Cloud-based Software Product Lines

Klaus Schmid¹, Andreas Rummler²

¹Software Systems Engineering, Institute of Computer Science, University of Hildesheim, Germany
schmid@sse.uni-hildesheim.de
²SAP Research Dresden, Chemnitzer Str. 48, D-01159 Dresden, Germany
andreas.rummler@sap.com

Please cite this publication as follows:

The corresponding BibTeX-entry is:
@INPROCEEDINGS{SchmidRummler12,
    author = {Klaus Schmid and Andreas Rummler},
    title = {Cloud-based Software Product Lines},
    pages = {164--170},
    year = {2012},
    volume = {2},
    publisher = {ACM},
    doi = {10.1145/2364412.2364440}
}

ACM, 2012. This is the authors version of the work. It is posted here by permission of the ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the 2nd International Workshop on Services, Clouds, and Alternative Design Strategies for Variant-Rich Software Systems (SCArVeS 2012) at the 16th International Software Product Line Conference (SPLC ’12), DOI: 10.1145/2364412.2364440.
Cloud-based Software Product Lines

Klaus Schmid
University of Hildesheim, Institute of Computer Science
Marienburger Platz 22, D-31141 Hildesheim, Germany
schmid@sse.uni-hildesheim.de

Andreas Rummler
SAP Research Dresden,
Chemnitzer Str. 48, D-01159 Dresden, Germany
andreas.rummler@sap.com

ABSTRACT
The traditional focus of Product Line Engineering (PLE) is on the customization of whole software solutions. So far, the combination of cloud computing with PLE techniques has hardly been discussed. In this paper, we discuss different approaches to cloud computing and their relation to product line technologies. We also describe both, specific opportunities and drawbacks, of these approaches.

We also provide a discussion of different combinations of these approaches as a way to combine their strengths.

Categories and Subject Descriptors
D.2.9 [Software Engineering]: Management – Productivity; D.2.13 [Software Engineering]: Reuseable Software – Domain engineering, Reuse models

General Terms
Design, Theory, Management.

Keywords

1. INTRODUCTION
Today product line engineering as a field is well established. Customizing software to the needs of an individual organization or stakeholder is a common need, which is successfully addressed in industrial practice [1], [2]. Initially conceived in the context of embedded systems, this principle is also well established today in the context of information systems, due to the strong need for context-specific customization of these systems [2].

With the increasing dominance of cloud computing approaches for information systems the question arises: how should the problem of variability be addressed in the context of cloud computing? Which strategies are available and which strategy is the most appropriate for a given context? It should be noted that we do not assume there will be a single best strategy. Rather we are interested in the specific benefits and drawbacks of potential approaches as a basis for selecting an approach in a specific situation.

So far, little work exists on cloud computing in combination with product line engineering. Exceptions either focus on some part of cloud computing like Software as a Service (SaaS) [3], [4] or they address indirectly related problems like selecting solutions for a cloud-computing environment [5].

Thus, while we will take the existing state-of-the-art into account, we do not aim at a bottom-up approach, which relies on the existing research work to provide a map of the various approaches as in a mapping study [6]. Rather, we aim at a systematic analysis of the potential of cloud computing to combine it with product line engineering technologies. We thus take a top-down approach in order to provide an overview, which is as broad as possible, and not limited by the techniques that are currently in use in research or practice.

In the following section, we will briefly discuss the basic concepts of cloud computing as far as they are necessary for our discussion. The following three sections will use the typical subdivision of cloud computing: Software as a Service, Platform as a Service, and Infrastructure as a Service. For each of them we will discuss the implications of using them in combination with a product line approach. In Section 7, we will summarize our results. Finally, in Section 8, we will conclude and discuss some open issues.

2. THE WORLD OF CLOUD COMPUTING
Cloud computing is not a homogenous concept. Rather, it consists of numerous different approaches, technologies and solutions. A common way to bring structure in this heterogeneous landscape is the sub-division into: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [7]. We will also follow this distinction here and discuss in the following sections the different approaches and their relation individually. In this section, we will first discuss these three different categories and highlight the main properties, which are relevant to our discussion.

Cloud computing can be seen as a form of virtualization of resources as a basis for providing services in an ubiquitous manner. The three different categories of approaches differ in terms of the level where virtualization starts and in terms of the nature and abstraction level of the provided functionality and thus form a virtualization stack as shown in Figure 1. We will start our discussion at the bottom and then discuss successively higher levels.

For each of the different categories we will mainly focus on the technical aspects due to our emphasis on product line engineering.
However, from a business model perspective a secondary aspect is that each of these approaches entails that the corresponding capability can be acquired on an on-demand basis leading to changes of the economic models.

2.1 Infrastructure as a Service
Infrastructure as a Service (IaaS) aims at providing virtualized computing resources. In this approach the real hardware (CPU, IO, memory, etc.) is virtualized, e.g., by providing several virtual machines on a single computer. Due to this low-level virtualization the individual virtual machines can even run different operating systems. While the basic principle is well-known for decades (e.g., multiple virtual machines on IBM /360 in the 1960s), it has been technically perfected (e.g., live migration) and became truly a service that can be provided on demand within minutes during the last decade.

Today, IaaS capabilities are typically seen in combination with standard PC-like hardware and are characterized by on-demand self-service capabilities where used resources can be extended and shrank within very small time periods. Important examples of this approach are: Amazon Web Services [8], Windows Azure [9], or OpenNebula [10]. Where the first two are commercial providers, while OpenNebula is an open solution. As the term infrastructure already indicates, the provided services respectively functionality can be assigned to an “infrastructure level”. This ranges from pure computing resources (with Amazon EC2 being an example) to basic services like data persistence (Amazon S3 can be named here as an example).

2.2 Platform as a Service
Platform as a service approaches the virtualization problem on a higher level. Instead of providing virtualized hardware, a platform – typically a programming platform – consisting of certain execution environments and supporting services like data storage, is provided (note that the border to infrastructure services is blurry). This platform enables software engineers to develop systems independent of location and resource usage, i.e., if there is more demand, automatic scaling allows that these requests are served as well. Platform as a Service ensures that this resource virtualization happens completely transparently from the point of view of the developer. The specific capabilities and restrictions depend on the individual platform.

While Infrastructure as a Service provides rather unspecific hardware resources, PaaS-solutions typically exhibit some domain-specificity. For example, Force.com is a platform that focuses on the development of additions to the well-known Salesforce CRM solution [11]. A borderline case is Google AppEngine [12]. While this still provides a very generic platform, it aims predominantly at the development of web applications. This is reflected by the specific properties of the platform. For example in Google AppEngine, programming of the platform is focused on handling individual web requests. In this case distribution and scaling is handled autonomously by the platform, but the platform also ensures that long-running requests are automatically terminated. Thus, the platform is hardly usable outside the realm of web application programming, e.g., for complex scientific computing tasks.

2.3 Software as a Service
This is perhaps the broadest and least precisely defined category. In this category specific (application) software is provided based on a service model. Thus, it could also be termed Application as a Service. In principle, this can be any kind of application software. Typical examples are office solutions like Google docs [13] or Office 365 [14], but also enterprise software like Salesforce [15] or Business by Design by SAP [16]. Even a web-based mailer, Google’s map service or a search engine can be regarded as SaaS from a technical perspective.

Due to this breadth (which covers basically any application area for information systems), Saas is very hard to describe exactly. In general, however, it means that the software somehow takes care of any necessary form of virtualization of resources. While in principle this can be handled explicitly by the application software, such systems are typically built on corresponding platforms that provide resource virtualization capabilities like abstraction from location and size of data stores.

3. A Running Example
As a reference example for our further discussion, we will take the hypothetical example of a business information system and infrastructure, which needs to be adapted to the specific concerns of individual customers. The possible dimensions of customizations could be manifold. We will discuss a broad range of possible customization requirements:

R1. Customization of layout and representation of the User Interface (e.g., adding logos or changing colors)
R2. Customization of the User Interface on a profound level like modifying the structure, removing certain parts or switching from a web-browser-oriented UI to a mobile user interface.
R3. Customization of the supported business processes.
R4. Customization of the underlying logic and the available business services. This may include content changes, modifying the service behavior, interface changes, or changes to availability (what services will be available).
R5. Customization of the deployment: where will services be made available, i.e., which entities are bound to which (virtual) resources.
R6. Customization of the software stack (platform), like modification of the middleware or the database to be used. This might be a requirement to enable compatibility with a previous solution used by the customer.
R7. Openness for third-party providers of services. Increasingly third parties are included in the ecosystem of the provisioning company and play an important role in the business models.

This list is on purpose rather broad to provide a discussion basis for our further analysis. We do not expect that they will all be relevant to a single information system and a specific customer; however, for each a context exists where this is a desirable customization for which a solution must be found.

4. Customizing IaaS solutions
In this section, we will discuss how in the context of IaaS-solutions customization can be performed. We will discuss what can be easily customized in such a context and what are the main advantages and disadvantages of a customization approach based on an IaaS infrastructure.

Customization Approach. IaaS only aims at virtualizing the (hardware) infrastructure. Thus, it is completely agnostic to any software running on top of it. It only provides some restrictions for low-level software like operating systems. Apart from this, an IaaS can use the available infrastructure like any other hardware environment. This leads to a tremendous flexibility, which can be exploited by the software engineer for customization.

As a result, the engineer can perform customizations like significantly altering the software stack. This may be up to and
including the operating system (e.g., Windows vs. Linux). Typically, it is also possible to explicitly control the deployment of a software stack with respect to the region (e.g., Europe vs. East Coast USA). Within the individual virtual machines deployed in an IaaS-environment typically a standard software stack is used. Thus, all existing variability implementation approaches and product line engineering methods can be used. As a result also all the customization requirements, given in Section 3, can be easily realized in such an environment, using the same methods and tools known from conventional PLE. This is in particular also true for software ecosystems as the complete freedom of development also enables a developer to provide an arbitrary base infrastructure to support ecosystems. However, this approach also does not directly provide any particular support for ecosystems.\footnote{Please note that some IaaS-providers like Amazon Web Services provide services, like payment services, which aim at bringing customers of the IaaS-based solution into the Amazon ecosystem. However, we regard customer-solution level payment actually as a PaaS-functionality, which is untypical for IaaS. Moreover, any IaaS-based solution is free to use any replacement for this.}

Benefits of an IaaS-based customization approach. The main benefit of IaaS-based customization is certainly its broad range of potential customizations. The fact that any existing variability management approach can be easily transferred to this situation also helps to simplify handling of such a situation. In addition, IaaS gives rise to further variability implementation approaches, which rely on the nature of virtualization itself. For example, solutions can be customized on the level of individual virtual machine images and then these can be individually deployed to as many virtual machines as desired. In this way scaling can be achieved even simpler than with non-virtualization-based approaches.

Moreover, to some extent virtual machines can be migrated between different IaaS-environments (e.g., OpenNebula mirrors some AWS functionality, VMWare machine images can be used in different environments). While migration between different IaaS-environments with different capabilities is a potential customization approach, it is limited as many management and supporting services are environment-specific.

In the example, in Section 3, this would mean that for each customer a set of specifically customized virtual machines could be provided, which is fully adapted to the specific needs of the customer. This approach would also not waste any resources from the perspective of an individual customer. For example, the size of the databases and their layout could be adapted to the needs of the software customization used by the customer, thus providing a customer with a fully optimized and adapted solution.

Drawbacks of an IaaS-based customization approach. The main strength of an IaaS infrastructure is also its major weakness. There is only little support for dealing with distributed computing issues available besides basic technologies like load-balancing. Thus, developers need to deal with many issues, especially those relevant to customization, on their own. Therefore, while all kinds of variability management approaches can be applied, also all kinds of variability management need to be handled explicitly by the developer.

Moreover, going this way may endanger a main benefit of cloud computing from a provider point of view: it leads to a fully tailored solution for each individual customer. This in turn hinders a cloud provider to realize the economies of scale he strives to achieve. This is due to the fact that the different customer installations may require different handling. Moreover, if one goes fully in this direction, i.e., providing each customer with a personalized installation, this leads from the perspective of the provider to a waste of resources as then no resource sharing among individual customer installations (except for the hardware level) is possible. This would be fully contrary to the basic concepts of cloud computing.

Summary. Thus, IaaS-based customization is probably only a good approach in very extreme cases like the following:

- **Very large installations**: if a customer continuously requires a significant amount of resources, resource sharing among multiple customers is not so important, anyway. Further, for very large and, hence, complex installations often specialized customizations may be necessary anyway.
- **Very significant differences among the systems**: If the individual systems for the various customers are necessarily extremely different, e.g., address different functionality, then it is justifiable to deploy these installations separately as resource sharing is hardly possible anyway.

## 5. Customizing PaaS solutions

In this section, we will discuss how customization can be performed in the context of PaaS-solutions. We will first describe the basic approach towards customization of a PaaS, before we discuss benefits and drawbacks of PaaS-based customization.

Customization Approach. A major question regarding PaaS-based customization is what kind of platform we are discussing. Is it a platform, which is basically a free programming environment that aims at a very broad range of applications, like Google AppEngine [12] (which aims at any kind of web-based application)? Is it a programming environment, which aims at software development in a limited domain, e.g., by coupling with a SaaS-solution like Force.com? Or is it a strongly domain-specific solution, which could also by itself be subject to a customization in a useful form? While the latter has received so far less interest in the wider range of discussions on PaaS, customization of platforms is today a very common approach in software development. We will thus discuss both the customization of rather broad platforms, which usually exist as a single runtime only, as well as the customization of domain-specific platforms, where we rely on results from the INDENICA-project [16], [17].

Domain-independent PaaS provide a basis for software development and are not expected to be customized themselves. Thus, in this case, if we develop software as outlined in our example, it is developed on top of a PaaS and is fully customizable. This software can be customized by any existing variability implementation approach, as long as it is in accordance with the constraints identified by the PaaS like the choice of implementation language.

However, within a PaaS typically the basic software stack like the available database, programming language, web server, etc. is fixed. The developer has no choice but to use this infrastructure. In some cases a few selections might be possible (e.g., alternative databases), but still the customer has no way to modify or extend this basic software stack.

However, there is also the possibility of modifying the PaaS itself. This can be possible for highly domain-specific PaaS. This, together with the question of how to achieve such customization
has been researched in the context of the INDENICA-project [17]. If we assume the implementation code of a PaaS is available, we can use all typical variability implementation techniques to derive a domain-specific instance of a PaaS. Together with additional techniques that address service variability or deployment variability this provides a broad arsenal of technologies that can be used to achieve customization. It should be noted, however, that today this kind of technology is not available on a large and practically relevant level. This also requires the capability of creating new PaaS-instances; if not explicitly supported this requires to go back to the level of an IaaS as a basis.

Benefits of a PaaS-based customization approach. Again, we will separate between the customization of applications on top of a PaaS and customization of the PaaS itself.

The first case of customization of applications on top of a PaaS leads to strong (nearly arbitrary) customizability of the applications on top of such a PaaS. This can successfully address requirements R1 through R4, as the necessary code can be arbitrarily changed. However, the underlying software stack is predefined and typically manages autonomously the deployment of the individual applications. A PaaS also supports the nurturing of a software ecosystem (R7), by allowing third-party providers to develop arbitrary applications on top of the PaaS. However, it should be noted with respect to (R7) that a PaaS-provider will typically try to control the overall ecosystem. Thus, he will try to draw the customers of the solutions on top of the PaaS into the ecosystem of the platform provider. As the freedom of the development within the PaaS would usually allow the solution provider to hinder this, PaaS-providers will typically use two types of strategies: a) license-based approaches and b) providing very strong support for their own ecosystem and none for any other, thus making it the preferable (simpler) solution for the solution provider to integrate with the platform ecosystem.

At the same time, the PaaS supports reuse and sharing of the software stack among the different applications. This leads to a rather efficient resource usage from the perspective of the PaaS provider. So in our example above, a PaaS could be established that provides the necessary resources like databases, which are shared among the customers, and the individual applications would be customized in a customer-specific way and run in parallel on top of the PaaS. Thus, application resources would not be shared among the applications, as application software is specifically tailored to the needs of each customer.

If we widen the scope to domain-specific PaaS as described above, this also allows increasing the resource sharing among applications. While there would actually be multiple PaaS-instances, reducing the sharing on the level of the software stack, the sharing on a higher level (domain-specific functionality) could be increased. The reason is that a customized PaaS could commit to more domain-specific functionality, increasing the suitability for a certain class of applications and enabling the sharing of the corresponding functionality.

Drawbacks of a PaaS-based customization approach. The drawbacks are directly related to the benefits. If we have a single PaaS and are not able to customize the PaaS, we cannot modify any decisions regarding the software stack, e.g., the choice of database or other infrastructure choices. The importance of this strongly depends on the kind of customizations that are required for the development purpose. In most cases of business information systems that need to be customized and deployed in the cloud, there will be very few cases that require a software technology stack customization. On the level of domain- and application-specific logic there does not exist a good basis for sharing. This is difficult both from the point of view of resource usage as well as from the perspective of maintenance complexity as for each problem the individual instance, in which the problem occurred, must be consulted.

These problems are reduced the more we move towards a domain-specific PaaS. (A mixed approach like the Force.com /SalesForce combination can be seen here as a middle ground.) As a consequence it makes sense to provide more customized domain-specific functionality. This reduces the amount of specific, non-reusable code. However, if we then need to customize this functionality, in order to address customization requirements like R4 or R5 and do this based on existing variability implementation techniques that result in code changes, this requires multiple instances of the complete PaaS. This might be appropriate in certain cases, but will be problematic in many others. At least it needs to be carefully evaluated based on the economic context of the provider.

Summary. We can summarize by saying that PaaS still provides a huge amount of flexibility, which will cover easily most typical product line situations. However, a pure PaaS-approach on top of which an easily customizable, but individual application is fielded, may easily lead to huge maintenance costs as many different instances of the application may be created. A major question in this context is: how much domain-specific functionality is appropriate within a PaaS. This is important, as the more domain-specific functionality exists, the higher the probability that a customization might lead to the need to customize as well the domain-specific functionality that is part of the platform. This in turn leads to the need to deploy multiple platforms using typical existing (code-modifying) customization technologies.

Of course, other technologies with a focus on runtime flexibility or deployment variation can be taken into account as well to address this problem. These would not necessarily require the full replication of the platform infrastructure, but have so far been only little addressed in the product line area. We will discuss this further in section 7.

6. Customizing SaaS solutions

In this section we will discuss customization issues in SaaS solutions. Again, we will first describe the basic approach towards customization of a SaaS before we discuss benefits and drawbacks of SaaS-based customization.

Customization Approach. If customization should be done in a SaaS-solution, this means that the customers should see different behavior and capabilities (actually different solutions) as described in Section 3. Again we can differentiate whether this is provided by a single solution (requiring runtime variability) or whether this may entail the customization of the SaaS, leading to several individual SaaS-solutions. The latter must not be visible to the end-user. There might be a common login followed by a redirect to the customized solution or a specific web-address per company or customer.

In the case of customizing the application itself, creating individual instances that are deployed individually, basically amounts to the situation, which we discussed already in sections 4 and 5. Thus, we will focus here particularly on the capabilities for enabling runtime customization and, in particular, on how a single SaaS implementation can provide customization capabilities. This basically amounts to runtime variability as we exclude the possibility of multiple customized instances.
per requirements R1 and R2 can be easily handled by means of parameters (for R1) or template processors (for R2). This isolates behavior over time [19]. The basic requirement is already well supported. For example, customizations of the user interface as it depends on the specific kind of customization that must be supported. For example, customizations of the user interface as per requirements R1 and R2 can be easily handled by means of parameters (for R1) or template processors (for R2). This isolates the customizations rather well from the main implementation.

Thus, runtime customization capabilities are needed that provide different customers the illusion of different systems while all these interactions are provided by a single integrated software system. It should be noted that this is a very specific variant of runtime variability, as potentially an application needs to exhibit simultaneously different behaviors to different customers as opposed to most work on runtime variability (or dynamic software product lines, DSPL) where a system changes and adapts its behavior over time [19]. The basic requirement is already well known in practice under the name of multi-tenant capability and has also received attention in the product line community [3], [4]. Several approaches also exist that support runtime variability, where flexible migration between runtime and development time variability is possible [20], [21], [22]. However, while they support runtime variability they follow the DSPL paradigm and aim at a complete, system-wide reconfiguration at runtime. It is an interesting research challenge to provide similar flexibility combining multi-tenant capability and design-time variability.

The simplest solution from a technical point to the development of multi-tenant solutions is to explicitly program the variation as part of the main application code. While this may be the most widely used solution, it loads the whole application with the need to support variability explicitly, leading to a rather bloated and overly complex implementation. Another approach, which is also discussed in [3], is to invoke middleware-support, by an appropriate infrastructure.

A major issue in this context is the de-facto isolation of the individual parts of the application that address the needs of the different customers. How strong this isolation needs to be, depends on the specific kind of customization that must be supported. For example, customizations of the user interface as per requirements R1 and R2 can be easily handled by means of parameters (for R1) or template processors (for R2). This isolates the customizations rather well from the main implementation.

<table>
<thead>
<tr>
<th>SaaS</th>
<th>Customization approach</th>
<th>supported variation</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Explicit programming</td>
<td>R1, R2 easily</td>
<td>Maximum resource sharing</td>
<td>Complexity in development</td>
</tr>
<tr>
<td></td>
<td>Template Engines</td>
<td>R3, R4 possible (but hard)</td>
<td>Simple management and maintenance</td>
<td>Restriction of customization capabilities</td>
</tr>
<tr>
<td></td>
<td>Workflow engines</td>
<td>Others difficult to support (except for explicit programming of the infrastructure)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| PaaS (applications) | customize application implementation (with arbitrary PLE technique) | Broad variability possible (R1-R4, R7) | Sharing of infrastructure | No customization of infrastructure possible |
| PaaS (implementation) | Customization of PaaS implementation | R1-R7 | Sharing of domain-specific functionality possible | No sharing of application implementation at runtime |
| IaaS | Customization of virtual machine possible (using traditional PLE techniques) | Any software can be freely exchanged and adapted | Broad range of variation can be supported | No sharing of software infrastructure |
|      | Deployment control (e.g. region) | All requirements R1-R7 can be supported | Any form of variability implementation can be applied | High maintenance complexity |

Benefits of a SaaS-based customization approach. Again, we focus on SaaS-customization in the sense of having a single main application that addresses all customizations simultaneously.

If such a solution is achieved, it brings significant benefits. As this covers the whole software solution, it leads to a situation where in principle all resources can be shared among all customizations. This leads to very low operational costs. These are further reduced by the fact that such homogenous software is cheaper from a maintenance and customer-support perspective.
**Drawbacks of a SaaS-based customization approach.** The downside of SaaS-based customization is that all variability needs to be squeezed into the same single piece of software and handled at runtime. This may increase the application complexity significantly. Moreover, this is only meaningful, if the various possible customizations are still strongly overlapping in terms of functionality and behavior. As a consequence typically the functional and behavioral variation is rather narrow and customer-specific functionality is not integrated (in a cloud setting) as it would impact the capabilities of all other customers. This limitation makes this approach today only appropriate in settings where very specialized functionality can be easily ignored and the required customizations can be well described and isolated from each other. This can be considered as an issue of prioritization, both from the provider as well as from the customer side. First of all it is important that in this area core functionality can be defined which most customers require. Second, many customers are willing to sacrifice very specialized requirements for the comfort of a cloud solution and perhaps lower cost. Third, the SaaS-provider is willing to sacrifice customers with specialized requirements and will not support them.

7. **Discussion**

In the previous sections, we discussed the repercussions of handling variability on the three main layers of cloud computing. We found that the specific layer on which variability is handled has a significant impact on the advantages and disadvantages that can be realized. As a means of summary, we provide an overview of the major results in Table 1.

Due to the differences in supported requirements and advantages and disadvantages it is more appropriate to discuss when to use which approach than which is the best approach: the main observation from our discussion is that the deeper the level of customization the more customization requirements can be supported (with IaaS counting as the deepest level) and thus it is possible to address all customer needs. Also the resource needs can be reduced per customer as only relevant software is required and no standardization on a broader than necessary software platform is required.

At the same time certain benefits can be accrued in the opposite direction from a provider perspective. Sharing of the cloud software leads to economic benefits from a provider perspective as the sharing among multiple customers leads to economies of scale. Moreover, standardization has a significant impact on maintenance costs and may significantly reduce them. Thus, a provider will usually aim at realizing customization on a rather high level. This may well go together with a certain inability to address the needs of certain customers.

This leads to a scoping question: as opposed to traditional scoping [24], it is not only necessary to answer whether a certain variability should be supported, but also if so, on which level. This leads to an interesting, additional complexity, which has so far not been addressed in product line research to the best of our knowledge.

In principle, it would also be possible to use a multi-layered approach in which a whole stack of cloud-layers is used. This is depicted in Figure 2. While we do not know whether such a form of realization has already been performed, it is known that at least providers like Google or Force.com use a multi-tiered approach where SaaS and PaaS solutions are offered and mutually enforce each other. This is shown in Figure 3. Based on our discussion it also becomes rather obvious why such a combination is useful. While SaaS-based solutions provide effectively customizable solutions within a limited range of variation, it does not well support software ecosystems [23]. This can be much better achieved using a PaaS-model. These partner-networks are today believed to be a major asset for any service-offering to take hold.

8. **Conclusions**

In this paper, we described the relation of the main categories of cloud computing with product line engineering and variability implementation techniques. We discussed the complementary benefits of these technologies and also described approaches to combine them.

The main conclusions are twofold: the main approaches researched so far in product line engineering lead to different implementations, which is undesirable in cloud computing as this means that multiple implementations need to be simultaneously maintained by provider personnel. This also reduces resource sharing from a provider perspective. Thus PaaS- and SaaS-solutions seem to be particularly relevant to cloud computing for most traditional product line engineering scenarios, upon migration to the cloud.

The second observation is basically that there is currently very little work that focuses explicitly on the customization (from a product line perspective) of cloud computing solutions. In particular, based on the previous observation this entails that these customization mechanisms need to support runtime resource sharing and enable complexity reduction, while at the same time supporting maximum resource and service sharing. From this perspective, we particularly expect that existing work on multi-tenancy will be very important in the future to cloud-based product line engineering.

In our future work, we will particularly focus on improving and developing variability management techniques that support runtime resource-sharing in cloud environments.

---

**Figure 2: Multi-Layer Realization of Customizations**

**Figure 3: Multi-Tier Realization of Customization**
ACKNOWLEDGMENTS
This work was partially supported by the INDENICA research project, funded by the European Commission grant 257483, area Internet of Services, Software & Virtualisation (ICT-2009.1.2) in the 7th framework programme.

REFERENCES
[18] INDENICA project, Deliverable 2.2.1, Variability Implementation Techniques for Platforms and Services (Interim), appeared 2011.