieved, not how. The actual implementation is proposed by suppliers in respective plans, e.g., Software Product Assurance Plan, and is reviewed at milestones. The plans serve to improve the visibility of the supplier's work, help us gather lessons learned, and are proofs of the implementation of requested processes. Besides milestone reviews, customer product assurance attends progress meetings, looks out for deviations, and defines the actions necessary to reach compliance. While much work is paper-based, the customer retains the right to visit a supplier's facilities any time and to perform inspections of work products. In case of non-conformances, product assurance participates in decisions about further measures like root cause analyzes. If a deviation's root cause is found to be in the supplier's processes, the deficiency is to be eliminated in the frame of process improvements.

7. EXPERIENCE REPORT: POLYSPACE

Static analysis is widely used for detecting potential problems in software by analyzing it without executing it. In contrast to simpler static analysis methods, tools based on *abstract interpretation* (called *sound* static analysis) can prove the absence of several run-time errors (e.g., division by zero, arithmetic overflows). The advertised benefits are obvious, but overcoming problems with its adoption was only made possible through the previously described management tools.

7.1. Gathering First Experiences

One of the first commercially available tools for abstract interpretation was Polyspace. Its promise to prove the absence of errors clearly made it a candidate for improving verification processes. In order to try out the capabilities of abstract interpretation, we set up pilot project. For analysis, 22,000 lines of fully validated and verified spacecraft code were provided voluntarily by one supplier. Polyspace

marks every line as either dead code, proven correct (green), definitely capable of causing a run-time error (red), or if a decision could not be made (orange). Its vendor forecast that for software of this maturity, it would still find about one error per 1000 lines of code. This forecast was met exactly. Several function-critical errors in the flight software could be fixed that might otherwise have caused serious troubles.

A drop of bitterness are the orange lines: In our and others' [35] experience, rarely more than 20% of lines are marked as orange. Still, these lines cause considerable additional validation effort. For a supplier, it may be more appealing to use tools that report fewer problems than unnecessarily checking many potential ones. Knowing about potential problems but not acting on them is legally delicate; a supplier might therefore rather not want to know about them.

7.2. Towards Wider Adoption

The above downsides, much higher license prices compared to other (sometimes even free) tools, and generic organizational inertia were major problems to adoption. Furthermore, the ECSS recommends but does not explicitly require the use of static verification. Respective requirements brought into contracts from our side were rejected, leading to separate and tiring discussions in each of three projects that were moving to a new phase at that time. Implementing the requirement top-down through the ECSS system seemed impractical because seeking multinational consensus could take years and proofs of extensive experience would be needed.

The national catalog offered the chance to implement the requirement nationwide. The expert group, which convened at that time, could be convinced to commit to this sound static analysis. From the catalog, the new requirement gets tailored automatically into per-project requirements when a project moves to the next phase. At the same time, the requirement has strong legitimation through the catalog.

As of today, sound static analysis is widely employed. Even without being forced by a requirement, major suppliers have started using it in their projects themselves. However, every now and then, discussions still arise. For example, if a supplier is for its first time confronted with the need to deliver the respective analysis reports as proofs to customer product assurance. A supplier can then be pointed to the project's statement of compliance they signed, and to the expert group in which their own head of quality has possibly participated. This usually suffices to convince the supplier.

7.3. Other Process Improvements

The experience report provided here is only one example of a single software process improvement. Further technologies are continuously researched and evaluated, and, if considered fitting, will be introduced into the national program in a similar way.

For example, an in-depth study on the capabilities of four major verification tools is currently contracted (cf. [36]). In the study, the tools are applied to the code of on-board middleware to identify possible run-time failures. The results that the tools report are then compared and analyzed in-depth to draw conclusions about their capabilities. The tools' findings are compared to one another, validated for correctness (trying to identify false positive reports), and also looking for false negative reports. It seems that the application of different tools can reveal different faults, and that the application of more than one tool can make sense.

The study is also one step towards establishing a new dynamic verification method that stimulates to provoke run-time errors. Not much unlike the introduction of abstract interpretation, suppliers are reluctant to use new and other methods. Additionally, while first projects with the dynamic verification method were successful, its actual benefits are not yet documented well. Part of the problem is that conduction of the verification has to be contracted by its vendor. While such projects were conducted in the past, there is a problem: the verification vendor was contracted as a subcontractor of the flight software supplier, reporting only to the prime contractor. The supplier did not allow that detailed results were given to the customer or made public. So the benefits of the verification method are difficult to assess for customers. Currently, a project is running which will produce the needed results.

As another example, we made first efforts to obtain some software metric data along the lines of ECSS-Q-HB-80-04A in projects. This data is going to build the foundation of a new multi-organizational database to allow to improve knowledge about software product assurance. However, suppliers are reluctant to be the first to provide such data. In addition to being a cost factor with respect to surveying of the data, providing it to the customer is a business risk because new insights or misinterpretations of the data might be used against the supplier. Trust has to be created first. In the long run, the resulting improvements in visibility will benefit the collaboration between customer and the supplier.

8. RELATED WORK

Numerous publications address organizations' software process improvement work; e.g. [4]. Much effort has been put into standards and maturity levels as a means of giving customers ways to assess supplier capability, for instance, ISO-15504 or CMMI. However, few is published from the view of customers' product assurance. This paper is an extended version of [37].

Sometimes the ECSS standards are made applicable in space projects without tailoring and in their full extent. This can cause unnecessarily high project costs. Therefore, tailoring of requirements for software development is practically omnipresent in space projects. Most ECSS standards include a tailoring notice that explicitly encourages tailoring of the standard. The ECSS-S-ST-00C [26] explains a basic tailoring approach based on putting all requirements in a table, and marking them as applicable with/without change or newly generated. A variation that is used by another space agency in Europe is to include all requirements in their original form, and then record any changes or deletions after the original text. This leads to very long and complex documents.

ECSS-E-ST-40C [33] and ECSS-Q-ST-80C [29] are "self-tailoring", meaning that they provide an annex containing a table with a recommended tailoring according to the criticality of the software. While this tailoring method has the advantage of applying different requirements to different parts of a software, it is based on technical considerations only, i.e., criticality of the software in the scope of the system only. It does not consider programmatic or political aspects. In order to mitigate such issues, several ECSS standards currently under revision will include a similar default tailoring per space product type and per phase in the future.

For the obsolete standard revision ECSS-E-ST-40B, ESA provides a wizard-style tool, which asks a series of questions to determine tailoring-relevant characteristics. Also Döler et al. developed a web-based tool capable of tailoring several standards including ECSS-E-ST-40B and ECSS-Q-ST-80B. Again characteristics like technical domain, software type, operational complexity are surveyed using 17 non-redundant single-choice questions. Example items are (answer choices noted in parenthesis): Technical domain (ground segment, flight software, other), software type (processing chain, calculation, embedded software, simulator, ...), development effort (large or medium, small, very small), software affected system criticality (high, low), requirements complexity (high, low), or operational complexity (high, low). The tailoring rules are based on comparisons with other standards and long-term experience from working in space projects [38]. However, neither the ESA tool nor the tool by Döler et al. seem to be actively maintained anymore.

Kalus and Kuhrmann present a systematic literature review of criteria for software process tailoring. They identified 49 tailoring criteria, such as team size, project budget, project duration, the degree of technology knowledge, the availability of commercial off-the-shelf products, tool infrastructure, legal aspects or the domain [39].

Armbrust et al. address product quality through scoping, i.e., what (not) to include in a process. Their view complements ours from the supplier's perspective. Cooperation with ESA triggered process improvements on their side [40].

9. LESSONS LEARNED

This section summarizes several lessons learned regarding the various management areas described in this article.

9.1. Lessons regarding Product Assurance and Cost of Quality

Agile software development might not be suitable for flight software development. To some extent the reason is that development is driven by hardware development. More importantly, flight software is much more critical, which might mandate more classical development approaches (cf. [20]). In the future, efforts are needed to determine whether, or how, both worlds could fit together.

Product assurance can sometimes be an ungrateful task: one has to be careful not to be mistaken for a cost driver because positive quality costs are often obvious. The negative quality costs — especially if they are in the form of risks — may remain hidden for very long. If a risk commences one day and the negative costs are realized, product assurance is deemed guilty of not having prevented it. Creating transparency regarding the positive and negative costs of quality may be the only way of preventing product assurance from getting caught in this trap.

However, the actual positive and negative costs of quality are unknown to a wide extend. Without proper data on the effects that process requirements have on these costs, the "knowledge is of a meager and unsatisfactory kind", as Lord Kelvin put it. To improve the work of a customer's product assurance, quantified knowledge about the effects of product assurance requirements is needed. Initiatives for creating multi-company software metric databases (like [15] but) with the goal of creating quantified knowledge about process requirements are much needed.

9.2. Lessons regarding the ECSS Standards

The ECSS system is yet to be harmonized with the standards of other important space-faring nations like USA or Russia. However, as Germany does not have own launch capacities or sites, it must rely on such partners and therefore needs to implement foreign standards at national level. Additionally, ECSS working group compositions depend on who joins, and the varying stakeholders' self-interests lead to varying quality of resulting outputs.

At the moment, it seems that the peculiarities of agile development cannot be perfectly aligned with the existing ECSS standards. An indicator of this problem could be that the ECSS was not yet willing to assign a task force or working group to the development of a new standard. Instead, a working group is tasked with trying to squeeze agile development into the existing, more traditional framework of development. Experts have made the mention of the infamous *agile waterfall*, i.e., first several iterations of requirements engineering, then several iterations of design, then several iterations of coding, and finally, several iterations of testing.

9.3. Lessons regarding Pre-Tailoring

Pre-tailoring helps to break the complex system of ECSS standards down to national needs. It enables a gradual, smooth and careful while steady transition from traditional processes towards the ECSS standards. Over the years, the percentage of requirements in the national catalog originating from ECSS has constantly risen, reaching 43% in 2008, 57% in 2010 and 63% in 2012.

The national catalog also forms an agreed baseline with an unevenly greater influence for DLR (as compared to ECSS). Individual requirements are much easier to put through and decision making is faster. Still, because major players have their votes in the expert group, it gets more difficult for them to reject requirements during projects. Novel processes (e.g., a new verification method) have few chances to get into ECSS without being widely established, but without such legitimation they will not be easily accepted.

However, maintaining the national catalog is costly. Working through several standards each with hundreds of requirements takes its time. And the work needs to be repeated every few years in order to keep up with changes in the still developing ECSS standards. Outsourcing work through the expert group is an important way of distributing the involved efforts and to unburden space

administration personnel. The web-based content management reduces the amount of required co-location time. It enables people with densely filled appointment calendars to still collaborate.

Other methods and tools for tailoring software product assurance exist (see Section 8). However, none of them seems to be actively used anymore. One reason could be that the tedious maintenance of the requirements and rule database might have been too costly.

9.4. Lessons regarding Per-Project Tailoring

Tailoring is necessary for fitting coherent but generic standards to the specifics of a project. Our automated single-source tailoring improves the traceability and repeatability of this process. It no longer depends on subjective preferences of the product assurance manager who tailors.

Moreover, the national catalog contains hundreds of requirements applicable to the software development life cycle. So, manually tailoring requirements to a project is a huge effort. Our semi-automated tailoring process based on the QMExpert Tailoring tool is without frills but sophisticated. It is straight forward enough to be practicable. A single person can tailor a complete product assurance requirements document for a project in a short time. While manual intervention is still needed in several phases of the process, the tool significantly reduces the efforts for tailoring. Through the years, the tool has aged technically relying on dated libraries and technologies but the process it supports and its contents have matured. Much tacit knowledge and experience went into the requirement level classifications, contributing to the quality of the tailoring results.

9.5. Lessons regarding the Assurance Program Implementation

During the project, the statement of compliance simplifies communication and improves visibility between customer and supplier by clearly summarizing the agreed-upon baseline of quality-related processes. It is part of the contract and often a point of lively debates. Still, it is better to have a focused discussion at the beginning of the project than to have dissents about stretched interpretations of requirements concealed in lengthy prose plans. Attention should be paid to the understandability of comments because this key document can sometimes be around for many years.

Quantified data gets more important, when a direct and full access to the development team is not given, as it is the case in a customer and supplier relationship. While technical visits at a customer's site are formally possible any time, in practice this option is used seldom. In such a situation, where only management reports are available, it is difficult to get a feeling for actual problems and their severity. Software metric data can help in such a situation to improve visibility for the customer.

9.6. Lessons regarding Polyspace

Conducting a pilot project was important to learn whether sound static analysis is worth its cost. Although facing strong rejection from contractors first, it is now broadly accepted. The achievement was made by establishing and exploiting the right management tools: ECSS as the common baseline, influence and agility on national scope through the national catalog, legitimation prior to the project through influential key persons from the expert group, consistent and broad introduction through automated single-source tailoring, and during project, enforcement supported by a definite statement of compliance that quells unnecessary discussions.

10. CONCLUSION

We present software process improvement from the viewpoint of a customer. Our goal is to assure the quality of a product that is developed for a single purpose: to assume critical functions in a spaceflight mission. In a small market where products cannot be bought off-the-shelf but are unique specimen specifically developed for the customer, close collaboration between customer and supplier is necessary to reduce risks and improve processes. Given visibility and trust, both sides can profit from cross-company quality management through the exchange of experience and lessons learned.

To reach our goal, we lay down requirements tailored per project to suppliers' processes, thereby driving improvement from a strategic perspective. The actual organizational implementation is up to the supplier. Major problems are to harmonize processes and to convince suppliers of process improvements in a sector of diverse stakeholders with varying objectives, histories and experience. Using the example of introducing of sound static analysis into national projects, we explained how we exploited several management tools: ECSS as the common baseline, influence and agility on national scope through the national catalog, legitimation prior to the actual project through influential key persons from the expert group, consistent and large-scale adoption of requirements through automated single-source tailoring, and during the project, cross-company quality management supported by a definite statement of compliance. We feed back experience upstream into ECSS and other standards for harmonization, and generate and disseminate knowledge regarding software technology to continuously advance space software development and quality.

REFERENCES

1. Mark Dowson. The ariane 5 software failure. *Software Engineering Notes*, 22:84, 1997.
2. James Oberg. Why the mars probe went off course. *IEEE Spectrum*, pages 34–39, October 1999.
3. C. W. Johnson. The natural history of bugs: Using formal methods to analyse software related failures in space missions. *LNCS*, 3582:9–25, 2005.
4. Gerard J. Holzmann. Mars code. *Comm. of the ACM*, 57(2):64–73, February 2014.
5. Raymond Anderson. Columbia, the legacy of "better, faster, cheaper"? *Space Daily*, July 2003. Online. `http://www.spacedaily.com/news/rocketscience-03zk.html`.
6. Daniel L. Dvorak, editor. *NASA Study on Flight Software Complexity*. Jet Propulsion Laboratory, 2009.
7. Eberhardt Rechtin. Reducing the costs of space science research missions. In *Proceedings of a Workshop*, pages 23–29. National Academy Press, 1997.
8. Alison Abbott. Battery fault ends x-ray satellite mission. *Nature*, 399:93, May 1999.
9. Hans-Arthur Marsiske. Wendepunkt mars. *Telepolis*, 2000.
10. David N. Card. Software product assurance: measurement and control. *Information and Software Technology*, 30(6):322–330, 1988.
11. S. E. Donaldson and S. G. Siegel. *Successful Software Development*. P.-Hall, 2001.
12. Wilfried Ley. Management von raumfahrtprojekten. In *Handbuch der Raumfahrttechnik*, pages 715–764. 2007.
13. ECSS-E-HB-40A: Space engineering – software engineering handbook, 2013.
14. Victor R. Basili, Frank E. McGarry, Rose Pajerski, and Marvin V. Zelkowitz. Lessons learned from 25 years of process improvement: The rise and fall of the nasa software engineering laboratory. In *24th Intl. Conf. on Software Engineering*, pages 69–79. ACM, 2002.
15. D. Greves, B. Schreiber, K. Maxwell, et al. The esa initiative for software productivity benchmarking and effort estimation. *esa bulletin*, 87:84–88, 1996.
16. Barry Boehm. Software and its impact: A quantitative assessment. Technical report, 1972.
17. Bernd Hardung, Thorsten Kölzow, and Andreas Krüger. Reuse of software in distributed embedded automotive systems. In *Proceedings of the 4th ACM International Conference on Embedded Software*, EMSOFT '04, pages 203–210, New York, NY, USA, 2004. ACM.
18. Henry Apgar. *Space Mission Engineering: The New SMAD (STL vol. 28)*, chapter Cost Estimating, pages 289–324. Space Technology Library. Microcosm Press, 2011.
19. Daniel Galin. *Software Quality Assurance: from Theory to Implementation*. Pearson Education, 2004.
20. Barry Boehm. Get ready for agile methods, with care. *Computer*, 35:64–69, 2002.
21. Ian O'Neill. The mars curse. Online. `http://www.universetoday.com/13267/the-mars-curse-why-have-so-many-missions-failed/`, 2008.
22. Hermann Kaufmann, Stefan Hofer, and Christian Chlebek. A hyperspectral future — enmap is redefining earth observation. *Countdown – Topics from DLR Space Agency*, 1:26–29, 2009.
23. Dominik Hayon. Raumfahrt: So viel zahlt deutschland an die esa. online. `http://www.chip.de/news/Raumfahrt-So-viel-zahlt-Deutschland-an-die-ESA_74218929.html`, 2014.
24. Christian R. Prause, René Reiners, and Silviya Dencheva. Empirical study of tool support in highly distributed research projects. In *Intl. C. on Glob. Soft. Eng.*, ICGSE. IEEE, 2010.
25. Myron J. Hecht. *Space Mission Engineering: The New SMAD (Space Technology Library, Vol. 28)*, chapter Reliability, pages 753–767. Space Technology Library. Microcosm Press, 2011.
26. Ecss system – description, implementation and general requirements. Standard, ECSS Secretariat, 2008.
27. Y. El Gammal and W. Kriedte. ECSS – an initiative to develop a single set of european space standards. In *Proceedings of Product Assurance Symposium and Software Product Assurance Workshop*, pages 43–50. ESA, 1996.
28. Daniel Schiller and Jürgen Heinemann. Ecss – 20 years of collaboration for european spaceflight. *DLR Newsletter Countdown*, pages 32–35, 2014.
29. Space product assurance – software product assurance. Standard, ECSS Secretariat, ESA ESTEC, 2009.
30. Space product assurance – software dependability and safety. Standard, ESA-ESTEC, 2012.

31. Space product assurance – software metrication programme definition and implementation. Standard, ECSS Secretariat, ESA ESTEC, 2011.
32. M. Jones, E. Gomez, A. Matineo, and U. K. Mortensen. Introducing ecss software-engineering standards within esa. *esa bulletin*, 111:132–139, August 2002.
33. Space engineering – software. Standard, ECSS Secretariat, 2009.
34. Wolfgang Jobi, editor. *Tailoring Catalogue: Product Assurance & Safety Requirements for DLR Space Projects*. DLR, 2012.
35. G. Brat and R. Klemm. Static analysis of the mars exploration rover flight software. In *Intl. Space Mission Challenges for Inf. Technology*, pages 321–326, 2003.
36. Christian R. Prause, Ralf Gerlich, Rainer Gerlich, and Anton Fischer. Characterizing verification tools through coding error candidates reported in space flight software. In *Data Systems In Aerospace*, DASIA. Eurospace, 2015.
37. Christian R. Prause, Markus Bibus, Carsten Dietrich, and Wolfgang Jobi. Tailoring process requirements for software product assurance. In *Proceedings of the International Conference on Software and System Process*, ICSSP, pages 67–71. ACM, 2015.
38. Norbert Döler, Anita Herrmann, Uwe Tapper, and Rolf Hempel. Ecss application in dlr space projects – experiences and suggestions for enhancement. Presentation slides from the ECSS Developer Day at ESTEC (Noordwijk), November 9th-10th, 2005.
39. Georg Kalus and Marco Kuhrmann. Criteria for software process tailoring: A systematic review. In *Proceedings of the International Conference on Software and System Process*, ICSSP, pages 171–180, 2013.
40. O. Armbrust, M. Katahira, Y. Miyamoto, J. Münch, H. Nakao, and A. Ocampo. Scoping software process models – initial concepts and experience from defining space standards. LNCS 5007:160–172, 2008.