Improving the Evolutionary Architecting Process for Embedded System Product Lines

Jakob Axelsson
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
jakob.axelsson@mdh.se

Abstract—**Many industries developing complex products based on embedded systems rely on architecting as a key activity. Furthermore, they use product line approaches to find synergies between their products. This means that they use a base platform which is adapted to different products, and the architecture of the product line thus evolves over time. In previous case studies we have seen that these companies often lack a defined process for the evolutionary architecting of these product lines. The contribution of this paper is to present such a process, which matches key characteristics of mature architecting practices. It is also discussed how this process compares to observations in industry.**

Keywords— architecture, evolution, process, embedded systems, product lines.

I. INTRODUCTION

In many companies developing technical products, such as the automotive industry, process automation, or tele-communication, *embedded systems and software* play an increasingly important role. The embedded systems have developed into a large number of computers with distribution networks and millions of lines of software. This increasing complexity leads to soaring development costs, and many companies strive to curb this trend by reusing software and hardware between products. Often, a *product line* approach is applied, where the same platform is used as a basis, with modifications to fit individual products and customers.

With a multiplicity of products and variants, the *architecture* is becoming very important and is a source of increasing interest for companies developing embedded systems. The decisions made by architects in the early phases influence many decisions made later on, and the architecting decisions are difficult to change further down the process [1]. With a poor architecture, downstream development activities will thus become much more expensive and time consuming.

We have previously done in-depth studies of the current architecting practices at a few automotive companies [2][3]. The issues we found were later validated also in other industrial areas where embedded software and systems play an essential role. Among the issues, we saw a lack of processes for architecture development, and the organizations had an unclear responsibility for architectural issues. Also, there was a lack of long-term strategy to ensure that legacy does not negatively impact future decisions, and a lack of methods to evaluate the business value when choosing the architecture. In short, the organizations rely on the performance and knowledge of individuals instead of on processes and methods.

Based on this information collected from industry, we find it plausible to assume that a mature organization would work with architecting of embedded systems and software mainly through stepwise refinement rather than large leaps. We call this an *evolutionary* architecting approach, in contrast with the *revolutionary* approach focusing on large but rare changes. Some of these companies state that they never again expect to start from fresh in their architecting, since it will be too expensive and complex. Instead, they will continue to refine their existing products. Some of the companies have tried to make major revisions of their architecture, but have failed spectacularly and been forced to revert to evolution of their existing solution.

The purpose and contribution of this paper is to present our findings on what such an evolutionary architecting process would look like. This has a value to industry because we have seen in our case studies that almost all companies lack a documented architecting process. Apart from the above case studies, the results are also based on an in-depth study of the actual evolutionary practices at an automotive OEM [4], and on a maturity model for evolutionary architecting that describes best practices [5].

The paper is structured as follows. In the next section, we discuss the nature of the evolutionary architecting process and its desirable characteristics, and also discuss its relation to the revolutionary process. In Section III, our suggested process for evolutionary architecting is described in more detail, and in Section IV, it is compared to the current industrial practice as observed in case studies. In Section V, related research is reviewed, and in the final section, the conclusions are summarized together with suggestions for future work.

II. EVOLUTIONARY ARCHITECTING

In this section, we present the context of evolutionary architecting as part of the overall product development process,

and also define what constitutes goodness for an evolutionary process. In other words, we look at the process from the outside.

A. *Process context*

The architecting process is closely related to other development activities, and for embedded systems the overall product development is often depicted in a V-model, as shown in Figure 1. The architecting activities are mainly related to the early design, where customer functions are transformed into technical systems. In addition, the architects deal with quality attributes which are non-functional properties of the architecture, both related to performance of the products and to things like modifiability over time. They also consider the life-cycle of the product, including future development, production, service, and operation.

The interfaces of the architecting process are highlighted in Figure 2. The triggering input is a change request from product planning (sometimes called business analysts, or similarly), usually to add a new function or enhance performance in some way. Other inputs are from various stakeholders on their needs and requirements, and the above mentioned quality attributes. Also, resources (mainly in terms of people) are required. The principle output is architectural prerequisites to system developers. These prerequisites are the architectural decisions that they need to respect in the detailed design, such as interface definitions, resource constraints, or design rules, and thus give an engineering context to their development work. In the figure, stakeholders include not only product planners but also representatives of all relevant product lifecycle stages as well as developers, in particular system developers, and testers.

B. *Evolution vs. revolution*

When a new platform is developed, there is an opportunity to do a major revision of the architecture. Changes that are typically introduced only at this time are a new communication concept, a different structure of the communication networks, or new basic software in the electronic control units (ECUs). Between these revolutionary steps, modifications such as the addition of a new ECU, a reallocation of some application software between two ECUs, or changing the connection of a sensor from one ECU to another, often occur.

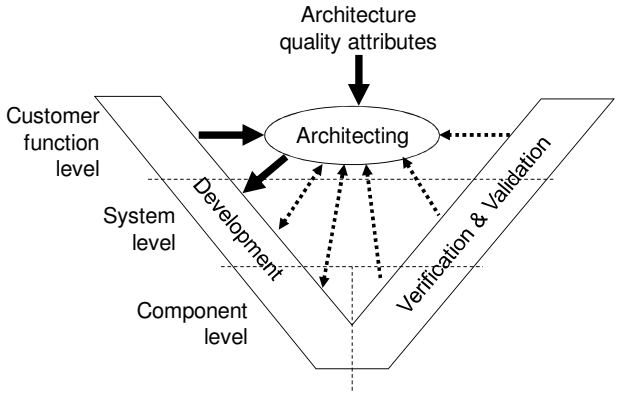

Figure 1. Development process context.

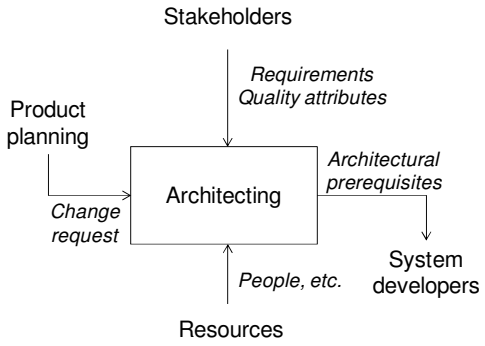

Figure 2. Interfaces of the evolutionary architecting process.

Some of the differences between the revolutionary and evolutionary architecting processes (RAP and EAP) are:

- RAP is done rarely as a defined activity or project, perhaps once every 5-10 years when a new platform is introduced and each time with a duration of a few years. EAP on the other hand is an ongoing process all the time.
- RAP deals with the architecture as a whole, considering all the functions and systems together. EAP usually deals with changes to a singular, or a few, functions or systems within an existing framework.
- RAP tries to dimension an architecture that can support many (yet unknown) changes as smoothly as possible for a long time, whereas EAP tries to implement a specific and concrete change in a specific architecture as efficiently as possible (while trying to assure that the resulting architecture still remains as flexible to future changes as possible, although this aspect is often less explicit in practice).
- RAP tries to predict future requirements, which is a speculative activity dealing with abstract information. One of the most important parameters is the expected rate of change which dimensions the flexibility needed. EAP deals with concrete requirements, functions and systems. This means that RAP must deal with uncertainty to a much higher extent than EAP.

With these differences pointed out, it should also be said that there are situations where some aspects of revolutionary nature is also conducted within EAP, simply because there is a need that was not foreseen at the time of the previous instantiation of RAP.

Many industries are currently heavily influenced by Japanese practices that are gathered under the label "Lean". One of the most cited aspects of Lean is *kaizen*, which stands for continuous improvement activities. The idea of evolutionary development is thus not new. However, Lean also contains the idea of *kaikaku*, meaning revolutionary change, and this has not been widely recognized in the western industry, nor has the interplay between the two been considered.

Within software development, the relation appears to be the opposite, with much focus on new development, and less on continuous improvement.

C. Desired characteristics

When defining the evolutionary architecting process, there are several characteristics that should be aimed for:

- *Efficiency*. The process should consume as little resources as possible, both in the number of architects needed but also minimizing the involvement of other stakeholders.
- *Effectiveness*. The process should deliver results with maximal value to their users. As can be seen in Figure 2, the main users are system developers, which indicates that the value created by the architecting process lies in the architectural prerequisites that has a large influence on the efficiency of down-stream development.
- *Timeliness*. The process should be able to deliver quality results in due time. To achieve this goal, it is essential to strive for short end-to-end processing time. The later architects start their work, the more accurate information will be available for them to base their decisions on. Still, they should ensure that they complete their analysis by the time output is needed by system developers.
- *Balance short-term and long-term*. Delivery to system developers in current projects is important, but a key role of the architects is also to ensure that the product line remains a useful asset also for future projects. Therefore, the architecting process must deal with both optimizing performance and cost of the system being developed now, and at the same time strike a balance with optimizing development efforts of future instantiations by maximizing modifiability. This means that strategic planning is an important part of the process.
- *Acceptance*. Architecting decisions can sometimes have a large effect on system development, and occasionally complicate a certain project to attain long-term benefits. Therefore, the process must yield results that are well motivated and understandable, for developers and projects to accept these decisions. As pointed out in [6], many architects spend about 50% of their time communicating with stakeholders.

III. BEST PRACTICE PROCESS

We will now open the box in Figure 2 to look at the evolutionary architecting process from the inside. An overview of the process is given in Figure 3. The main activities are:

A. Task planning and prioritization

The evolutionary architecting process is triggered by change requests originating from product planning, typically asking for a new customer feature or a performance upgrade.

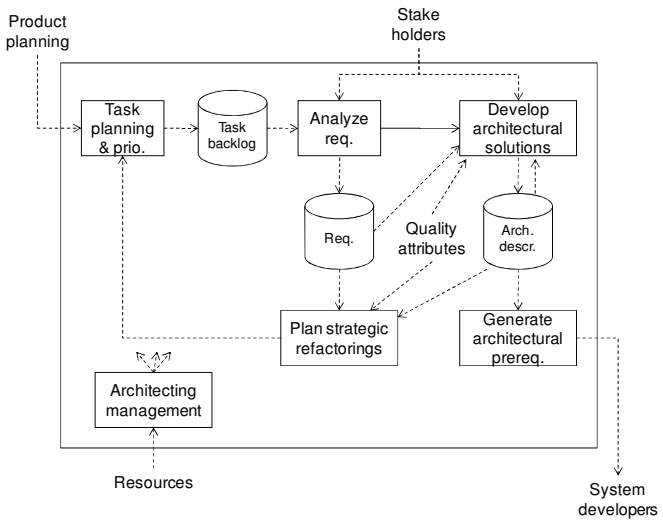

Figure 3. The evolutionary architecting process.

The change requests can arrive at any time, so this activity is event triggered. As a first step, the architects analyze the effort needed to deal with the request, and then place the request in the architectural backlog, which is a queue of requests waiting to be dealt with. The backlog is in priority order, and whenever an architect becomes available they pick the highest priority item in the backlog.

B. Analyze requirements

When an architect picks a change request from the backlog, the first step is to analyze the different stakeholders' needs, and transform these into requirements. In a mature organization, requirements are stored in a database and are version controlled, so in this activity architects update the database with new requirements related to the change request, and resolve any conflicts with existing requirements that could affect other products on the product line. Note however that architects are not concerned with the complete set of product requirements, but only those that are architecturally significant, which is a much smaller subsets.

C. Develop architectural solutions

The next activity is to develop an updated architecture that fulfils the objectives of the change request, while satisfying the new and old requirements. The main sub-activities are, as shown in Figure 4:

1) Synthesize architectural alternatives: Given the architecturally significant requirements, the architects derive alternative solutions that can be compared for merits. These do not have to be formally defined and described in all detail, but could equally be more informal sketches, as long as they contain the necessary information. The current architectural description forms a basis for identifying the changes needed in the evolution.

2) Evaluate architectural alternatives: The alternative solutions are evaluated based on the requirements, but also on their effect on the quality attributes.

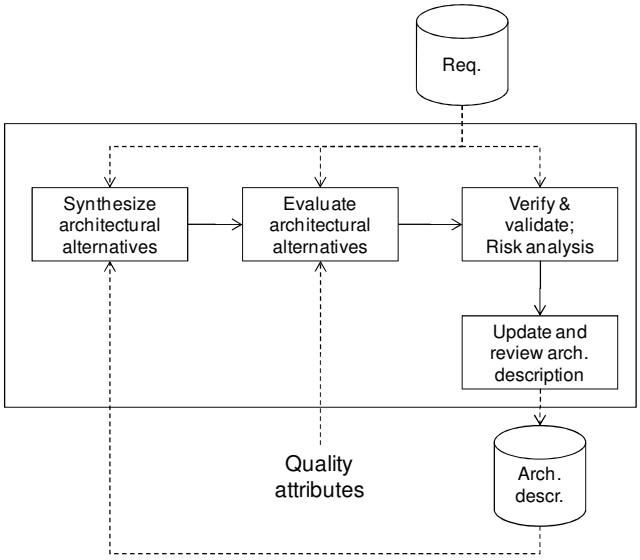

Figure 4. The actitity to develop architectural solutions.

3) Verification and validation: Verification is performed to check conformance to requirements, and validation goes back to stakeholder needs.

4) Risk analysis: Major uncertainties connected to the architectural decisions are analyzed, together with the consequences. If these are severe, mitigation actions are identified. These could be both technical, such as increasing tolerance in design, or in the form of activities, such as additional testing.

5) Update and review architectural description: The final step is to update and review the architecture description with the new solution. A mature organization uses configuration management to keep track of different versions of the architecture description, associated with different products in the product line, and to allow parallel work by architects dealing with different change requests simultaneously.

In practice, the development of the architectural solutions and the previously described activity, analyze requirements, often overlap in time, and are done iteratively. This is because what requirements are architecturally significant actually depends in part on what architectural solution is selected.

D. Generate architectural prerequisites

Whenever there is a need of architectural prerequisites for system developers on a specific product project, architects derive those prerequisites from the current architectural description. This activity is essentially time triggered, based on the project plans of each product project.

E. Plan strategic refactoring

Refactoring is the process of changing the architecture without modifying its external behavior in any significant way. This activity is motivated by the need to make the architecture optimal over time, so that it can support evolution of the product line. As an example, after adding a number of features, parts of the architecture are bound to become bottlenecks that need to be removed by adding more capacity. Refactoring is planned strategically by monitoring trends in key quality attributes over time, to predict when they will reach unacceptable levels. This, together with prognostics of change request rates, allows identification in due time of refactoring. This activity is hence mainly time triggered, since strategic analysis is a recurring activity. It is important to note that refactoring are here not based on vague wishes of architects, but founded in facts and measurements which increases the acceptability of the consequences. Since it deals with predicting the future, sensitivity analyses are used to assess the risks associated with wrong estimates.

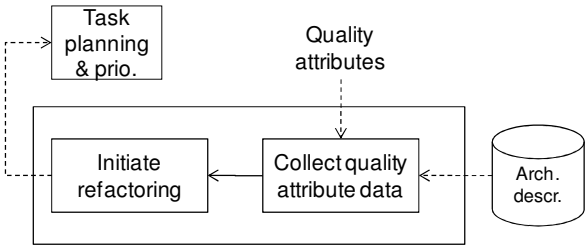

Figure 5. The actitity to plan strategic refactoring.

F. Architecting management

In addition, there is a need of a management process to coordinate and support architects. This sub-process includes the following activities:

1) Resource management: Each task dealing with a change request is assigned an architect who is responsible for taking it through the process.

2) Progress tracking: Tasks are followed up to ensure that they follow their plans, but also to gather process performance data that can be used for further increasing the quality and performance of the process.

3) Co-ordination: Usually, several change requests are being processed in parallel by different members of the architecting team, and this activity synchronizes their work to ensure that they do not create conflicting solutions. Typically, a periodic meeting is used for this, and possibly also for the previous activity.

4) Process development: The management process also is in charge of the continuous process development efforts to ensure that the overall evolutionary process remains efficient, which also entails making process measurements and planning periodic process reviews.

5) Organizational implications: A final responsibility of this sub-process is to deal with identifying organizational implications of architecting decisions. In essence, it is a consequence of Conway's law [7], which states that an organization basically always will produce systems whose

structure is a copy of the organization's communication structure. Thus, if the architects identify a need to change the technical structure, the organization will need to adapt to remain efficient.

IV. REVIEW OF CURRENT INDUSTRIAL PRACTICES

In this section, we will now relate our proposed process to what practices we have previously observed about how evolutionary architecting is done in industry. The empirical data is somewhat focused on the automotive industry, but also involves studies from companies in process automation, telecommunication, defense, and other industries. All in all, data from 16 companies are used, that were collected in seven different studies. Five companies occur in several of the studies, and thus are more influential on our results, where others only appear once. The data sets and the overlap between participating companies are shown in Table 1, together with references to publications describing the studies. In particular, the first data set is used, since it is the most structured and detailed set. (It should be noted that the discussion below is based on the full data from these studies, which is sometimes more comprehensive than what actually appears in the cited publications.)

We will now describe our observations for each sub-process in turn, and also describe what the consequences are of deviating from our process, in terms of the desired characteristics indicated in Section II.C above, namely efficiency, effectiveness, timeliness, balance between short-term and long-term, and acceptance.

A. *Task planning and prioritization*

When a new change request arrives, it is necessary to plan the architecting task. There is evidence that the interface between the product planning and development organizations is not always functioning [3], which directly influences the ability to perform task planning and to prioritize different tasks. A consequence of this could be reduced efficiency and timeliness due to long task planning time, but also acceptance in the results if there is no agreement on what the task actually is.

B. *Analyze requirements*

Most of the companies appear to collect stakeholder needs or requirements in some form. However, in the study reported in [9], it is noted that almost none of the companies elicited and documented architecturally significant requirements in an explicit way, and those requirements were only intuitively known based on the experience of the architects and developers. This is also observed in [3], where one of the issues is the lack of a process for requirements.

In [5], the same pattern is confirmed, where a translation from stakeholder needs to formal architecture requirements is almost non-existent, and an analysis of the requirements is even rarer. Since the requirements are not formalized in the first place, a natural consequence is that changes are not managed in a structured way, and we have not seen any organization that ensures traceability from requirements to architectural solutions.

TABLE I. DATA SETS AND COMPANIES INVOLVED.

Data set	Ref	Companies					
		A	*B*	*C*	*D*	*E*	*No. of other*
1	[5]	X	X	X			
2	[2]	X					
3	[3]		X		X		
4	[8]	X	X	X	X	X	1
5	[9]		X			X	8
6	[4]	X					
7	[10]	X	X	X		X	2

In the same study, some of the companies claim that they make trade-offs between requirements, even though they are informal. However, a further issue identified in [3] is that the desired balance between cost, time, and quality is unclear, which is one of the most fundamental trade-offs of all.

There are severe consequences of these weaknesses in the requirements process. Even though it can be very efficient to rely on expert individuals who work informally, one must bear in mind that the process discussed here is a repetitive one. This means that the requirements collected in the processing of one task will be input to other tasks in the near or distant future. Relying on the memory of individuals in that situation can lead to huge effectiveness problems where solutions are produced that do not match the full set of requirements. Also, timeliness can be poor, or at least very variant, if occasionally architects need to go back and elicit old requirements that were already known but forgotten.

C. *Develop architectural solutions*

We would have expected that the process where architectural solutions is generated would be fairly strong in practice since this is in some sense the core activity, but actually there are several areas where improvements can be made. We will now look into each area in turn.

1) Synthesize architectural alternatives: It is rare to work with more than one alternative solution in a structured way [10]. Instead, organizations appear to develop one solution, check if it is good enough, and if not generate a new alternative. If the right solution is actually found, this could be very efficient, but there is a large effectiveness risk in that the architects lock their thinking in a sub-optimal solution at an early stage.

2) Evaluate architectural alternatives: All our studies agree on the fact that companies do usually not evaluate the proposed architectural solutions in a structured way [8][9][10]. In fact, it has been observed that methods are lacking for evaluating the business consequences of architectural decisions [3] and that decisions are made based on experience and gut feeling [2]. Also, there is a lack of methods for trade-offs between short-term cost and long-term flexibility [4]. Many organizations do not even have quality attributes

defined [5], which would be an essential input in an evaluation.

Producing solutions without structured evaluation is very much like looking for a lost key under the lamppost. The solution has probably been produced driven by some concerns, and the architects' gut feeling is likely to only confirm that this is a good solution if other attributes are not considered.

Again, one could argue that it is efficient to save time by not doing elaborate analyses, but effectiveness is at stake. Also, acceptance by stakeholders outside the core architecting team is likely to be low, because they might value other properties higher. Finally, this is one key activity to balance short-term and long-term value of the architecture, and if this is not analyzed properly, it is likely that short-term cost will be prioritized, leading to costly rework on the architecture, or missed business opportunities, further along.

3) Verification and validation: Once a solution has been crafted, it is necessary to check that it fulfills the given requirements (verification), and also that it meets stakeholder needs (validation). Given the poor status of formal requirements, it comes as no surprise that companies seldom perform verification of the architecture, and that they lack routines for it. They are however somewhat better at validation, usually through reviews with stakeholders, but again routines are missing [5].

We believe it would be very beneficial for these companies to create routines for their quality assurance activities. When they are done in an ad-hoc fashion, they do not only reduce effectiveness but also efficiency, since it is hard to work with process improvement without a standard way of doing things. Also, a standard procedure tells everyone involved what their role and task is, and improves planning. Further, getting stakeholders routinely involved in validation reviews can build acceptance in the result at an early stage, or alternatively provide rapid feedback on needed changes.

4) Risk analysis: We have found no company that has a defined risk management strategy connected to their architecting process [5]. Occasionally, risk analysis and mitigation is performed, but this appears to be based more on individuals than on standard procedures. The lack of risk management can have large effects on timeliness, since the consequence of a risk occuring is often that additional activities are needed.

5) Update and review architectural description: Most companies document their architecture [9], and they have well-defined formats for doing so [5]. However, there is still room for improvements when it comes to routines for versioning, quality reviews of the description, and for ensuring that the documentation actually matches the product as it is built in the end. It is not uncommon in real-life situations that late changes are necessary, and in a stressful situation time is not always spent on going back and updating the architectural description. The consequence of this could be that later evolutions of the architecture starts with a faulty view of what the product looks like.

D. Generate architectural prerequisites

Once the architectural solutions have been generated, system developers need to be given prerequisites for their work. We do not have sufficient evidence on how this is done over a wide range of companies. However, it appears to be common that system developers are given a rather thick documentation of the complete architecture, and it is left to them to figure out what parts are relevant for them. This can be very inefficient since a lot of people need to digest a lot of documentation to find a small portion that is of interest. The practice of releasing a complete architecture also means that the processing time for the architecting team before they can deliver is long and this affects timeliness. It is a well known fact that processing of large batches hampers flow and that small batches are preferable. We would therefore suggest that organizations find ways of generating dedicated architectural prerequisites for each development team when they need it, i.e. basing the process on "pull" from the developers rather than "push" from architects.

We have also seen a tendency that architectural decisions are poorly motivated and that it is difficult to reach consensus [3] which seems to indicate that acceptance from system developers is not always satisfactory.

E. Plan strategic refactoring

We have found no companies that do refactoring as a standard routine, but instead they rely on ad hoc practices [9]. Many organizations try to identify improvement opportunities and use these to trigger redesign, but it is not an institutionalized procedure [5]. As noted above, many companies lack defined quality attributes, but even in organizations that have defined attributes, they do not follow how these change over time [4]. Instead of observing trends and extrapolating when a refactoring is necessary, they base these changes on the current situation. This leads to a reactive behavior, where insufficient time is available to plan properly.

A good refactoring process is instrumental in striking a balance between short-term and long-term value since it removes bottlenecks that will cause problems in the future. Basing refactoring decisions on data is also a way of increasing acceptance in these decisions. It is today often difficult for architects to gain management buy-in in refactoring activities which are essentially investments in the future with a fuzzy customer value. Also, for system developers, refactoring can cause changes and added work to their systems without any direct benefit for them, so this group of stakeholders also needs convincing arguments why it is necessary.

However, it is perhaps also one of the more difficult areas to improve in since there is a lack of established techniques. More research is needed to provide good methods for evaluation, but it is also a question of practitioners to define good quality attributes that can be linked to actual data. Far too often, architects use very general attributes that cannot be measured at all.

F. Architecting management

Architecting management is an area which has not been given very much attention in research, but we believe it is a key

enabler for improvement. As a starting point, these managers need to ensure that explicitly defined processes for architecting, such as the one proposed in this paper, are institutionalized. There is clear evidence from our studies that such defined processes are absent in many organizations [3][4][5].

1) Resource management. Almost all companies we have encountered have a team more or less dedicated to architecting, and the role of the architects is at least informally defined. We do not have sufficient evidence on exactly how resources are allocated to different tasks, but our impression is that it is not a big issue.

2) Progress tracking. Many organizations have ways to qualitatively follow up progress of the different architecting tasks, often through a weekly meeting [4]. This meeting is also used to take corrective actions when the task is deviating from its plan. However, the organizations do not have routines for handling such deviations, and they do not systematically analyze the root causes and remove these [5]. This affects the ability to consistently deliver quality, and thus is a threat to effectiveness.

Lacking are also more quantitative approaches to progress tracking [5]. This is important as an input to process improvement which can raise efficiency. It would be good to also follow up process adherence, but this is difficult since many organizations do not have a described process to use as a baseline.

3) Co-ordination. As mentioned previously, a weekly meeting is often used to co-ordinate parallel activities of different architects. This is a practice which appears to work well. However, there could be room for improvement, e.g. by using configuration management tools more systematically to discover when architects are working on the same part of the architecture.

4) Process development. We have seen few examples of organizations that work systematically to improve their architecting processes [5]. Again, the lack of a process description is an explanation for this. To some extent, the organizations do assess and plan process improvement needs, usually in an ad hoc fashion through a brainstorming session. However, we are not aware of any organizations that collect performance data on their architecting processes, and consequently they deny themselves of all possibilities to do statistical analyses for process improvement. This is a pity, because the rate of change requests flowing through the evolutionary process is often sufficient to make such an analysis meaningful. We believe that great benefits in process efficiency could be gained with a limited effort by improving these practices.

Many of the organizations have identified the need for training in their ways of working, and provide education for both architects and stakeholders. To some extent, this could reduce the consequences of the missing process descriptions, and also raise acceptance among stakeholders through a better understanding of the role of the architecture.

5) Organizational implications. Occasionally, changes in the architecture should also lead to changes in the organization to allow efficient development. However, this is an area where, in our experience, many companies have difficulties, and the organizational changes lag substantially in time. One explanation for this could be that the architecting teams are placed on a fairly low level in the organizations, usually parallel to the individual system development teams [10]. Having some kind of direct link to upper management for these issues would be beneficial, instead of having to go through managers that are likely to be personally affected by the proposed changes.

V. RELATED WORK

A generic process for creating and maintaining architectures is presented by Hofmeister et al. [11]. That process is based on a comparison of five different software architecture design methods. However, this process and most other in literature only deal with the core architectural design work, but the framework showing how to interact with other processes and how to resolve strategic planning is lacking.

It is also interesting to compare the evolutionary architecting process with *Agile* practices for software development. Although there are similarities, such as the emphasis on iterative completion of small steps at a time rather than long-lasting development of large chunks, or the use of a backlog, there are also differences that are related to the nature of the deliverable. Since architects produce, in the end, documentation to be read by other developers, rather than executable code, things like daily builds or test-driven development makes less sense. Also, Agile teams work intensively together, whereas architects tend to work in parallel on different change requests, making co-operation less intensive. Instead, architects communicate vividly with other roles outside the architecting team, often serving as translators.

A different approach is the work of the Software Engineering Institute (SEI) on a framework for software product lines [12]. It consists of a large catalogue of practices and patterns that an organization should follow, according to the authors, to be successful in software product lines. The role of the architecture is emphasized, but the scope is much wider than this, addressing an entire enterprise. Our approach is more limited in scope, focusing on architecting without any presumptions about other areas, and this is since we believe that an effort to improve architecting has a value, even without reforming an entire company. Also, the SEI approach is relying on long-term planning and predefined rules to evolve the product line assets, making it fairly heavy to use and intensive in secondary documentation. Our view is in this sense more similar to Agile, being much more lightweight and lean, and giving the architects freedom to modify the platform when needs arise. The architects thus act reactively on pull from product development rather than being pushed by plans.

From SEI come also the evaluation methods ATAM [13] and CBAM [14]. However, these methods are not so widely used in practice, and we suspect that they would be too

resource intensive to fit in the rapid evolutionary process that we envision.

VI. CONCLUSIONS

In this paper, we have outlined an evolutionary process for architecting embedded system product lines which goes beyond the scope of previously presented processes. We believe this process to be well adapted to the needs of many industries, such as those we have previously observed in various case studies. In particular, we have seen weaknesses in areas such as architectural requirements, evaluation of architectural solutions, risk management, refactoring, and process improvement.

One of the root causes for these deficiencies in current practice is the lack of an explicit process description for evolutionary architecting. Our proposed process can serve as a starting point for organizations, and once in place, it can be subjected to systematic process improvement and hence be adapted to the particular circumstances of the company. Through the process, industries can integrate architecting into their natural flow of product development based on product lines, giving benefits such as faster delivery of architectural prerequisites to the system developers, and less rework due to poor modifiability. Ultimately, the revolutionary approach may never have to be used again.

There are several opportunities for future work within this area. Above all, we would like to gather more data on the effects over time of a full implementation in industry of this process. Also, there is room for identifying new analysis methods to be used in some of the steps, in particular for dealing with strategic refactoring. Uncertainty is a fact of life in architecting, and a deeper understanding of what kinds of uncertainty occurs and how its effects could be minimized is valuable. Possibly, this could lead to the introduction of additional feedback loops in the process. Finally, a deeper analysis is needed of the cost of making the process improvements we suggest. We believe many of the changes are not necessarily costly, but it is always a question of stopping at the right level, and not to over-engineer the routines. Changes should be incorporated gradually, and be accompanied by continuous monitoring and process improvement with focus on efficiency and removing wastes. In the transition from an ad hoc to a structured process, the maturity model presented in [5] can serve as a valuable guide that suggests which step to take next.

REFERENCES

[1] Smith, P. G. and Reinertsen, D. G. Developing Products in Half the Time: New Rules, New Tools. John Wiley, 1998.

[2] Wallin, P. and Axelsson, J. A case study of issues related to automotive E/E system architecture development. In Proc. 15th IEEE Intl. Conf. on Engineering of Computer Based Systems, pp. 87-95, Belfast, Northern Ireland, March 2008.

[3] Wallin, P., Johnsson, S., and Axelsson, J. Issues Related to Development of E/E Product Line Architectures in Heavy Vehicles. In Proc. 42nd Hawaii International Conference on System Sciences, Hawaii, January 2009.

[4] Axelsson, J. Evolutionary architecting of embedded automotive product lines: An industrial case study. In Proc. Joint 8th Working IEEE/IFIP Conference on Software Architecture & 3rd European Conference on Software Architecture, Cambridge, UK, Sept. 14-17, 2009.

[5] Axelsson, J. Towards a Process Maturity Model for Evolutionary Architecting of Embedded System Product Lines. In Proc. 4th European Conference on Software Architecture, Vol. 2, Copenhagen, Denmark, August, 2010.

[6] Kruchten, P. What do software architects really do? Journal of Systems and Software, vol. 81, pp. 2413-2416, 2008.

[7] Conway, M. E. How do committees invent? Datamation, April 1968.

[8] Wallin, P., Fröberg, J., Larsson, S., and Axelsson, J. Practitioners' views on key issues and their solutions in system and software architecture development. Unpublished.

[9] Ozkaya, I., Wallin, P., and Axelsson, J. Utilizing Software Architecture for Managing System Evolution – Observations from Practitioners. In Proc. 5th Workshop on Sharing and Reusing Architectural Knowledge, Cape Town, May 2010.

[10] Gustavsson, H. and Axelsson, J. A Comparative Case Study of Architecting Practices in the Embedded Software Industry. In Proc. 18th IEEE International Conference on Engineering of Computer-Based Systems. Las Vegas, April 2011 (in press).

[11] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, Generalizing a Model of Software Architecture Design from Five Industrial Approaches. In Proc. 5th Working IEEE/IFIP Conference on Software Architecture, pp. 77-88, 2005.

[12] Northrop, L. M. SEI's Software Product Line Tenets. IEEE Software, p. 32-40, July/August, 2002.

[13] R. Kazman, M. Klein, and P. Clements. ATAM: Method for architecture evaluation. Technical report, Carnegie Mellon Software Engineering Institute, CMU/SEI-2000-TR-004.

[14] R. Kazman, J. Asundi, and M. Klein. Making architecture design decisions: An economic approach. Technical report, Carnegie Mellon Software Engineering Institute, CMU/SEI-2002-TR-035.