Perspectives on Software Product Lines

Report on Second International Workshop on Software Product Lines: Economics, Architectures, and Implications
Workshop at 23rd International Conference on Software Engineering (ICSE)

Peter Knauber
Fraunhofer Institute for
Experimental Software Engineering (IESE)
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
(+49) 6301 - 707 242
Peter.Knauber@iese.fhg.de

Giancarlo Succi
Center for Applied Software Engineering
Faculty of Computer Science
Free University of Bolzano - Bozen
I-39100 Bolzano-Bozen, Italy
(+39) 0471 - 315 640
Giancarlo.Succi@unibz.it

1 INTRODUCTION

Product line engineering is a recent concept and one of the hottest topics in software engineering aiming at synergy effects in software development. Diverse benefits like cost reduction, decreased time-to-market, and quality improvement can be expected from reuse of domain-specific software assets, several successful product line projects have been performed and documented [3]. Also non-technical benefits as result of network externalities, product branding, and sharing organizational costs have been observed.

Following the remarkable success of the "First International Workshop on Software Product Lines: Economics, Architectures, and Implications" held at ICSE 2000 in Limerick [1], this second workshop stresses more the non-technical, that is, business and organizational aspects of product line adoption and institutionalization. Another major topic of interest are product line tools, as tool support seems to become more and more critical for the success of product line approaches. Different tool concepts have been proposed and discussed during the workshop. Requirements for tools and respective solutions seem to become more concrete, maybe resulting from the fact that the technical concepts and solutions of product line approaches are better understood and can therefore be better supported with tools.

The strong emphasis on establishing contacts and giving experts and practitioners from academia and industry a platform for discussion has been continued during this second workshop.

Section 2 of this workshop summary describes the formal structure of the workshop. In Section 3, a short summary of the invited talk on issues and opportunities in product line research is given. Section 4 summarizes key points of the presentations of the workshop participants that were given based on their submitted papers that are fully available at [2].

In Section 5, the plenary discussion is described and its major lessons learned are summarized. Section 6 concludes this paper.

2 WORKSHOP STRUCTURE

During the starting session of the workshop in the morning, a detailed schedule for each of the sessions was defined. The decision was to limit the presentations of position papers to 15 minutes each in order to save more time for general discussion at the end. Also, it was decided to not have a dedicated panel at the end of the workshop and instead allow for plenary discussion.

Thus, the workshop was structured into the following parts:

Three discussion sessions allowed for short presentation of theoretical and practical issues of product lines. The first of these sessions was concerned with product line introduction and organizational aspects, as well as with extensions of current product line concepts. The second session addressed the handling of product line assets including their derivation for concrete products, change management, and tool support. The third session was concerned with modeling and evaluation of product line architectures and testing of generic assets. The most important topics of each session are briefly summarized in section 4.

In an invited talk, Paul Sorenson reported from two case studies exploring the potential of product lines and summarized some lessons learned as input for future research opportunities.

More details on this talk are given in section 3.

A final plenary discussion with the objective to define some product line research agenda concluded the workshop. According to this objective, the discussion was structured into the three parts "Where are we?", "Where ought we to be?", and "How do we get there?".

More information about the topics of this discussion is given in section 5.

3 THE INVITED TALK

Paul Sorenson gave an invited talk entitled "Issues and Opportunities in Software Product Line Research".

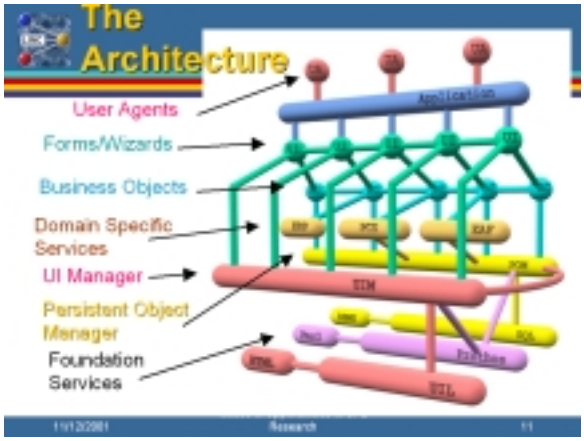

Figure 1: Product Line Experience (P. Sorenson)

He presented two case studies where the product line infrastructure was implemented in form of frameworks, issues that arose during their design and implementation, and lessons learned. In the first case study, an Engineering Application Framework was developed with the underlying goals to allow for easy evolution of the UI and workflow, to support deployment-related activities, and to anticipate refactoring of services. The resulting product line architecture is organized as set of domain-specific application frameworks composed of sub-frameworks where each sub-framework provides a small set of services (cf. Figure 1). The applications derived from this infrastructure are collections of services with varying degree of coupling. In the second case study, a framework was developed which can be used to build small client/server applications that run on the web (e.g., multi user games).

A couple of lessons learned can be derived from the case studies: Frameworks can be difficult to learn and understand, so a means of lowering the learning curve is needed. Tutorial sessions to show users what can be done with the framework, tools to investigate the framework and explore the use of hooks without having to understand the whole framework, and documentation including examples how to use the framework proved to be very useful. On the other hand, some research questions still need to be addressed. For example, it proved to be hard to decide if a framework is compatible with a particular application to be built or not. Another open issue is the evolution of frameworks: what constitutes a framework breakage and how can applications be protected from framework evolution?

4 THE SESSIONS

The topics addressed in the three presentation and discussion sessions can be clustered around four topic areas: product line introduction and organization, generic assets, architectural issues, and product line approaches. Due to timing constraints, it was not possible to allocate each of these topic areas to exactly one workshop session.

4.1 Topic Area 1: Product Line Introduction and Organization

Concerning this topic area, economic and organizational aspects of software product line introduction were addressed: when and where it is worth to introduce a product line approach in an organization, what are the expectations of the stakeholders concerned, how should convincing business cases look like, and can the risks of a product line transition be managed by following an incremental approach for product line introduction?

In his presentation, *Sholom Cohen* emphasized the need for a solid business justification, mainly based on a projection of costs for the current software development mode vs. costs of product line development. He argued to start with a data collection from legacy programs to provide a basis for comparison and then do a pilot study of product line development in order to collect data as basis for projected savings from product line development. As an example he presented a case study where three different reuse scenarios were analyzed: first, construction and then routine maintenance of an asset base for an initial set of domains, second, the later enhancement of that asset base, and finally, the later extension of the asset base into new domains. The potential benefits of the different approaches can then be illustrated with figures like Figure 2 which is taken from a case study.

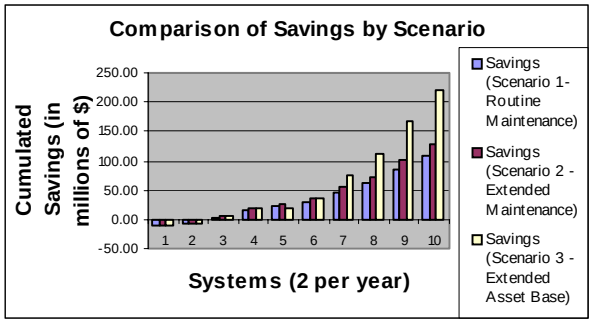

Figure 2: Comparison of Savings (S. Cohen)

Dirk Muthig argued to introduce product lines always in an incremental way. He presented some dimensions along which the incremental introduction can happen, two of which are software development life cycle stages and sub-domains.

Following the first dimension, stage after stage of the development life cycle would be transitioned towards product line mode, always encompassing both, domain and application engineering. Following the second dimension, one sub-domain after the other would be focal point of (full) domain engineering activities, thus getting revenues from the more promising sub-domains early while others are still handled product- (i.e., not domain-) specific.

In any case, following a non-incremental strategy implies changing products, processes, organizational structure, and the way people think at the same time, causing high upfront investment and uncontrollable risks. Instead, an incremental introduction enables to collect useful data to prove (and

quantify) success and benefits of the product line approach as well as to control risks and react early in case of problems.

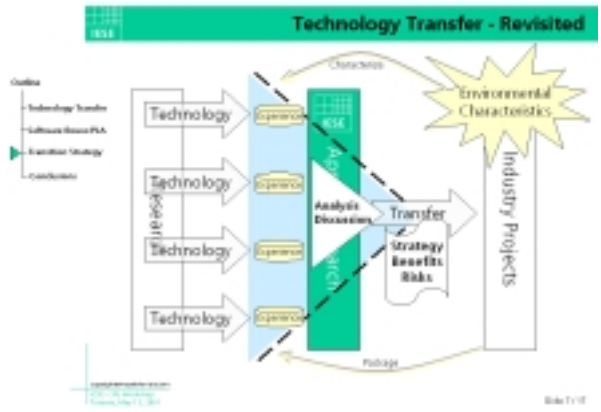

Figure 3: Dimensions for the introduction of PL (D. Muthig)

The paradigm of experimental software engineering in technology transfer based on data collected in the past (i.e., experience, cf. Figure 3) supports this incremental transfer.

4.2 Topic Area 2: Generic Assets

In his presentation, *Klaus Pohl* explained, how scenarios can be used to bridge the conceptual gap between solution space (represented by requirements) and solution space (as defined by the architecture), especially in product line development. Scenarios help to identify changes in requirements and differences between user-specific and product line functionality while at the same time reducing the complexity of product line description through the means of abstraction.

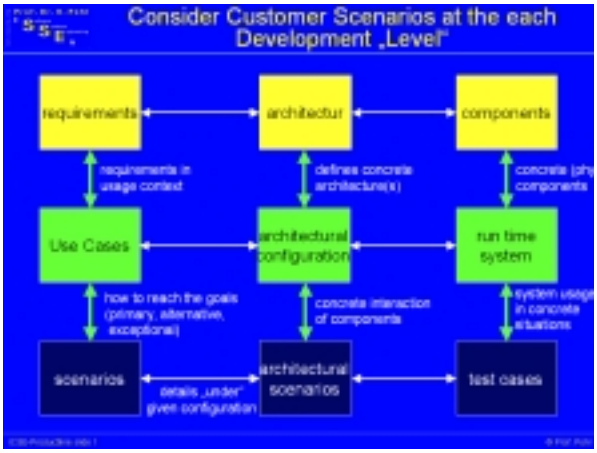

Figure 4: Customer Scenarios (K. Pohl)

Figure 4 illustrates the use of customer scenarios at each development level. The nine different artifacts used in this figure can be characterized at the meta level, including typed dependencies for each artifact type.

This way, traces among the artifacts can be modeled that help ensuring consistency when, for example, requirements are changing or, in the product line case, the generic artifacts are instantiated and adapted to user-specific needs.

John D. McGregor proposed a way to integrate test activities and their related assets in a product line context. Contrary to single system development organizations, product line organizations typically provide more continuity.

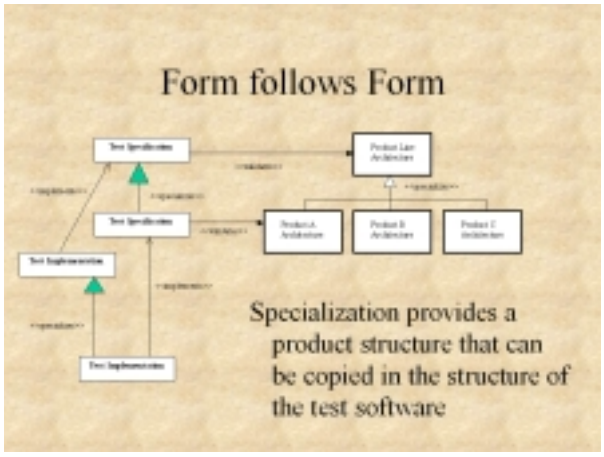

Figure 5: Integrating test activities in PL (J.D. McGregor)

This can be used to save testing effort while at the same time getting good estimates of the reliability of the products from the line. Figure 5 demonstrates how the test software should reflect the structure of the products derived from the product line. Since testing may require resources in the range from 50% up to 200% of the resources needed for the development itself, savings in the testing process due to process and/or test software optimizations can have significant impact on the overall performance of the development project.

In his presentation., *John MacGregor* argued in favor of a bottom-up product derivation process (cf. Figure 6) over a top-down parameterization process: Starting with the

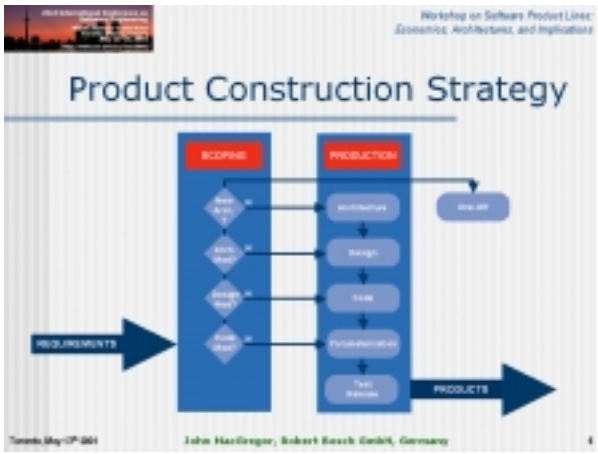

Figure 6: Integrating top-down and bottom up product line development (J. MacGregor)

requirements for a new product from the line one should first try to realize the new product using parameterization of code or to modify the code of the most similar existing product; the design would stay untouched. If that fails, a design adaptation has to be made, keeping the architecture intact. If that fails again, the architecture has to be modified in order

to adapt to the new requirements, in the worst case a new architecture has to be developed. In all cases, changes to higher abstraction levels have to be propagated down to the code.

An appropriate asset repository keeps the latest update of all assets touched, thus ensuring that all previously existing products from the line can be regenerated with equal or better quality without further intervention.

Giancarlo Succi presented Holmes, a framework supporting the different tasks for domain engineering while supporting multiple users, maintaining data and change consistency, and providing semantic support for relationship comprehension. Instead of re-implementing functionality that exists already in other, specialized tools like Rational Rose, Holmes allows for their easy integration.

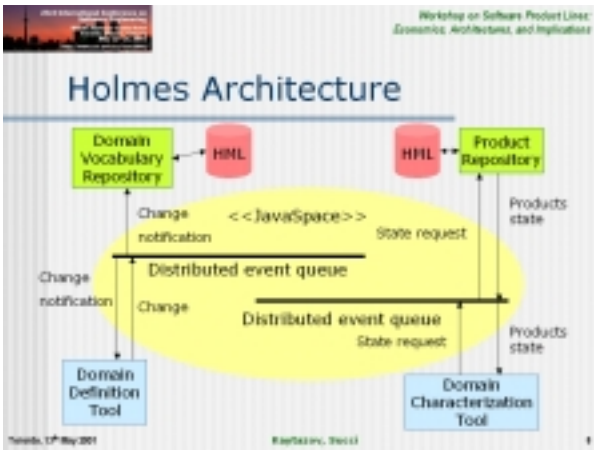

Figure 7: Architecture of Tool Support (G. Succi)

As shown in Figure 7, Holmes is based on JavaSpaces. Messages that can be passed from tool to tool are represented as objects together with their respective functionality. Queues are used to group related data, 3rd-party tools interested in one or more of these groups can then subscribe to the respective queues. Ongoing work is the realization of a critiquing tool written in Prolog that listens to (potentially) all queues and informs the user about potential flaws, independent of the tool creating the respective messages.

4.3 Topic Area 3: Architectural Issues

In his presentation, *Hassan Gomaa* demonstrated how to represent product line model variability in UML diagrams. He argued that UML is a well-known and established approach for modeling a software design, also for concurrent, real-time, and distributed applications. Analysis and design methods for UML exist. Thus, using UML for product lines will ease the transition from single systems to product lines. Use case diagrams are in a single system context used to define functional requirements for a system. For product line modeling, they can be used to model features and feature dependencies, including kernel, optional, and variant use cases.

Class diagrams can represent static/structural relationships, UML stereotypes are used to indicate kernel, variant, and optional classes. Statecharts and collaboration diagrams,

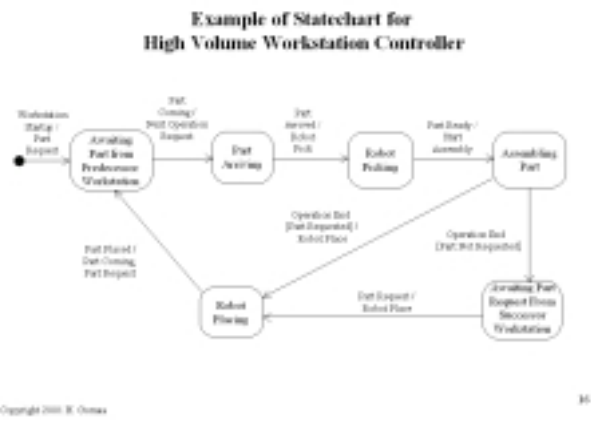

Figure 8: Using UML for PL (H. Gomaa)

again classified as either kernel, optional, or variant, are used to represent the dynamic behavior of the system (Figure 8 shows an example statechart). Future work should encompass the development of an UML-based software engineering environment for software product lines.

Andrea Savigni presented the concepts of Kaleidoscope, a reference architecture for command and control systems, and experience from two industrial case studies. The main property of Kaleidoscope is the separation of concerns, in particular, of computation, distribution, and strategy. The components are distribution-neutral, this way information alignment and information processing is kept strictly separate; separate entities, so-called projectors, take care of

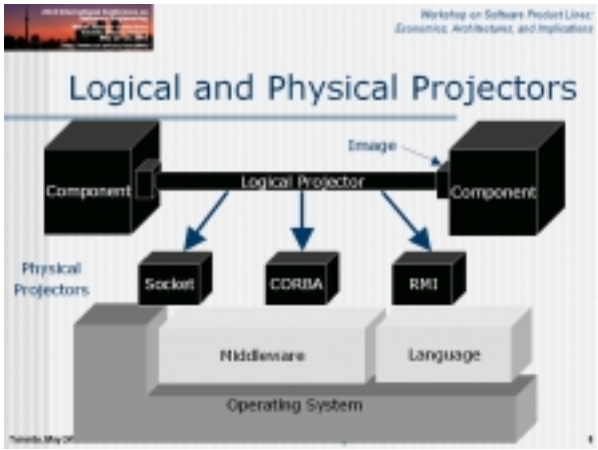

Figure 9: The Kaleidoscope Reference Architecture (A. Savigni)

the alignment (cf. Figure 9). The components are also strategy-neutral, that is, a separate entity takes care of triggering projectors for aligning and components for processing information.

The advantages of Kaleidoscope could clearly be demonstrated in two industrial projects. Experience from these projects shows that time-to-market can be reduced when configuration activities are assigned to domain experts (i.e., not to software engineers). Product line architectures,

as demonstrated with Kaleidoscope, are considered to be invaluable for large-scale reuse across applications and can be seen as the fundamental step in the transition from craftsmanship to actual engineering practice.

In his presentation, *David Rine* described requirements for and the design of a test framework product line for low earth orbit satellite constellations. This framework provides a common set of test threads that are required for the deployment, operations and maintenance, and anomaly detection and resolution phases. The test threads cover significant parts of satellite components that need to be defined and executed on the majority of satellite constellations with a minimum amount of data storage required.

The framework definition was done in four phases. In First, the focus area of research for the following activities was determined; this was necessary because it would have been impossible to achieve complete coverage of all possible constellations. In the second phase, the major nominal threads of testing necessary for each component on focus were identified. Then, a set of analysis criteria was determined and finally, each test thread was analyzed against these criteria. As a result, a common reusable set of test threads for multiple satellite constellations was developed under the strong resource constraints given by the domain.

4.4 Topic Area 4: Product Line Approaches

Juan Carlos Guzmán presented work and concepts from the embedded systems context. Usually, software and hardware are modeled using different notations and modeling techniques. For example, a variety of HDLs is used for protocol specification, special languages are used for software architecture description, and different programming languages are used for software coding. On top of that, lots of formal languages and mathematical notations exist. On the other hand, a high level of systematic reuse across soft- and hardware can only be achieved by a comprehensive design approach for complete systems.

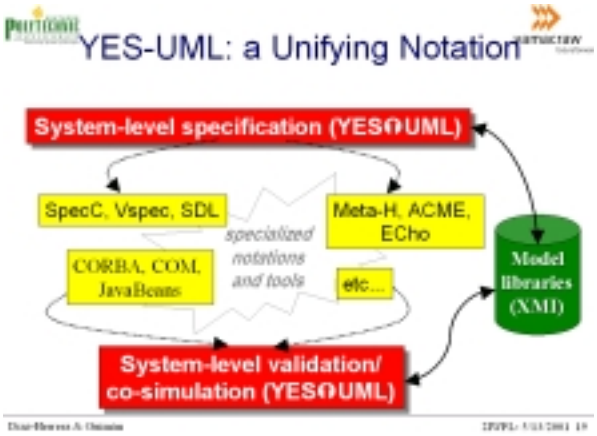

Figure 10: Structure of YES-UML (J.C. Guzmán)

For this purpose, he presented an UML extension called YES-UML that is developed and used in the Yamacraw project (see Figure 10). YES-UML shall contain concepts from description languages for hardware (like SpecC and SDL), software architecture (like Meta-H and ACME), and components (like CORBA and JavaBeans). All language concepts are represented based on XML. The start is made with SpecC, supporting tools are on their way. Using YES-UML as common representation for concepts and design of both, software and hardware, is expected to enable the use of existing software reuse approaches like product lines for complete system design.

In his talk, *Jeffrey Thompson* presented initial efforts in n-dimensional and hierarchical product lines.

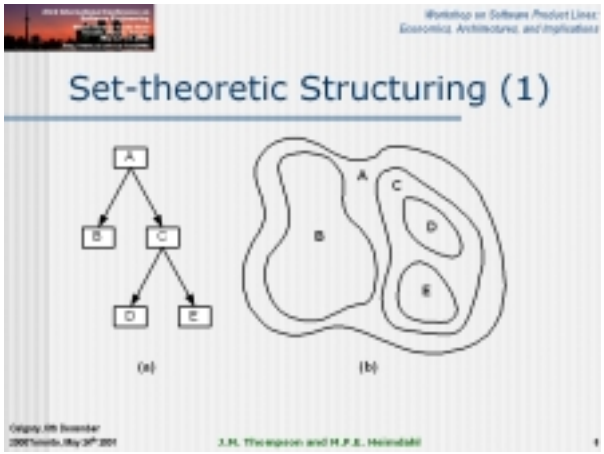

Figure 11: Formalizing n-dimensional and hierarchical product lines

N-dimensional product lines address the issue that many families are multidimensional in that they include several different, yet completely valid organizations that in turn may each have meaning in the context of the domain. In hierarchical product lines, sub-families can refine variabilities of the respective parent family or can add commonalities and variabilities unrelated to the parent. Such hierarchies can be represented like set structures (cf. Figure 11) and be handled using set-theoretic operations. Hierarchical product lines enable an easier extension of the family (the parent family can be much broader than otherwise), the possibility of refactoring in a way that sub-families have meanings in terms of the domain, and the possibility of an incremental product line introduction where one sub-family is implemented first, resulting in payoffs that can be used to finance further efforts in the implementation of other sub-families.

5 THE PLENARY DISCUSSION

A plenary discussion concluded the workshop. The discussion was organized in three major parts:

- Where we are
- Where we ought to be
- How to get there

The plenary discussion was heavily participated with contributions from several people, including -in order of appearance: Peter Knauber, Klaus Pohl, John D. McGregor, Andrea Savigni, James Paulson, Sholom Cohen, Dirk Muthig, Hassan Gomaa, Juan Carlos Guzmán, Mikko

Raatikainen, Paul Sorenson, Jan Bosch, Mike Bingle, Amr Kamel, John MacGregor, Jeff Thompson, Karl Reed. Moderator was Giancarlo Succi.

5.1 Where we are

Process modeling requires more investments in product lines, so managers have to be willing to accept initial investments that are likely to result in returns later.

Ideally, it should be possible to proceed incrementally, asking for small, incremental investments in exchange to a small, incremental transition to a product line approach.

However, not only incremental adoption of product lines is technically difficult, and does not make easier to convince managers, nowadays with very short term objectives to pay for something customers are not going to appreciate in the short term.

The scenario is not so new, as the same happened in the '60s and in the '70s with the moving to structured programming.

And the one thing we can be sure about and that we have learnt in the last 15 years of research on product lines and reuse, is that management support is essential.

The *good news* is that now software is viewed more as a product, and not any longer as just code. Companies are now realizing this, without really providing the essential support from the product, the product is not going to be of an adequate quality.

The *bad news* is that still among researchers there is no consensus whether product lines in software require more investment than in other disciplines, such as hardware or manufacturing.

A possibility to avoid stalling could be to start introducing systematically product lines in domains prone to product line approaches, such as those of component based or parameterisable.

More business case analyses are required, insulating the different costs occurring in product lines, outlining all the different benefits, including quality, time to market, lower know-how turnover, … Altogether, .the need for more accurate process models is the need for more costing and business advantage models.

It is interesting to analyse the connection between **product lines and frameworks**. So far, often people have made the claim that there is an intrinsic link between product lines and architectures. However, this is not so obvious. Moreover, trying to extend the framework into a product line may result in a breakage of the structure of the framework.

Testing for product lines should become more systematic. It should be performed right from the beginning of the process. The issue appears to be on how testcases can be derived. Formal documentation is required for an appropriate testcases. The level of documentation overhead depends on the desired quality. But developing test cases is itself a fairly lightweight work. Most of the automation required by practitioners is already present in existing tools.

Novelties in the area of product lines come from the ideas of **variation points** and **hotspots**, which have been popular since the early '90s.

An idea that is emerging is to use product lines as a tool for **safety critical** applications, or even to **certify** reference architectures for lines of safety critical applications.

5.2 Where we ought to be

Product lines are intended capture the evolution of software products and to last for a fairly long amount of time. Therefore, one of the hardest and most important aspects to consider is the **ability of product lines themselves to evolve** and change. Still, the only variability that they should have is that really needed to capture the required evolution, as a too general product line would end up being not manageable.

Also, it should be possible to determine when a given product line cannot be extended beyond a certain level, and that, for instance, a given functionality could not be added to an existing product line with a reasonable effort.

In simple terms, it should be possible to answer the question "How much more does this additional feature cost?"

Limited research has been posed on this aspect. Some work has been performed for the "Maintainability Assessment." It is an analysis focused on scenarios and maintenance profiles that enable to infer properties of the underlying system

Still, a better understanding is required to invest in, and produce product lines that are effectively usable.

It is also important to **perceive and appreciate the difference between evolution and variation**. Variation does not require a change of architecture. Evolution does.

A possible route could be the organization of the variation. Changes that are not taken into consideration in the variation points are really hard to manage. However, part of the problem is that it may not be known in advance what is variability and evolvability.

There is also a school of thought thinking that the problem is not evolution per se but the added complexity coming from evolution.

It is also important to improve the **use of compatibility** and incompatibility as means to develop open product lines. They can be profitably used both as means of developing open/close architectures, and as a tool to develop suitable economic models.

There are lots of other issues that require a deeper understanding. There is a gap between assets and products. It is still unclear how to bridge analysis assets and architectural assets. The longly-advocated paradigm shift from architecture based reuse to reuse based architectures has not occurred yet.

5.3 How to get there

It is hard to give advices in such an open area, where lots of promises have been made in the past year, most of which have not been kept.

It appears that **the creation of a database of codified experiences would be extremely useful**. Being able to systematize the existing experience into a single framework would create an infrastructure on which several open questions could be answered. This would be even more relevant if also industrial experience were to be used.

However, there are lots of issues related to the creation of such open database. Some relate to the quality of the data inserted in it. There is a non trivial problem of how to compare the data stored in it; meta-analysis could be a possible solution, but has not been used widely in software engineering. Moreover, suitable confidentiality agreements should be set with the companies from which the experience is to be extracted.

With such a database it should be possible to draw more definite answers to questions, such as the following:

- The use of domain isomorphism from requirement engineering and domain engineering to predict the likely variability in specifications
- The difference, if any, in the notion of changeability in time (evolvability) and in space (variability)
- The relevance of frameworks for product line architectures

6 CONCLUSION

Altogether, the workshop was large success, due to the quality of the submitted papers, the level of participation of the audience, and the profile of the panelist. Several positive feedbacks have been received; for this reason, we have decided to publish the papers as a collection in [1].

At ICSE 2002 in Buenos Aires the "Third International Workshop on Software Product Lines: Economics, Architectures, and Implications" will be held. We look forward a lot of papers and participants, to discuss the advancement in the discipline brought by the Y2K and to exchange our experience.

REFERENCES

[1] Software Product Lines: Economics, Architectures, and Implications, Peter Knauber, Giancarlo Succi (Editors), Proceedings of 1st ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, Limerick, Ireland, 2000, Fraunhofer IESE Report No. 070.00/E, 2000, available at http://www.iese.fhg.de/pdf_files/iese-070_00.pdf

[2] Software Product Lines: Economics, Architectures, and Implications, Peter Knauber, Giancarlo Succi (Editors), Proceedings of 2nd ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, Toronto, Canada, 2001, Fraunhofer IESE Report No. 051.01/E, 2001, available at http://www.iese.fhg.de/pdf_files/iese-051_01.pdf

[3] P. Clements, L. Northrop: Software Product Lines. Practices and Patterns, Addison-Wesley, 2002