Software Engineering: Are we getting better at it?

Michael Jones
Mission Data Systems Division, ESA Directorate of Operations and Infrastructure, ESOC, Darmstadt, Germany

All modern businesses throughout the World depend on software and for many of them it is mission-critical. Software therefore accounts for a significant proportion of global GDP. It follows that software engineering, which enables efficient development of quality software, is of great importance and not just a niche specialisation. Of course, in ESA and in the space business generally, we are critically dependent on software and associated Information Technology (IT) systems, which pervade all our space systems, both onboard the spacecraft and on the ground.

Why Software Engineering is Important

Surveys have shown that the average corporation pays 3-7% of its annual revenues for hardware and software for corporate data processing, and another 2-5% for maintenance: that is 5-10% of the economy in the developed World! Information technology is critical for many businesses in that they rely on the efficiency and integrity of their IT systems for their day-to-day operations. This is especially the case for ESA and the European space industry, where software is all-pervasive in onboard avionics and payloads, in ground segments and also in the associated commercial business processes.

An epigram by the philosopher George Santayana carved over the entrance to the US Archive in Washington DC reads: *"Those who cannot remember the past are doomed to repeat it."* A variation of this could be: *"Those who cannot remember the past, may not be able to repeat their successes"*. So you need to remember the past in order to avoid mistakes and to repeat successes.

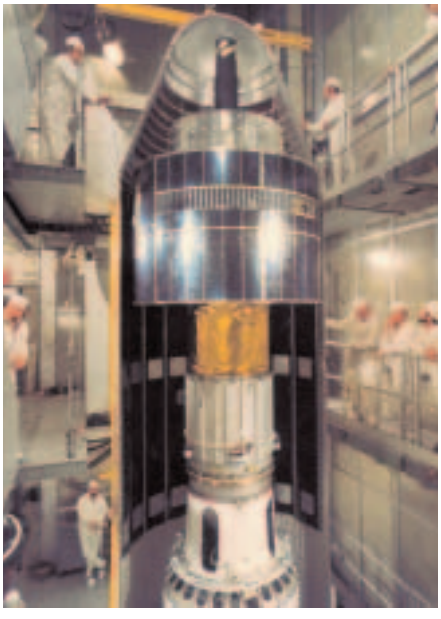
The need to take account of 'lessons learnt' is well-recognised in ESA and measures to ensure the feedback of experience into improved working practices are being actively pursued. In the case of software-engineering, the Agency has been doing this for over 25 years. In fact, software engineering is basically about recording experience in developing and maintaining software and documenting it in the form of best practices.

Software Engineering Advances at ESA

Advance 1: The importance of requirements

Software engineering in ESA started in the mid-1970s. The initial impetus came from the experience of Dr. Carlo Mazza during the Meteosat Ground Computer System (MGCS) project, which was responsible for the data-processing system for the first of ESA's meteorological spacecraft. The MGCS covered mission control, payload data processing, dissemination of meteorological products generated from the payload data, interactive product quality control and data archiving. The system aimed at a high degree of automation, particularly for repetitive but complex tasks such as the extraction of wind directions from sequences of satellite images, which was unusual at the time.

It became apparent about a year before Meteosat's launch that the MGCS project was in difficulties. The problem as Carlo Mazza saw it was that the requirements that the system was supposed to meet were not documented. Carlo therefore took the decision to stop work on development (i.e. design and coding) and document the requirements – a decision that required both vision and courage – so that management could then make informed decisions on development priorities.



The first Meteosat spacecraft, being readied for its launch from Eastern Test Range in Florida on 22 November 1997

Console at the XMM-Newton Science Operations Centre at Villafranca, near Madrid

The lesson learned was never to undertake such a project without first drawing up well-defined requirements, and it subsequently resulted in the formation within ESA of the Board for Software Standardisation and Control (BSSC), with Carlo Mazza as its first Chairman. Its terms of reference included the provision of a software engineering standard, which was eventually published as ESA Procedures, Standards and Specifications document PSS-05-0. It clearly highlighted the distinction between user requirements and software requirements, which was far from being well recognised at that time. In fact, a key message from the authors of that standard was the need to analyse and negotiate user requirements to produce an unambiguous baseline for the software developers, which can be implemented with the technology available and within the budget provided.

Advance 2: Customer/supplier equilibrium and firm-fixed-price contracts

In the 1980s and early 1990s, most software development at ESOC was done under Fixed Unit Price (FUP) conditions. That meant that the Centre bought man-hours of effort from a contractor and took full responsibility (i.e. fully 'owned the risk') for the software requirements and their implementation. As the software contractor team took no such responsibility, they were under no obligation to scrutinise requirements critically. In fact, the situation was quite the reverse – the more requirements that were accepted, the more work the software contractor got, but without ever having to take responsibility for the end product.

The problem with this lack of a suitable customer/supplier equilibrium, or division of responsibilities, came to a head with ESOC's SCOS II project, where, for the first time, an ambitious attempt was made to write comprehensive user requirements for a reusable spacecraft-control system. The resulting User Requirements

Artist's impression of the Envisat spacecraft

Document was impressive, with over 7500 requirements, but they were neither layered nor prioritised for staged implementation. There was no resistance from the supplier's side because under the FUP approach, which was then the norm, the supplier did not have to make any committing cost estimate.

In 1995, the concept of Firm Fixed Price (FFP) contracts for the development of spacecraft control systems and simulators was systematically introduced at ESOC for software development. This was motivated at the time not by issues of risk ownership, but rather by the need to move contract staff off-site to their own companies' premises to relieve site overcrowding, something that was much more practicable under an FFP regime whereby the contactor is providing either a service or a product. This idea very much influenced the new 'frame contracts' that were issued around that time (in fact, late 1996). The result was:

- Far more rigorous scrutiny of requirements by frame contractors.
- Equitable risk sharing between ESA and its suppliers.
- Formal change control (Contract Change Notices, or CCNs) for changes in requirements. A consequence was that end-users became more careful about defining their requirements.

Projects that immediately benefited from the new regime included:

- the development of the XMM-Newton Science Operations Centre (SOC)
- the development of the Envisat Mission Control System (MCS)
- the consolidation of the SCOS II command chain.

The XMM-Newton SOC is a particularly good example because the software was developed in London and Rome, integrated in Darmstadt, Germany, and finally installed and accepted at ESA's Villafranca ground station in Spain. The team consisted of over 50 software developers at its peak. Although XMM-Newton was launched ahead of the original schedule, both the control system and the SOC were ready in time.

Advance 3: Devising a good software engineering standard

Software engineering standards are used by many people and so they have to be easily understandable and practical to apply. To take one example, ESA PSS-05-0 covered:

- software engineering processes, and
- supporting processes (software project management, software configuration management, software quality assurance, and software verification and validation).

It was a compact volume of about 100 pages, covering:

- about 200 practices
- five major output documents
- four plans.

This was a remarkable achievement by the BSSC because it is concise, comprehensible and contains the right balance: providing sufficient detail, but not too much, and leaving leeway for the application of local practices. It is not over-prescriptive and focusses on important reviews and outputs.

It is easy to get this balance wrong, as illustrated by the example of the SCOS II Development Standard. One of the early and, as it turns out, correct decisions taken on the SCOS II Mission Control System project was to use Object-Oriented development methods, which, it was thought, demanded iteration during the development. The SCOS II development team prepared a development standard based on PSS-05, but explicitly spelling out the iterations within the phases. This turned out to be totally impractical, since the iteration involved repeated reviews and associated document deliveries, which, if followed, would have led to review overload. Thus it destroyed the PSS-05 equilibrium by taking a highly prescriptive approach to iteration.

As anyone who has been involved with architectural design will know, the design described in an Architectural Design Document does not spring fully formed from the minds of the designers. There is always iteration and re-design, even if it is not explicitly spelled out. This, however, is not the main lesson!

Apart from this fundamental methodological flaw, the SCOS II Development Standard was rather difficult to read, perhaps as a result of being written with limited resources by people without the necessary standards-writing skills or experience. So the main lesson is don't try to rewrite software engineering standards as part of your project. Writing such standards is time-consuming and requires special expertise, apart from the fact that most project funding does not include allowances for rewriting standards.

Advance 4: Sustaining expertise via frame contracts

Frame contracts were first introduced at ESOC in the early 1980s. The initial idea was based on the observation that following the full classical ESA procurement approach for all software developments was slow and inefficient, with a high administrative overhead. The basic idea of the frame contract is to use the full ESA contracting procedure to select a limited number of competing contractors that would be awarded all the software development work in a given domain over a 3 to 5 year period. Work packages are awarded competitively either in the initial tendering or later during the contract's duration. A streamlined competition procedure is used for each new work package.

These frame contracts were an important advance because:

- they encouraged competition
- they developed pools of expertise
- the expertise thus developed became a competitive advantage, not just for ESOC, but also for the frame contractors themselves who, on the basis of experience at ESOC, were able to get work elsewhere in the space industry.

Advance 5: Increased software productivity through reuse

A quite remarkable feature of the original terms of reference of the BSSC is the inclusion of responsibilities relating to software intellectual property rights, and software libraries. We now take the re-use of software, particularly on our desktop PCs, for granted. In 1977, however, when the

BSSC's terms of reference were written, software re-use was not such an obvious productivity winner as it now. It almost certainly stemmed from the environment at ESOC, where re-use started early and seriously with the Multi-Satellite Support System (MSSS) in the early 1970s.

There are various strategies for increasing computer programming productivity, but software re-use is arguably the most effective. At ESOC the results have been very apparent for mission control systems, where the ease of use and functional richness of the SCOS-2000 reusable mission control infrastructure has reduced development costs by a factor of 3-5 over the past six years or so (see Tables 1 to 3).

The XMM-Newton and SMART-1 missions are both very successful examples of software reuse. The gains are lower in the case of XMM SOC because there were many requirements for which reusable software did not exist, but 50% is nevertheless a still-impressive doubling of productivity. For SMART-1, only about 7% was new code, giving a very substantial leverage effect: apparently for an investment of 7% you get 100%. This is a bit over simplistic, because one also has to take into account the costs of integrating the reused software and testing the whole system. The fact remains, however, that a very sophisticated control system was developed at a low cost.

Software reuse works best when the software has been specifically developed with reuse in mind, which usually means a considerably larger initial investment, and that the end users must commit to the software's reuse.

Software Engineering Issues Today

Software failures

Software failures are still a serious concern for everyone in the space business, as the accompanying table shows. ESOC is no exception and there have recently been several potentially mission threatening software failures, including:

- MCS failure during the MSG launch and early operations phase (LEOP) 20 minutes after launch.

Table 2. SMART-1 MCS reuse figures in units of number of modules (files) from each contributing project

Subsystem	LOC
XMM Science Ops. Centre	30,000
Payload Monitoring	27,000
Proposal Handling	17,000
Sequence Generation	22,000
Archive Management	42,000
File Transfer	13,000
Payload Calibration	40,000
Quick-Look Analysis	32,000
Observation Data Handling	14,000
Infrastructure/Libraries	13,000
Subtotal XSCS w/o SCOS-1	250,000
SCOS-1A/B	250,000
Total	**500,000**

Table 1: XMM-Newton software reuse in terms of lines of code (LOC)

- Detection of a problem with handling the leap year (again in the MCS) a few days before the Rosetta launch.

How can such problems be avoided? There is no easy answer to this, but first we should look into the background. As explained above, ESOC now outsources the development of software, but remains responsible for the definition of requirements and acceptance testing. The delivered system is therefore acceptance-tested against the requirements, for which good tools have been developed at ESOC. The problem is that this is 'black box' testing, in the sense that it is done without taking into account the system's design or structure, i.e. the box is opaque! As such, it is not exhaustive, and many parts of the software product will not be executed if only explicit requirements are addressed. Testing against requirements therefore has to be complemented by extensive structural or 'white box' testing, which should ideally cover all the logic paths through the software. This is in effect outsourced to the software developers. Another factor is that error removal usually occurs at the end of the development project's life cycle and in the event of a schedule crunch is the most likely task to get neglected.

The obvious conclusion is that error-free software is an impossible goal. A more realistic goal is that of producing software without high-severity errors. We therefore need systematic ways of detecting the latter!

ECSS Software Engineering Standards – The Problem of Tailoring

The judicious equilibrium of the original PSS-05 in terms of prescriptiveness versus freedom has already been mentioned. What of the latest standard currently in use in ESA, namely ECSS-E-40? Like the solid international standard ISO 12207 on which it is modelled, ECSS-E-40 is based on the twin concepts of:

– comprehensiveness: it covers all types of projects, including highly critical ones, and
– tailoring to adapt to individual projects.

In the particular case of E-40, tailoring is essential, since if it were applied in full it

Software Origin	Source	Header	Total	%
SCOS-2000 R2.2.1	953	929	1882	76.4
Rosetta	139	160	299	12.1
Integral	63	55	118	4.8
Meteosat Second Generation	2	3	5	0.2
SMART-1	73	85	158	6.4
Total	**1230**	**1232**	**2465**	

Project	% Reuse	Productivity gain
XMM-Newton Science Ops. Centre	50	2
SMART-1 Misson Control Centre	93.6	16

Table 3. Productivity gains from software reuse

would be too heavy and prescriptive and would result in too many output documents, many of which have to be delivered several times. However, the sensible tailoring of a complex standard like ECSS-E-40 requires a detailed understanding of the whole standard, which the average software developer or project manager does not possess. Two approaches have therefore been identified:

– *Use of a tailoring tool*: The latter allows the developer to answer a set of questions about the project and the tool tailors out requirements in a consistent way, according to the answers. The tool can then print a table of retained requirements and output documents required.
– *Organizational tailoring:* For an organization such as ESOC many of the projects are similar, and so it could be expected that the same tailored standard could be used for all of them. Alternatively, a limited number of tailored versions could be considered, corresponding to different categories of projects in the organization, e.g. safety-critical and other (i.e. non-safety-critical) projects.
The second approach is the one we to apply at ESOC.

Project	Incident	Root cause
Ariane 501, June 1996	Launch failure, loss of 4 Cluster spacecraft	Lack of proper software and system validation
Mars Climate Orbiter, September 1998	Loss of Orbiter	Confusion between imperial and metric units. Lack of proper software/system validation
Mars Polar Lander, December 1998	Loss of Lander	Software error forced premature shutdown of landing engine: lack of proper software validation
Titan-4B, April 1999	Failure to put USAF Milstar spacecraft into useful orbit	Centaur upper stage failed to ignite. Decimal-point error in the guidance system. Lack of proper software validation
Delta-3, April 1999	Launch aborted	Failure of on-board software to ignite main engine. Lack of proper software validation
PAS-9, Arch 2000	Loss of ICO Global Communication F-1 spacecraft	Error in an update to ground software. Lack of proper software validation

Table 4. Software-related mission failures

Conclusion

It is time to return to the question posed in the title of this paper, namely 'Software Engineering: are we getting better at it?' The reader will hopefully agree that, with the major advances described here, we have come a long way in the last 25 years, and that ESA as an organization is in a much better position than when the BSSC was first set up to establish software engineering standards for the Agency.

Firstly, this article has looked at software engineering as a branch of the empirical discipline of computing and asked if laws or rules exist to describe the practices or processes concerned. The conclusion is that while there is no 'unified field theory' of software engineering, there is a useful body of rules and hypotheses.

Secondly, it has highlighted a number of practical advances in the application of software engineering at ESOC that have demonstrably contributed to sustainable success in developing software, and in some cases have helped to reduce costs by remarkable amounts. The conclusion here is that we have indeed got much better at software engineering, and advances have also been made in the management and contractual approaches. However, the need for vigilance is eternal: with highly complex space systems there is always a danger of something going wrong, no matter how careful we think we have been about software development and validation.