Software Product Line Engineering for Long-lived, Sustainable Systems

Robyn Lutz[1,2], David Weiss[1], Sandeep Krishnan[1], Jingwei Yang[1]

[1] Department of Computer Science, Iowa State University
[2] Jet Propulsion Lab/Caltech
{rlutz, weiss, sandeepk, jwyang@cs.iastate.edu}

Abstract. The design and operation of long-lived, sustainable systems (LSS) are hampered by limited support for change over time and limited preservation of system knowledge. The solution we propose is to adopt software product-line engineering (SPLE) techniques for use in single, critical systems with requirements for sustainability. We describe how four categories of change in a LSS can be usefully handled as variabilities in a software product line. We illustrate our argument with examples of changes from the Voyager spacecraft.

Keywords: software product line, sustainable system, long-lived system, variability, commonality/variability analysis.

1 Introduction

Sustainable: "meets the needs of the present without compromising the ability of future generations to meet their own needs"

- UN Brundtland Report, on sustainable development [1]

Our society is becoming increasingly dependent on software-intensive sustainable systems. Examples include embedded medical devices, web-based archives, interplanetary spacecraft, power grid monitors, telecommunication switches, and sensor networks. Future examples include nuclear power plants, health databases, and global networks of solar arrays, perhaps in orbit. Many such systems are safety critical, with varying degrees of autonomy. They typically evolve over long periods of time in response to changed needs, new technologies, and failed components.

We consider a sustainable system to be one that has the following attributes.

- It has an extended lifetime,
- It makes efficient use of resources to achieve its goal.
- It maintains its capabilities despite obstacles and failures.
- It is adaptable, so as to accommodate change, and is expected to evolve with changes in technology and requirements.

More broadly, a sustainable system is forward-looking and is structured so as to guide future decisions. The goal of evolving over time to meet changes in technology and requirements, distinguishes sustainable systems from legacy systems. Accordingly we use the term long-lived, sustainable systems, or LSS, for them.

In this paper our perspective is the preservation of system knowledge over time in the service of handling change (both anticipated and unanticipated) in LSS. While the preservation of knowledge and change handling are not unique to LSS, extended LSS lifetimes exacerbate the problems. LSS have a longer period of operations over which both planned and unplanned change can occur. Their long operational periods are accompanied by considerable personnel turnover, resulting in knowledge loss that complicates operations and adaptive maintenance. Historically, these inadequacies have jeopardized LSS [2]. Their design and maintenance is challenged by the need to envision, plan for, and handle on-going change, and to preserve and pass on the knowledge needed to do so.

The problem, then, is *how to better design and operate a LSS to preserve system knowledge and to support needed changes over time*. The solution we propose is to adopt software product-line engineering (SPLE) techniques for use in single LSS, an adoption that we believe is natural to both. SPLE provides a process framework to identify, document, and make decisions regarding alternatives now and in the future, taking into consideration their risks, dependencies and consequences, both in cost and value. It focuses on sustaining artifacts and domain knowledge over a long haul.

Change can be usefully treated as variability, and SPLE handles variability well. To illustrate this we discuss examples of anticipated and unanticipated changes from the twin Voyager spacecraft, launched in 1977 and still actively collecting science data. We show how Voyager "did it right" in planning and organizing for change, and in maintaining system knowledge.

We suggest that the lessons to be learned from Voyager regarding the design and operation of a LSS system are not just consistent with software product line engineering but are, in fact, most readily transferable to other LSS in the context of SPLE techniques. SPLE describes how a set of similar systems—a software product family—develops over time. We treat an evolving LSS as if it were a set of similar systems that developed over time. (In fact, PLE also evolves over space, that is, there may be several products in a product line that are produced and maintained concurrently, so our problem is simpler.) This paper thus proposes *to apply SPLE techniques to single systems, where those single systems must be long-lived and sustainable*, and presents, in the context of Voyager, the advantages of doing so.

2 LSS Example: the Voyager Spacecraft

The two Voyager spacecraft, launched in 1977 and now the farthest human-made objects from Earth, are among the best-known LSS. The spacecraft continue to return truly invaluable data as they approach the heliopause. For example, Voyager 2 recently discovered a strong magnetic field that holds the interstellar cloud together [http://voyager.jpl.nasa.gov/]. The Voyagers are expected to continue to communicate until loss+ of power and fuel mutes them around 2020. The spacecraft have efficiently used their early-1970's era resources to adapt to a changing set of ambitious scientific goals. The spacecraft software has also been repeatedly changed to handle failed components and reduced power. Voyager did not explicitly use software product line engineering. However, viewed in retrospect, Voyager exemplified the SPLE process

of carefully identifying possible variations that might be needed in the future, of designing a modularized architecture that would allow those anticipated changes to be made, and of specifying the constraints that would guide the decisions to be made. Voyager also demonstrated that even unanticipated change is made easier when a serious effort has been made to design for possible future changes.

3 SPLE for LSS Change Management

The software product line engineering FAST process, used here, identifies, distinguishes, and documents what is assumed to stay the same (across systems and time) and what may change [3, 4]. It relies on three artifacts: (1) a commonality/variability analysis that formally specifies the allowable range of values for each variability, the constraints among the choices of value for the variabilities, and the binding time for each variability; (2) a modularized architecture with a mapping between modules and the commonality/variability specifications described above; and (3) a specification, called a Decision Model, of the partially-ordered sequence of choices that must be made to build a new product, subject to the constraints and binding times specified earlier.

Some required behavior must be invariant for a LSS to succeed. For example, a software requirement that has to be satisfied throughout the lifetime of the Voyager spacecraft is that it shall be able to communicate with Earth and automatically detect and respond to a loss of uplink from Earth. Similarly, a LSS is built and operated on certain assumptions regarding those things that will not change (e.g., in the environment). Such assumptions can usefully be modeled as commonalities. Note also that if the assumptions later become false, we have a way in FAST to document both what the change is and why the change occurred, preserving knowledge and providing guidance to later generations of maintainers, as suggested in [5].

We next describe how the handling of both anticipated and unanticipated change can be improved by the use of software product line engineering techniques.

3.1 Anticipated Changes

We can anticipate some changes that will likely be made during the lifetime of a LSS. Several standard techniques assist in this identification: investigation by domain experts, experience with similar systems, goal/obstacle analysis, defect patterns in similar systems, and analysis of previous changes. On spacecraft, we know that if hardware breaks, the software will often have to be updated to take on the required capability previously allocated to hardware [2]. Similarly, we know that as different mission phases are reached (e.g., launch, interplanetary cruise, orbital insertion) the software will need to be updated. In LSS, many of these changes will be made to handle failure or degradation of hardware components.

Such anticipated changes can usefully be modeled as product line variabilities in the commonality/variability analysis. The FAST process documents the envisioned ranges of optional and alternative requirements and parameters. Making an anticipated change after launch then becomes analogous to taking a different path

through the Decision Model. In so doing, you use the Decision Model to check that the impact of the change is acceptable, based on the constraints between the proposed alternative and the choices implemented earlier to produce the existing product. This provides some assurance that the software architecture can accommodate the change. Checking the constraints can often be partially or fully automated.

Modifiability, or "changeability", is a defining attribute for a LSS. In [6], the quality attributes of modifiability are organized into four categories: (1) extensibility (changing capabilities, adding new functionality, repairing bugs); (2) deleting existing capabilities; (3) portability (adapting to new operating environments); and (4) restructuring (modularizing, optimizing, or creating reusable components).

We describe critical changes that occurred on the Voyagers during operations for the first three of these categories. Because of page limits, we exclude the fourth category here, but note that several subsequent spacecraft re-used Voyager hardware and software. We interleave examples from the two spacecraft, despite some small differences between them. For each change, we show how it fits into a SPLE context.

Anticipated extensibility: At launch, the Voyagers' authorized flight plan included only Jupiter and Saturn, primarily for budget reasons. However, the spacecraft had been designed for extension to take advantage of the fact that Jupiter, Saturn, Uranus and Neptune were aligned, something that happens once every 175 years. When the flight was extended for the "Grand Tour" to all four planets, the architectural design was in place to allow this. Note that the cost of designing for this extensibility was far outweighed by the value of the scientific knowledge gained from it.

Anticipated deletion: It was known that each instrument drawing significant power would have to be turned off at some point, as the onboard battery capacity decreased. For example, the cameras were turned off in 1990 after the last planetary encounter. Weighing the tradeoffs and deciding when in the mission to turn each instrument off was a complicated task, but made possible by anticipating its need.

Anticipated portability: A new algorithm was required to obtain images at Neptune. Without it, the low sunlight levels, combined with the torque imparted when the tape recorder was turned on and off, would have caused images to be smeared. The new feature automatically fired the attitude jets to compensate for the spacecraft torque at the longer exposure rate.

3.2 Unanticipated Changes

Unanticipated change is a problem for any system. SPLE provides a structure for dealing with unanticipated change as well as anticipated change. In particular, it allows one to reason about the effects, dependencies, and risks of a proposed change.

Unanticipated extensibility: In 1987, a new science opportunity appeared. A supernova occurred that Voyager could observe. Software commands were thus designed and sent to the UV spectrometer to capture data from the stellar explosion, taking advantage of a design that anticipated the need to reprogram the spacecraft.

Unanticipated deletion: Slewing rates for the scan platform (containing the cameras, etc.) were unexpectedly restricted by a new project policy for the planetary encounters of Uranus and Neptune. The change was in response to an earlier incident where the platform jammed, likely induced by a period of heavy usage.

Unanticipated portability: Soon after launch, Voyager 2's primary receiver failed and its backup receiver was reduced to "hearing" in a very narrow, changing frequency band. To compensate, a new ramping algorithm was quickly designed and implemented, so that prior to sending any software commands to the spacecraft, ground operations could tune the transmission to the receiver's current state.

SPLE excels at identifying what needs to be known and storing it so that the new customer (here, the spacecraft team) need only be concerned with the information preserved in the structures, rather than having to learn all the underpinnings of the system. Because the system is specified and designed for change, new personnel know where to look to understand the system and the implications of change.

4 Conclusion

Our hope is that use of SPLE techniques will make it easier to make changes to LSS where the value/cost ratio is high, as it is on spacecraft and other critical or one-of-a-kind systems. It remains to test this hypothesis, perhaps as a shadow effort with an on-going LSS project. In particular, we think that these SPLE techniques will be easy to use and fit in readily with the way sustainable projects work.

The benefits that we anticipate may accrue from the use of SPLE for LSS include:

- Improved preservation of project knowledge over extended lifetimes, leading to lower cost to maintain.
- Better capture of assumptions (commonalities) and dependencies among choices (variabilities) that can help reduce risk of change.
- Increased emphasis on investigating and ranking possible changes and on verifying architectural support for modifiability.
- Viewing potential changes as options that, when exercised, can bring high value, and so merit investment to preserve the needed information.

Acknowledgments. This work was supported by grants 0541163 and 0916275 from the National Science Foundation.

References

1. Our Common Future, Report of the World Commission on Environment and Development, Oxford University Press, (1987).
2. R. R. Lutz and I. C. Mikulski, Empirical Analysis of Safety Critical Anomalies during Operation, IEEE Trans. Software Engineering, vol. 30 (3), 172–180, (2004).
3. Weiss, D. M., Lai, C. T. R.: Software Product-Line Engineering, A Family-Based Software Development Process. Addison-Wesley (1999).
4. Weiss, D. M., Li, J. J., Slye, H., Dinh-Trong, T., and Sun, H.: Decision-Model-Based Code Generation for SPLE, in SPLC'08, pp. 129-138. (2008).
5. Parnas, D. L., Clements, P. C.: A rational design process: How and why to fake it, IEEE Trans. on Software Engineering, vol. 12(2), 251-257, (1986).
6. Bass, L., Clements, P. and Kazman, R.: Software Architecture in Practice, Addison-Wesley (1998).