Don't get taken by surprise: planning for software obsolescence management at the ALMA Observatory

Erich Schmid[a*], George Kosugi[b], Jorge Ibsen[c], Morgan Griffith[d]

[a] European Southern Observatory, Garching bei München, Germany
[b] National Astronomical Observatory of Japan, Tokyo, Japan
[c] Joint ALMA Observatory, Santiago de Chile, Chile
[d] National Radio Astronomy Observatory, Charlottesville, VA, USA

ABSTRACT

ALMA is still a young and evolving observatory with a very active software development group that produces new and updated software components regularly. Yet we are coming to realize that - after well over a decade of development - not only our own software, but also technologies and tools we depend upon, as well as the hardware we interface with, are coming of age. Software obsolescence management is needed, but surprisingly is not something we can just borrow from other observatories, or any other comparable organization. Here we present the challenges, our approaches and some early experiences.

Keywords: ALMA, obsolescence, software, computing, processes, risk analysis, prioritization, mitigation

1. INTRODUCTION

The ALMA observatory is gearing up for full operations, and while most parts of the ALMA software are still being very actively developed and improved, some of the underlying technologies, third-party tools and hardware, as well as parts of the ALMA software itself are already showing signs of age. This is no big surprise after well over a decade of development in the rapidly changing world of software engineering. But it needs to be managed proactively to avoid small issues becoming big problems in the long run. An obsolescence management plan should come to help.

For hardware to be maintained having a solid obsolescence management plan is very common and many good examples and practices to follow can easily be found. For software this does not seem to be the case. While the core principles of establishing and maintaining a bill of materials, performing risk analysis, prioritization and mitigation can be taken over from hardware in a very similar fashion, many of the considerations to be taken into account for assessing the probabilities and risks are very different for software.

Based on well-established obsolescence management practices for hardware, we present an approach to tackle the problem of software obsolescence management for the ALMA observatory. This is not only important to keep our software maintainable for the lifetime of the ALMA observatory as the environment around us evolves and changes, but also to avoid falling victim to the urge for never-ending rewriting, refactoring and replacement.

The obsolescence process is of course part of the larger maintenance and operations context. Risks identified and actions suggested from this process must be combined with overall observatory priorities to determine which risks need to be addressed and which actions need to be taken.

We hope that this paper will help others in a similar situation to become aware of the need for software obsolescence management and perhaps benefit from our ideas and findings, and maybe more so to help establishing an active dialog between projects and organizations to mutually benefit from each other.

[*] eschmid@eso.org; phone +49 89 3200 6326; www.eso.org

2. WHAT IS SOFTWARE OBSOLESCENCE?

In most complex systems, software life cycle costs (redesign, re-hosting and re-qualification) contribute as much or more to the total life cycle cost as the hardware, and the hardware and software must be concurrently sustained [1]. ALMA is of course no exception. So we took the action of looking into creating a software obsolescence management plan. The first thing one does in such a case, when we face a task that none of us has ever tackled before, is to check on the Web. To our surprise, there was not much to be found. Apart from some articles that introduce the need for software obsolescence (e.g. [1]), there was no recipe that we could use. So the next step was to ask our colleagues, who after all work for some of the biggest and well established observatories, but once again we were left with the encouragement that this is a great idea and something they will also need to address in the near future. With the promise to stay in touch and keep them in the loop, we went off and started at the very beginning by asking the question, "What exactly are we trying to do?".

A typical definition of hardware obsolescence is something like:

"A part becomes obsolete when it is no longer manufactured, either because demand has dropped to low enough levels that manufacturers choose not to continue to make it, or because the materials or technologies necessary to produce it are no longer available."[1]

This looked simple enough to be applied to software. So we derived from it the following list, that tells us that a piece of software – big, small, in-house, commercial or open-source – becomes obsolete if:

- … it is no longer supported by the provider
- … it is no longer available for purchase or download
- … it is no longer updated
- … its user community shrinks
- … it is superseded by better products we would benefit from, if we used them
- … it no longer runs on the platforms we use or plan to use
- … it does not work well or at all with other new products or technologies introduced
- … it does not keep up with our growing demands
- … it becomes known for security vulnerabilities
- … it is no longer needed
- … we cannot easily hire skilled people to support it anymore

Collecting this list, it also became evident that there are many not so good reasons to declare something obsolete, or rather use it as an excuse to introduce a new piece of software into the observatory, something most software developers love to do. So we had better complement this list by another one that tells us what software obsolescence is not. So a piece of software has not become obsolete because:

- … a different tool has appeared that does the same job, but is more modern, has a fast growing community, or is simply the *cool thing* of the moment
- … a different tool does the job better, but we don't really have a problem to fix
- … we would use a different tool if we started again today
- … there is a cheaper or free alternative
- … another project uses something different

Saying this, any of the items in this list may be a legitimate reason to decommission a particular tool and move to another one, but we should avoid confusing this with obsolescence management. Obsolescence must not be used as an excuse to switch to newer technologies quicker. Quite the contrary approach should be applied. It makes a lot of sense to apply a conservative approach of trying to stick with technologies that have been extensively tested as long as possible. Do not forget regular operations as the best possible test bed for any integrated system. The same level of confidence about a tool or technology will never be reached by any evaluation or test campaign, however detailed and well thought out it may be.

3. A STEP-BY-STEP APPROACH TO SOFTWARE OBSOLESCENCE MANAGEMENT

We take the top-level process flow suggested in [2] as a starting point. Figure 1 below visualizes this seven-step process.

Figure 1 - Obsolescence Risk Assessment Process Best Practice

Although this process has been defined for managing hardware obsolescence, it also is an excellent basis for developing a process for managing the obsolescence of software. In the following we show for each of these steps how it can be applied to the specific software environment of the ALMA observatory. It is important to focus on a specific operational environment at hand in order to ensure that all the relevant information is taken into account, but the principle should be easily transferable to any other astronomy operations environment, or really any other environment of similar complexity and requirements. In the following sections we frequently refer back to the article presented in [2].

Step 1: System support plan assessment

Here we have to find out the time for which the system has to be sustained. ALMA is currently still in the lead-up to full operations and the life expectancy of the observatory is at least 30 years. For the purpose of this exercise it seems that we should not assume any end of life date by which any of the software components will no longer be needed, simply because the observatory as a whole will be decommissioned.

However, for some of the software that supports or depends on specific hardware devices, e.g. the ALMA antennas, frontend receivers, or the correlator supercomputers, there may already be more information available. We will need to identify all the parts that will influence software in one way or another, if they are either exchanged or become obsolete themselves. We need to also carefully consider the impact the replacement of particular hardware components will have on the software. This impact can be huge, if we think of new network, server or database architectures that would require major refactoring or redevelopment of major software components. Some may only have marginal consequences, if for instance ALMA antennas will be replaced, while others may even have no impact, e.g. if frontend receivers will be superseded by newer, better versions, that follow the same interface specifications.

We will need to produce a document that stipulates our assumptions in more detail and provide a comprehensive list of all the hardware components that may influence the obsolescence risk of our software. It will be a sizeable task at first, but subsequent reviews and updates will need to be done only at fairly long time intervals of two or more years, and should not consume much effort.

Step 2: Resources Planning

In this step we need to identify the resources that are needed to perform the obsolescence management, as well as the resources that are actually available. At this time ALMA has no resources, be it people, tools or budget, foreseen for this activity. This is of course no surprise, as the whole area of software obsolescence management appears to be very much unchartered territory. However, we believe that this is not a major issue, because software obsolescence management can also be seen as a specific activity in the whole area of software maintenance. Specifically it can be accredited to preventive or perfective maintenance, which is of course foreseen in our budget envelope as an important activity.

The management team of ALMA computing will therefore take on the task of leading and coordinating the effort, with the appropriate delegation to the software operations and the software engineering teams as required. Very importantly someone needs to be appointed as the *Software Obsolescence Manager*, who leads and coordinates all the work. In a globally distributed operations environment it is near impossible to find a single person who is familiar with all the various systems that form ALMA, but it appears most logical to appoint someone who is in the software operations group at the observatory in Chile.

It is not yet clear if specialized tools for maintaining an inventory and planning will be required, but we believe that, at least in the initial steps, spreadsheets will be sufficient. Initial – hands-on – experience will allow us to better identify the requirements for the potential need of a specialized tool.

It has been decided that there will be no additional budget allocated specifically for this task and the required effort will come from the regular software maintenance allocations. It is important to remember that this is only for the *planning* part of obsolescence management. Any sizeable effort that will result out of this, e.g. evaluating, testing and deploying alternative technologies, may very well require additional funding. This could then be done e.g. as an ALMA development project.

Step 3: Extract and filter bill of material

We need to break down the entire system that directly or indirectly influences the ALMA software into manageable portions. It is at the discretion of the Software Obsolescence Manager to define the lowest level of detail that makes practical sense.

It is very important to note that this bill of material (BoM) is not limited to software components, but must also include all the hardware components that can, by their own obsolescence risks, imply an obsolescence risk for the associated software.

A practical approach seems to be to distinguish between the following broad categories:

- Third-party software products: this includes not only software libraries and systems used at runtime, but needs to be extended also to development tools, such as Eclipse or Jenkins, or supporting tools, such as JIRA or Wiki.
- Commercial off-the-shelf (COTS) hardware, such as servers, network equipment, storage media, etc.
- Proprietary hardware produced and supported by ALMA, such as the antennas, frontend receivers or correlators.

In any case this will be a rather long list and it will be a tedious process to compile it. But is essential that this list is complete for the whole process to work. The good news is that subsequent reviews and updates of this list, which should occur at regular intervals of one to two years, will only be a relatively small effort.

Once the list has been completed it will be the Software Obsolescence Manager's duty to classify all the items into one of three risk groups:

- High risk indicates detailed attention – not necessarily action.
- Medium risk items need to be assessed, but at a higher level than the high-risk items.
- Low risk indicates that no attention is currently warranted, but this may of course change in later years.

This classified list will then serve as the basis for the detailed risk analysis and prioritization in the following steps.

Deviation: Risk analysis for hardware

Before we continue with the next steps, it is important to understand the way risk analysis is conducted for hardware, on which we then will base our own approach for software.

First one needs to determine the probability that a component becomes obsolete. This can be done by comparing the general availability of a component with the predicted consumption. Figure 2 illustrates this process:

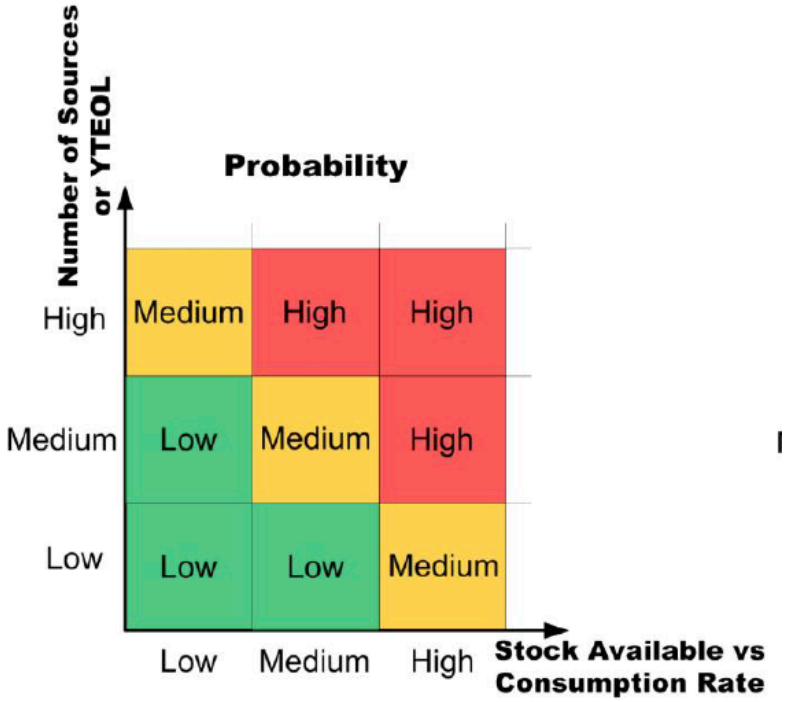

Figure 2 - Hardware obsolescence probability

The general availability risk is determined by one of two measures:

- The number of sources or suppliers through which the component can be obtained. A single source poses a high risk, two sources a medium one, and three or more suppliers indicate a low risk.
- The (to be expected) years to end of life (YTEOL). Less than two years mean a high risk, two to five a medium one, and more than five years indicate a low risk.

The numbers given in the above are of course only indicative and must be adjusted for each particular case.

The predicted consumption risk is determined by comparing the available stock to the actual consumption rate:

- Low stock and a high consumption rate mean a high risk
- Low stock and a low consumption rate mean a medium risk
- High stock and a high consumption rate also mean a medium risk
- High stock and a low consumption rate mean a low risk

The colored squares in the graph shown in Figure 2 then indicate the probability that a component becomes obsolete.

The actual obsolescence risk is now determined by comparing this probability with the impact the obsolete component would have on the system as a whole. This is illustrated in Figure 3:

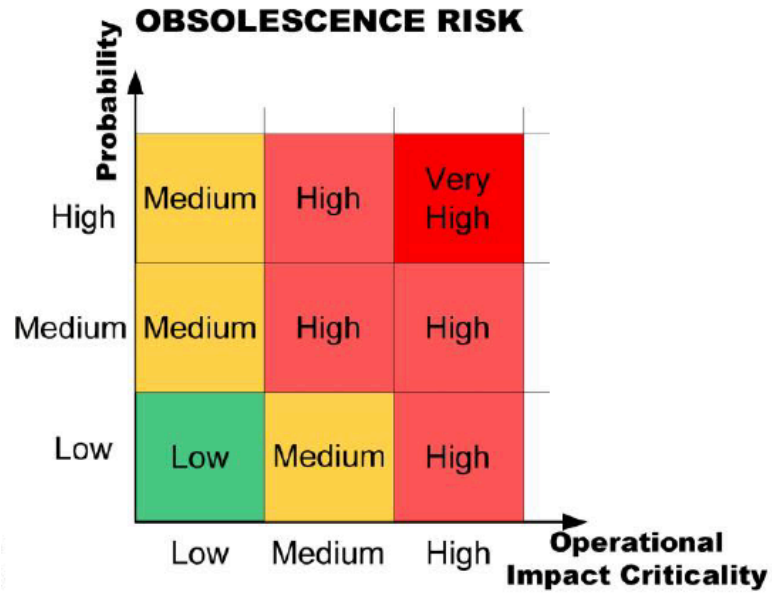

Figure 3 - Hardware obsolescence risk

The operational impact criticality is determined as follows:

- Safety critical components are rated as high.
- Mission critical components are usually rated as medium, but some could be rated as high, at the discretion of the obsolescence manager.
- All other components are rated low.

The colored squares in the graph shown in Figure 3 then indicate the obsolescence risk, providing four broad categories.

Before concluding this brief deviation into hardware obsolescence it is worth noting that this has not only been important in order to develop a similar model for software obsolescence; this exact methodology can of course be used for determining the hardware obsolescence risk for all the relevant hardware components in our bill of materials.

Step 4: Risk analysis for each component

First we need to define a suitable way of measuring the probability that a software component becomes obsolete. Here we cannot borrow directly from the hardware equivalent, which is based on a consumption and stock-keeping model. Software simply has different properties here: it can easily be reproduced and does not break by using it more.

Nevertheless software can become unusable or difficult to maintain for a number of reasons. We therefore invent a new term called the *liveliness* of a software component. A number of factors must be taken into account to determine the risk that decreased liveliness of a software component increases the probability for obsolescence:

- The number of providers or sources of a component or a technology: One provider or source means a high risk, two providers a medium risk and three or more a low risk.
- The predicted number of years to end of life or end of support by the provider of the software. Sometimes this information can be directly available from the supplier, for others we need to estimate this based on historical information or available predictions for comparable systems. Less than two years to end of life means a high risk, two to five years pose a medium risk, while more than five years means a low risk.
- The number of updates in the form of bug fix releases, or general updates or service releases available from the supplier: three or more updates indicate a low risk, one or two updates pose a medium risk, while less than one update per year points to a high risk.
- The user community: there is no absolute measure as some products are targeted for a wide audience with a huge number of users, while others are niche products that have a very limited amount of users. A better indication is the growth of the user community: a growing user community poses a low risk, while a stable community means a medium risk. A shrinking number of users points to a high risk.

There are a number of other considerations that have also to be taken into account. Technologies or products we use can become superseded by new technologies that are either more performant, easier to maintain or simply more cost effective. This by itself does not automatically mean an obsolescence risk, but can of course lead to a shrinking user community or an eventual discontinuation of a software product. Another important factor can also be that older – non-fashionable – software can become harder to maintain, not because the product itself becomes unsuitable for the application, but simply because it will be increasingly difficult to find skilled people to maintain it.

Using the guidelines presented here it should be possible to categorize all the software components in terms of their liveliness into one of the three risk groups, even though it may require some subjective judgment at times.

The liveliness measure, which substitutes the *number of sources or years-to-end-of-life* measure in the hardware area, would be entirely sufficient for determining the obsolescence probability if the ALMA observatory was a stable environment without any additional growth requirements. But this is of course not the case. So we introduce *ALMA's growth requirements* as the measure to substitute the *stock available vs consumption rate* measure used for hardware.

Once again, there is no simple measure for the growth requirements of an entire observatory, in particular when it comes to judge their applicability or impact on a particular piece of software. So for each software component we have to assess the risk the various factors that account for the growth of the observatory impose on each of the software components being assessed. The following brief list shows some of the factors that need to be considered:

- Increased data rates: there are predictions of how the overall data rate of the ALMA observatory will evolve, based on not only increasing operational efficiency, but also on new observing capabilities and/or hardware upgrades, and of course also increased use of ALMA data for archival research purposes as more and more project data becomes publicly available. All these factors can have impacts on various software components in the areas of storage requirements, throughput demands or memory consumptions.
- New features required: The need for new and improved functionality, especially for the software parts of ALMA that are not part of the control software driving the actual observations, has been huge and is likely to remain so for the foreseeable future. Every new feature increases the risk that various software components will no longer be suitable to support new requirements and demands.
- Usability improvements: ALMA will need to keep track with new demands and expectations on its user interfaces. While a very conservative approach may be feasible for internally used software, the demand to keep up with ever evolving user interface de-facto standards is much higher for the user facing software.
- Hardware changes: as mentioned before we will have quite a few software components whose obsolescence risk is directly bound to the obsolescence risk of specific hardware components. That's why our bill of materials is not limited to hardware only. Ideally we can simply get the required obsolescence information from the engineering department, but for certain hardware, such as computers, hard disks or network switches we may need to do this ourselves in order to predict the liveliness of relevant software components

There will be several other factors to be considered, but we believe that these considerations are a good starting point. So after summing these findings up once more by categorizing the growth requirements into our three-level categories, we produce use the graph as shown in Figure 4. We slightly deviate from the symmetrical graph used for hardware probabilities earlier, by declaring all categories with a low liveliness risk also as a low probability for obsolescence. If a product is well supported, there should be no reason for considering it as a candidate for obsolescence.

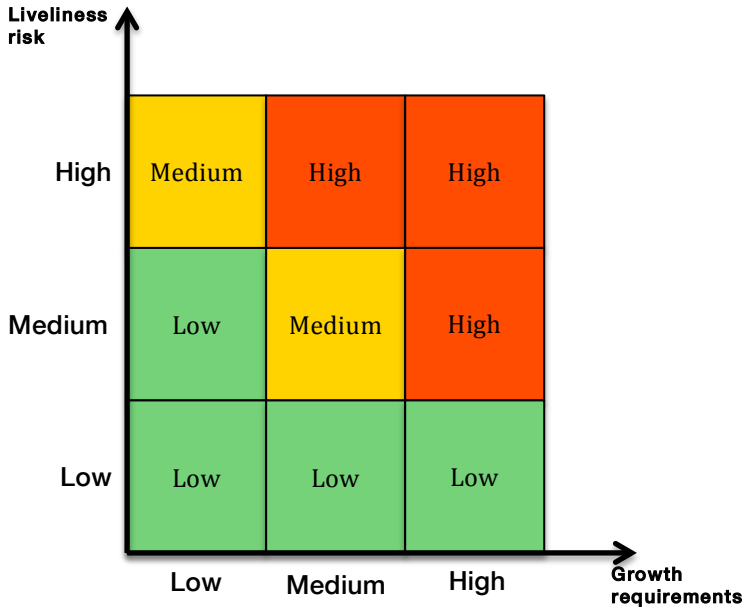

Figure 4 – Software obsolescence probability

Now we have to assess the operational impact criticality, by answering the question how bad it will be if or when a software component becomes obsolete. Following directly the hardware example does not quite work in this case. In particular it is important to note that the ALMA software under consideration here is by definition not suitable or designed for safety critical areas. There are of course safety critical parts in the ALMA observatory, but they are not within the scope of the software provided by the ALMA computing team and are hence excluded from our considerations.

A more suitable categorization for our needs is:

- High impact criticality: software that directly impacts end-to-end operations, including the entire observing project life-cycle from proposal preparation to data delivery and archive research. Impacts could be in many different areas, e.g. reduced efficiency, higher failure rates or reduced quality.
- Medium impact criticality: software that has minor impact on operations in the same areas as identified above, or impacts maintenance, e.g. by requiring more effort, the need to maintain third-party code or requiring additional expertise that is not currently available.
- Low impact criticality: all software that does not fit into the other categories by having very minor or only cosmetic impact on operations.

We can now use the graph shown in Figure 5 to determine the obsolescence risk for each software component under consideration by comparing the obsolescence probability, which we determined before, with the operational impact criticality. Here we also deviate from the hardware equivalent by declaring all components with a low probability also as a low software obsolescence risk.

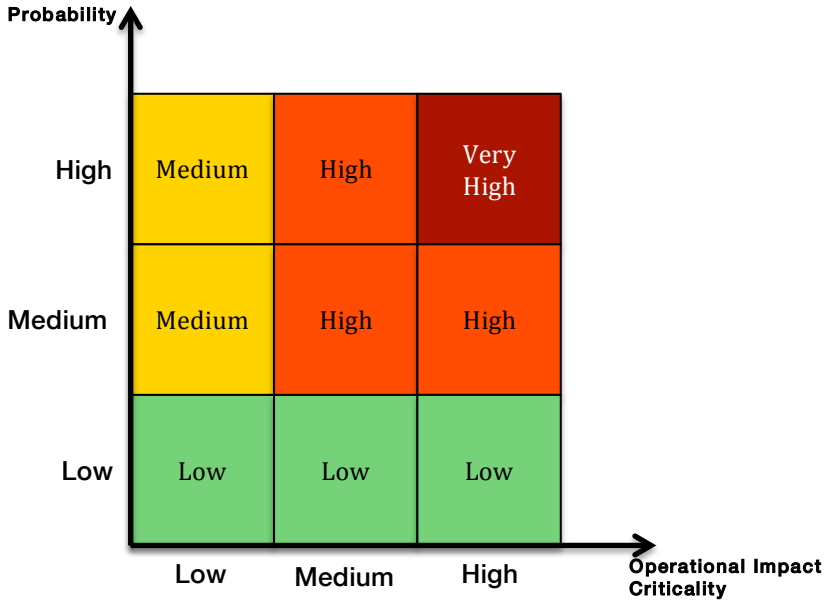

Figure 5 - Software obsolescence risk

Step 5: Components prioritization and mitigation decisions

The last step is perhaps the biggest chunk of work to be done. We have categorized all the software components into one of four risk categories, ranging from very high to low. Using this information we now need to further prioritize the components in each risk category, where one may actually skip the low or even medium risk categories. But it may also be useful to still rank them individually by a subjective impression of their importance for the observatory. Even for the high and very high categories it will require a panel of experienced stakeholders who are familiar with a wide range of technologies and operations processes to determine a meaningful ordered list of all the components within each category. The Software Obsolescence Manager will be responsible for organizing the panel and review, and will take the final decision in case of conflicting opinions.

Once the detailed prioritization has been done, mitigation decisions will have to be taken with a different approach for each of the risk categories:

- Very high obsolescence risk: action is required for each of these components. There are several possibilities here:
    - Replacement: this is perhaps the first and most obvious choice that comes to mind. Depending on how widely used a particular software tool is and how many interfaces it has to other parts, it may be a very costly and risky exercise. In any case, there should always be a cost/benefit analysis conducted. Sometimes it may not even be possible to replace software, simply because no suitable or affordable alternative is available.
    - Freeze code: if the software is stable and performant, simply staying on a particular version may be a sound option. There is, however, a not to be underestimated risk of increased internal workload, should maintenance be required after all. And it should be noted that this option will never be a permanent solution, but only allow us to gain some time in the hope that there may be better alternatives in the future or perhaps currently occupied resources become available.
    - Drop: a perhaps surprising, but not to be overlooked alternative is to simply ask if a component is still required or can be done without. Often tools are developed to satisfy a certain need at some point in time, but are then either superseded by other tools or simply no longer needed or only kept as nice-to-haves.

    - Partnering agreement with supplier: for heavily used software products on which big parts of the observatory depend, it may be too costly and risky to even consider a replacement or to maintain them in-house. Therefore a partnering agreement with the supplier or a specialized consulting firm may be a good alternative.
    - Insource maintenance: for certain third-party products, in particular such ones where the in-house staff has already gained significant expertise, taking care of future maintenance ourselves can also be a good option. This is particularly attractive for stable products for which the source code is available via open source licensing or other arrangements.
- High obsolescence risk: these items are treated in the same way as described for very high above, but with less time pressure. The Software Obsolescence Manager will decide on a case-by-case basis, if action should be taken immediately or can be deferred until later.
- Medium obsolescence risk: no action is necessary, but these items should be monitored frequently and carefully. They may be tackled if it makes sense by e.g. bundling the appropriate action with one or more of the higher risk items.
- Low obsolescence risk: these items should be checked against the filtered out ones and may be skipped in future reviews, always at the discretion of the Software Obsolescence Manager.

The Software Obsolescence Manager can now use this information and create the appropriate action items for the relevant teams to carry out the relevant measures. This is the point where the responsibility of the Software Obsolescence Manager finishes. The task of carrying out the recommended actions are not within the scope of his or her role, and also of course out of scope of this paper.

Step 6: Risk register update

All the collected data from the previous steps must be stored in a risk register. Ideally this should be a specialized database tool, but initially it is probably better to start with a spreadsheet. This is not so much for cost or time saving, but without any concrete experience it would be quite difficult to choose the best tool for the job.

So far we know that the risk register must, as a minimum, contain the following information for each component under consideration:

- Status: is the component obsolete or not?
- Expected lifetime of the component, if applicable
- Additional information about the component, e.g. interfaces with other components and dependencies
- Risk level: ranging from low to very high as identified earlier
- Details about the risk level and supporting data, including information how the risk was determined (e.g. liveliness, growth, impact)
- Planned strategies, if any
- Next steps if the component is obsolete

Based on these initial requirements, it seems plausible to use a general-purpose database with an easy-to-use frontend to enter and update the risk register, and a simple reporting engine. But it is entirely possible that a first full round of this process leads to a much wider scope with additional complexity.

Once established. it will be of paramount importance to keep the risk register up to date. This includes not only the regular updates as part of the yearly or so cycle through this entire process, but it is equally important to register any updates resulting from mitigation actions. Failing to do so will result in extra efforts to be spent in subsequent cycles to reestablish an up to date risk register.

Step 7: Review

The final step in our obsolescence management cycle is to review the outcome and decisions on a regular basis. In order to do so, we are planning to establish an informal review board, consisting of members of the management teams for computing, engineering and science operations. The Software Obsolescence Manager will report annually at one of our coordination and planning meetings to this board the work that has been done, an overview of the risk register, and the currently taken mitigation decisions. The board can then sign off all these actions and decisions, request additional actions or give particular charges to the obsolescence manager for the following period.

The decisions, actions and charges will be recorded in the regular meeting proceedings.

Conclusions

Once we have all agreed on this process, it will be most important to simply get started. There is a lot of room for discussions and changing details here or there, all of which may lead to a slightly different but not necessarily a better process. And it will take valuable time.

We also believe that writing a formal obsolescence management plan in a lot more detail than presented in here, and of course more customized to the particular operational environment, is an absolute necessity – in the long run. We rather suggest using the first round of actually doing the job, to also document the various steps in more detail and compiling the initial version of the management plan. Without some first hand experience, many questions would be left open, or the wrong questions would be addressed.

The initial plan to tackle software obsolescence management in this way was presented to all the stakeholders of the ALMA computing group about five months before the writing of this paper. ALMA observatory management has in the meantime confirmed that this is a high-priority topic to be addressed. The action to appoint a Software Obsolescence Manager should be concluded very soon, after which the tedious process of compiling our first bill of materials will commence.

Last but not least we want to note that following such a process to address software obsolescence also has some side-benefits. It will enforce a regular review of the system architecture, which is important at a time when many of the original designers of the system have moved on and the architecture has also deviated from its original design in several areas. And the need for change may also be an opportunity to spark new ideas, which are not strictly addressing obsolescence, but may be conducted as part of development projects.

We hope that this paper catches the interest of other projects and operational sites and that we can establish an active dialog to tackle this task in the most effective and efficient way. We would be equally excited if others find the information presented in this paper useful or to learn ourselves from the experiences others had and adapt our processes accordingly.

REFERENCES

[1] Peter A. Sandborn, “Software Obsolescence—Complicating the Part and Technology Obsolescence Management Problem”, IEEE TRANSACTIONS ON COMPONENTS AND PACKAGING TECHNOLOGIES, VOL. 30, NO. 4, DECEMBER 2007, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4383342

[2] F J Romero Rojo, R Roy and S Kelly, “Obsolescence Risk Assessment Process Best Practice”, Journal of Physics: Conference Series, 2012, Volume 364, 012095, https://dspace.lib.cranfield.ac.uk/bitstream/1826/7764/1/Obsolescence_Risk_Assessment-2012.pdf