UNIVERSITY OF MANNHEIM

Assessing cloud development platforms – What Platform as a Service offers and what not

Oliver Gaß[1], Hendrik Meth[2] and Alexander Maedche[1,2]

[1] Chair of Information Systems IV, University of Mannheim
gass@eris.uni-mannheim,de
[2] Institute for Enterprise Systems
{meth, maedche}@uni-mannheim.de

Working Paper Series in Information Systems

No. 3

July 26th 2012

Assessing cloud development platforms - What Platform as a service offers and what not

John McCarthy, an early pioneer in computer science research, first formulated the vision of computing as a utility in a speech at the MIT Centennial in 1961: “If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility”. Cloud computing appears to be the latest and most mature materialization of this dream and has rapidly become a computing paradigm of great interest to the software research and practitioner community [1] [2].

Cloud computing promises virtually unlimited computing power and storage capabilities, a wide variety of application platforms and new service offers. Similar to the software stack on a local computer, the cloud is usually clustered into three major service levels: infrastructure as a service (IaaS) provides pure hardware and system software without any application level service using a pay-per-use pricing model [3]. Platform as a service (PaaS) furnishes a broad spectrum of elaborated application-level services and offers an execution and development environment on top of a cloud infrastructure [4]. It thereby enables the delivery of cloud services without the cost and complexity of buying and managing the underlying infrastructure. Finally, software as a service (SaaS) provides applications that run in the cloud and provide a direct service to the end user [5]. Developers can build and deploy cloud applications directly on IaaS infrastructure or use predefined capabilities of a PaaS solution (see figure 1) [4].

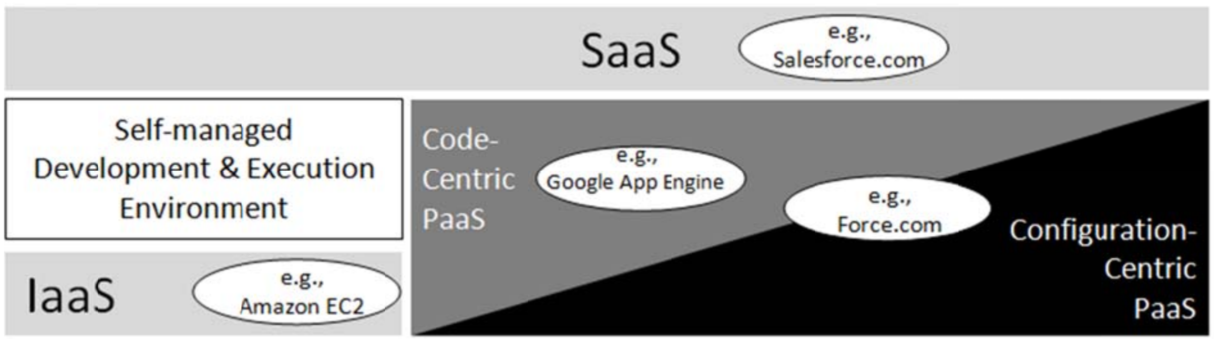


Figure 1: Alternative ways to build and deploy cloud applications

PaaS enables developers to implement or upload their applications in the cloud where they are accessible like regular web applications and are automatically scaled up when usage grows [6]. While some platforms focus on either development or execution capabilities, most vendors combine them into a single product offering [4]. One of the potential and also heavily promoted advantages of PaaS-based development is the increase in development productivity the systems are supposed to offer. PaaS includes several characteristics and capabilities which are assumed to have positive effects on development productivity: First, the entire development infrastructure including hardware, databases, operating system and the corresponding patches are provided as a payable service, freeing developers from laborious setup and maintenance activities [7]. Second, the platforms offer shared components such as pre-defined objects or built-in access and security features which can be leveraged. Third, the development is often supported by wizards and point and click features which are supposed to simplify the creation of applications. These configuration capabilities are usually provided via web-based instead of local development environments.

In the last years, a significant number of different PaaS solutions with an overlapping range of capabilities have appeared. Some PaaS offerings emerged as extensions of already existing SaaS products, while others are functional extensions of former IaaS offerings. Currently, the market of PaaS solutions is characterized by two development paradigms: Some platforms focus on regular programming of applications (but in a cloud environment), e.g., Google apps, while some others promote a configuration-before-coding paradigm requiring less programming (e.g., force.com).

While potential advantages of PaaS are intensively promoted in marketing brochures, most vendors provide little guidance about the limitations of their advertised solution. In order to decide whether to further develop PaaS at all and subsequently pick a specific solution, developers have to look beyond the shiny marketing brochures and identify the relevant technological details necessary to make an informed decision. Unfortunately, those details are often distributed over countless manuals, training tutorials or posts in the developer community. Finding the important information may become very time-consuming and frustrating, especially if one has no previous experience with PaaS technology and therefore does not know what to look for in the first place. PaaS technology, as any other technology, has its unique characteristics which make developing often easier, sometimes harder and in some cases even impossible. Developers must be aware of these unique characteristics in order to quickly identify the technical and non-technical details and make a well-founded decision whether a particular PaaS solution suits their own needs or not. The findings of this article help developers to arrive at an informed decision in two ways:

First, we use a real-world software development project to derive a general taxonomy of functional and nonfunctional characteristics of PaaS technology. Second, we apply this taxonomy to three contemporary PaaS offers to assess each one individually, but also learn about the current stage of PaaS technology as a whole. These findings help developers to create awareness of important characteristics and the strengths and weaknesses of current product offerings, but also help them to realize what PaaS as a whole can currently offer and what not.

A taxonomy to assess PaaS solutions

We developed the taxonomy based on data collected during a four month case-study conducted as part of a master’s course at our university: 19 developer groups comprising three master's students each were formed and asked to implement a cloud based alumni network solution. The key requirements for the alumni solution included:

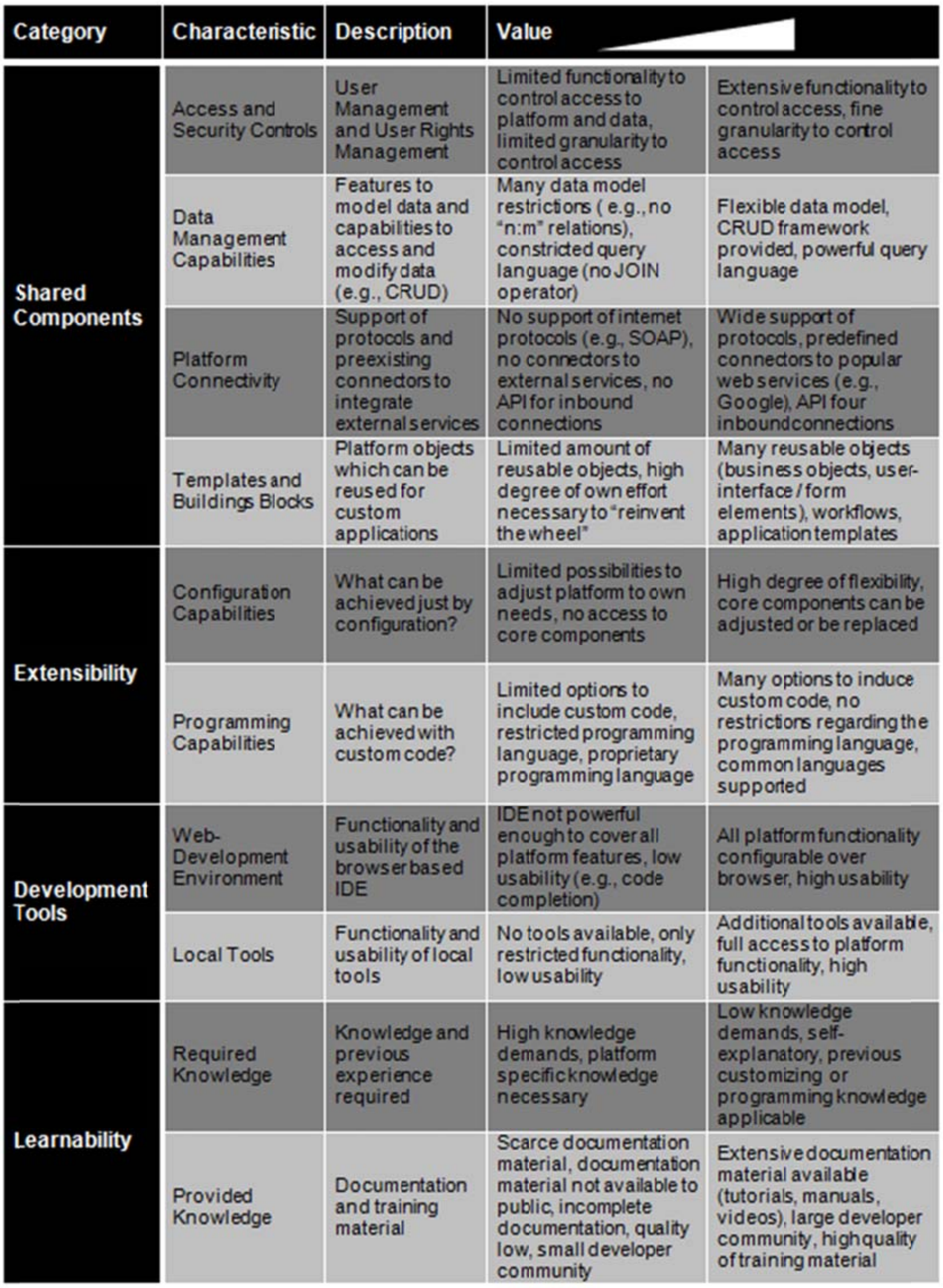
- self-service capabilities to allow each alumni to maintain their own profile
- social network capabilities to connect and communicate with other alumni
- event management functionalities to plan and organize alumni events and manage the guest list (including RSVPs)
- integration of external web services to provide a variety of additional features, such as weather forecasts for alumni events or routing information
- tracking of historic data to see the development of an alumni network

Each group was presented with the same set of requirements and randomly assigned to one out of three commercial PaaS products. The requirements were handed out in the form of listed features and additive sketches illustrating particular functionalities. Data was collected at several discrete points during the twelve weeks' timeframe of the software development project. At the beginning, we gathered control variables such as existing programming skills and previous experience with PaaS solutions. During the project, each team had to keep a developer diary, tracking every implemented requirement, the time needed and any platform-related obstacles they encountered. After the twelve weeks, the teams were also required to reflect on the project and hand in a project report. We advised the groups to focus on the perceived strengths and weaknesses of the used PaaS technology and explicitly track when a particular characteristic of the platform facilitated or hindered the realization of a requirement.

Each group submitted a working prototype and delivered a developer diary. In addition, we collected 19 reports with an average size of 25 pages. In the following qualitative data analysis, the research team evaluated the prototypes, developer diaries and project reports. First, we marked positive or negative quotes made in the diaries and reports. In a second step, we assigned codes of a similar abstraction level to the identified quotes. We then continued with clustering similar codes to derive functional or nonfunctional characteristics. Identified characteristics were only considered if they were sufficiently grounded in the documents via multiple instances (at least by half of the groups). Discrepancies among the groups, e.g., differing comments on platform capabilities, were analyzed and resolved. If groups reported missing capabilities, we cross-checked their statements with the official platform documentation and, if necessary, dismissed faulty claims. In summary, data analysis identified ten final functional and nonfunctional characteristics of PaaS technology which were either perceived as beneficial or hindering in a software development project. The identified characteristics refer to four major questions which arise whenever the decision whether or not to use PaaS has to be made (see table 1).

First, what shared components are provided by the platform? This question strongly relates to the degree developers can save effort by reusing existing components shared among all applications running on the platform. Four platform components were identified as relevant for developers: Access and Security Controls, which are functionalities to control access of users and the access to data (e.g., platform-wide user-management or discretionary, mandatory and role-based access controls). Capabilities which are related to the management of data, including predefined data models or automatically provided create, read, update and delete (CRUD) queries. Such capabilities are often characterized by restrictions regarding the modeling of data or missing capabilities of the supported query language (e.g., no JOIN operator). Platform connectivity functionalities assist in establishing inbound and outbound connections to other applications on the same platform or external web services. Typical aspects that need to be considered in this context are the availability of API access, support of protocols (e.g., SOAP) and connectors to popular web services (e.g., Google or Facebook). Templates and reusable building blocks summarize all elements provided by the platform which can be adjusted and used for individual applications. Such building blocks are application templates for particular use-cases (e.g., for a CRM use-case), user-interface components (e.g., calendar widgets), predefined object types (e.g., objects to store contact information) or workflow templates (e.g., order processing).

Second, how well can the shared functionality be adjusted to one's own needs and, if necessary, extended? The answer to this question is defined by the configuration and programming capabilities of the platform. Configuration characteristics relate to the capabilities of developers to adjust a platform just by customizing the parameters of existing functionalities. A very important aspect in this matter is the degree of flexibility the platform supports. For example, does a platform allow adjustments to core components of its architecture (e.g., relations between core business objects) or does it restrict changes to only secondary parameters (e.g., layout of elements of a form, help texts, etc.)? Programming capabilities describe the possibilities of developers to complement the functionalities of the platform with custom code. This is often necessary when requirements exceed the flexibility of existing components and developers have to use custom code to extend or even replace some components. Two aspects are important in this matter: First, what are the possibilities to add custom code? Are there predefined exits for custom code, e.g. exits for server-side validity checks, or does the platform build on a modularized architecture which allows the replacement of core components such as the complete front-end or CRUD methods? Also, what are the capabilities of the programming languages supported to implement custom code? Some platforms support established programming languages such as JavaScript or Java to implement extensions; other vendors however promote their own proprietary programming language with only limited functionalities. Also, most configuration-centric platforms offer several, yet less comprehensive, ways to include custom code.



Category	Characteristic	Description	Value	
Shared Components	Access and Security Controls	User Management and User Rights Management	Limited functionality to control access to platform and data, limited granularity to control access	Extensive functionality to control access, fine granularity to control access
	Data Management Capabilities	Features to model data and capabilities to access and modify data (e.g., CRUD)	Many data model restrictions (e.g., no "n:m" relations), constricted query language (no JOIN operator)	Flexible data model, CRUD framework provided, powerful query language
	Platform Connectivity	Support of protocols and preexisting connectors to integrate external services	No support of internet protocols (e.g., SOAP), no connectors to external services, no API for inbound connections	Wide support of protocols, predefined connectors to popular web services (e.g., Google), API four inbound connections
	Templates and Buildings Blocks	Platform objects which can be reused for custom applications	Limited amount of reusable objects, high degree of own effort necessary to "reinvent the wheel"	Many reusable objects (business objects, user-interface / form elements), workflows, application templates
Extensibility	Configuration Capabilities	What can be achieved just by configuration?	Limited possibilities to adjust platform to own needs, no access to core components	High degree of flexibility, core components can be adjusted or be replaced
	Programming Capabilities	What can be achieved with custom code?	Limited options to include custom code, restricted programming language, proprietary programming language	Many options to induce custom code, no restrictions regarding the programming language, common languages supported
Development Tools	Web-Development Environment	Functionality and usability of the browser based IDE	IDE not powerful enough to cover all platform features, low usability (e.g., code completion)	All platform functionality configurable over browser, high usability
	Local Tools	Functionality and usability of local tools	No tools available, only restricted functionality, low usability	Additional tools available, full access to platform functionality, high usability
Learnability	Required Knowledge	Knowledge and previous experience required	High knowledge demands, platform specific knowledge necessary	Low knowledge demands, self-explanatory, previous customizing or programming knowledge applicable
	Provided Knowledge	Documentation and training material	Scarce documentation material, documentation material not available to public, incomplete documentation, quality low, small developer community	Extensive documentation material available (tutorials, manuals, videos), large developer community, high quality of training material

Table 1: Taxonomy of functional and nonfunctional characteristics

Third, what tools are provided for development and how well do they work? This question relates to one key characteristic of PaaS, namely that applications are predominantly developed in a browser based IDE. Even though most vendors promote their web-based IDE as the major development tool, previous experience with web-based user interfaces indicates that the latter still has some conceptual disadvantages compared to local solutions [8].

Therefore, the availability and the capabilities of additional local tools, for example Eclipse based IDEs or tools for data management, also need to be considered.

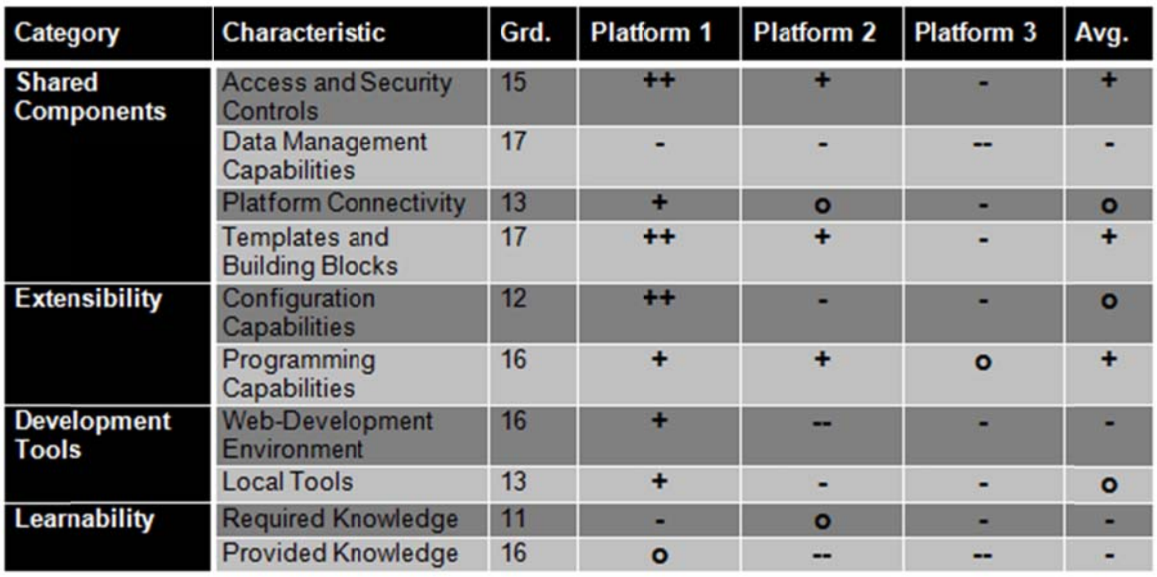
Fourth, how much knowledge is necessary for development and how easy can this knowledge be acquired? The last question relates to the learnability of the platform. The learnability addresses the issue of how much pre-existing knowledge, or previous experience, applies in the context of the platform. If the required knowledge is not available, either because the developer is inexperienced or because the platform is unique apart from common customizing or developing paradigms, the aspect of how the necessary knowledge can be acquired arises. Typically vendors provide documentation material in the form of manuals and tutorials. Developers also benefit greatly from available developer communities which help solve specific problems which are not addressed by the official documentation.

A review of three commercial PaaS products

Having the general taxonomy on hand, we were eager to determine the current state of practice in PaaS technology. In order to do that we looked at each platform separately and calculated how well developers assessed the implementation of a particular functional or nonfunctional characteristic. This was done for each group and characteristic by weighing the negative and positive comments. The resulting values were then normalized to a discrete scale with -1 for predominantly negative comments, 0 for balanced comments and 1 for predominantly positive comments. The normalized opinion values of the groups were then summed up to calculate the average opinion value for a given characteristic and a given platform. Afterwards, we mapped the resulting averages to a 5 point scale (++, +, o, -, --) to emphasize that our values merely represent a general tendency rather than a particular number value. The industry average was calculated based on the averages of the three platforms in each characteristic and then mapped to the same point scale.

Table 2 displays the value of each platform and the average for each characteristic. In summary, the table reveals significant differences in particular characteristics among the platforms (e.g., reusable building blocks), but also unveils common strengths and weaknesses across all three solutions (e.g., DBMS). However, what are the reasons for these significant differences in developer evaluation? The next few paragraphs look into each characteristic and each platform separately to determine the design decisions which lead to a positive or negative evaluation.

As described earlier, all three platforms provide ready-to use, shared components to speed up the development process. One of these components is *access and security controls*. For applications which do not have very elaborate access and security requirements, the features provided by PaaS applications are often sufficient. In contrast, if an application is built from scratch without using a platform, the provision of according features is a standardized activity, which nevertheless requires significant development efforts. In the case of PaaS usage, these kinds of commodity features are already sufficiently provided by the platforms, allowing developers to concentrate their efforts on more challenging and differentiating functionality. Respectively, developers graded those capabilities overall positively. All platforms apply well-known concepts from enterprise solutions by regulating access based on user profiles, user roles and user accounts. Differences exist in the level of detail of how access can be restricted. Platform 1, which received the best feedback, allows the regulation of access down to particular attributes of business objects. In contrast, platform 3 regulates user rights at the level of business objects. Even though this simplified approach reduces complexity, it was overall perceived negatively by developers since the restrictions overweighed the positives. In particular, platform 1 exemplified that a well thought-through concept for access and security controls can limit complexity without reducing functionality and achieves a high appreciation among developers.



Category	Characteristic	Grd.	Platform 1	Platform 2	Platform 3	Avg.
Shared Components	Access and Security Controls	15	++	+	-	+
	Data Management Capabilities	17	-	-	--	-
	Platform Connectivity	13	+	o	-	o
	Templates and Building Blocks	17	++	+	-	+
Extensibility	Configuration Capabilities	12	++	-	-	o
	Programming Capabilities	16	+	+	o	+
Development Tools	Web-Development Environment	16	+	--	-	-
	Local Tools	13	+	-	-	o
Learnability	Required Knowledge	11	-	o	-	-
	Provided Knowledge	16	o	--	--	-

Table 2: Score of assessed platforms

PaaS technology replaces the complex of problem how data is modeled and how it is accessed with a standardized approach. Therefore, capabilities of the platform helping to reduce the complexity of managing data are of great importance for the evaluation of PaaS. Our study reveals that the current capabilities of platforms to manage data are widely perceived negatively. It seems that the greatest strength of PaaS is also its greatest weakness: the reduced complexity which is the result of standardized modeling and accessing of data also comes with many constraints. All three evaluated platforms restricted the modeling of data. N:M relations, for example, were only possible by using workarounds. These workarounds prevented the use of preexisting CRUD methods of business objects and required the implementation of custom queries. Custom queries were often limited to a particular amount of data sets or restricted in run-time. To make things worse, all three platforms did not support a JOIN operator in custom queries, which again forced developers to use workarounds. The described restrictions were systematic for all three platforms. While some of them may not be important for simpler business cases, a great degree of freedom and flexibility when modeling or accessing data is of high relevance for more complex scenarios, just as the observed alumni network case. Consequently, the evaluation of the DBMS characteristics turned out negative for all three platforms.

In the context of an ever-growing heterogeneity of infrastructures, the question also arises, whether or not PaaS systems provide sufficient *connectivity* capabilities to integrate with the existing non-cloud IS landscapes. The provisioning of mature connectivity features can be a distinctive success factor for a cloud-based platform especially in the context of applications working on large data volumes (e.g., analytical solutions). Growing IT landscapes require API access to integrate processes and data. In particular, less powerful solutions are also very dependent on the possibility to complement existing features with external services since they do not provide sufficient functionality on their own. However, more comprehensive products may also benefit from connectivity since they may also lack the functionality for a particular use-case. Our results indicate that not every vendor is aware of the importance of connectivity. Less powerful platforms (e.g., platform 2 and 3) in particular lack connectivity, even though they would benefit the most from it. Platform 1, superior in functionality, also provides superior connectivity. An obvious explanation for this could be that connectivity features are equally prioritized by vendors as other PaaS features, resulting in extensive connectivity features within powerful platforms and vice-versa. Alternatively, limited connectivity can also be a strategic choice forcing developers to implement as much as possible (from scratch) on the same platform. However, for developers mindfully considering different PaaS alternatives, this potential lock-in may be an additional reason to decide on a larger and more powerful platform in the first place. Hence, the strategic decision made to chain developers to a particular product can easily backfire and prevent that developers pick the product in the first place.

Templates and Reusable building blocks of the tested platforms include entire applications (e.g., through a market place, application templates), reusable object types (e.g., for contact management), user-interface elements (e.g., ready-made calendar widget) and business logic (e.g., approval workflows). Similarly to the usage of patterns or templates in other developments contexts, reusable elements can improve developer productivity, when new applications can leverage the work done in previous developments. This capability seems to have reached a considerable maturity in the investigated platforms, showing a positive overall rating by the developers. Remainders of the former SaaS application manifest mostly in the reusable objects (e.g., data model of core business objects). In summary, reusable building blocks are one of the distinctive characteristics of PaaS since they are responsible for a major part of the benefits of PaaS technology. Therefore developers on the one side should thoroughly assess the extent and quality of reusable building blocks of a platform while, on the other side, vendors should lay extra focus on these characteristics of their product.

At least as important as the available buildings blocks, are the capabilities of a platform to adjust those building blocks to one’s own requirements. In general, developers positively assessed the *configuration capabilities* and the underlying paradigm promoting configuration before programming. However, the diverting results achieved by the three platforms can be traced back to the different configuration approaches followed. Platform 1, on the one hand, promotes a bottom-up configuration approach starting with the data model of an object. Platform 3, on the other hand, centers all configurations around the user-interface of an object from which a data model is derived. According to the qualitative data, this makes the configuration process more complicated than necessary. With regard to their flexibility, the data indicates that all three platforms have not found a satisfying solution for the question to which degree a developer should be able to adjust reusable objects. Even though all three platforms allow the configuration of particular attributes of reusable objects, some attributes are not changeable due to non-obvious reasons. Even more frustrating is the lack of consistency in this context, manifesting, for example, in the fact that a similar attribute can be easily changed in one reusable object but not in another one.

In addition to pure configuration, most platforms also provide classical *programming capabilities* to define the user-interface or implement scripts on the client or server side. Here, the qualitative data reveals two important points: First, developers appreciate the functionality to implement custom code since it strongly increases flexibility and is also necessary for most use-cases. From our sample, only platform 1 followed a holistic approach to embed custom user-interfaces, client-side code and server-side scripts. Second, developers seem to appreciate if a platform supports well- known programming / mark-up languages such as HTML, XML, CSS, JavaScript or JAVA. Platform 1, which promotes a proprietary language to implement server side scripts, received some critical comments on this point.

Usually, the inherent tasks and characteristics of a development process on PaaS remain similar. Consequently, developers in our study seemed to transfer their experiences and expectations from traditional integrated development environments to the platforms in use. This resulted in rather negative evaluations of the *web development environment*, e.g., developers which were used to automatic source code highlighting missed equivalent functionality in the platforms. This and other deficits lead to an overall negative evaluation of these capabilities. In addition to missing development functionality, the results of the study suggest that there are still usability constraints of web-based development environments in comparison to local IDEs. Although web-based applications often have difficulties to cope with the usability of local IDEs, one of the platforms (platform 1) surprisingly received an overall positive assessment. Its web development environment matched the functionality and usability of local solutions. The source code editor supported code highlighting and auto-completion. Yet, also for platform 1, developers missed the flexibility to arrange windows and criticized the higher lag when clicking at buttons or jumping back and forth between two screens.

In general, developers recognize any additional *local development tools* available, e.g., for batch uploads of data or to implement in a local eclipse environment. Qualitative data indicates a strong negative correlation between the quality of the web development environment and the concept of additional tools: when a platform received negative marks for its web development environment, developers criticized the absence of additional local development tools which would fix the shortcomings of the web-based environment.

For developers who are about to move from a (traditional) local development environment to PaaS, the *learnability of the platform* is an important factor. Our study indicates that developers acknowledge the steep learning curve when it comes to the implementation of simple functionalities using customization, e.g., creating a business object and its attributes. However, regarding more complex functionalities, such as implementing business logic, users criticize that the required knowledge is disproportionately higher since it requires a holistic understanding of the platform and its peculiarity. Furthermore, the tested PaaS are built on proprietary architectures which do not follow accepted standards and often fail to maintain consistency. Platform 1, for example, distinguishes between standard objects, going back to the original SaaS solution, and custom objects. Different rules and regulations regarding configuration apply, making it harder for developers to acquire the necessary knowledge. In order to be able to determine the best solution for a development task, it is necessary to have all the knowledge about the platform available.

One important aspect of acquiring this knowledge is through the provided *documentation* material. The results of our study show that platform vendors address this issue only partially. The vendor of platform 1 offers comprehensive training material. All major features are covered by dedicated manuals. In addition, a huge developer community exists which can give very specific help. Developers however, complained about the lacking ease of access to required information. Platform 2 and 3 lack both sufficient documentation material as well as big a developer community. Therefore, competent help is very hard to get. The main reason for the bad scores of platform 2 and 3 was however that some central features of the platform remain completely unaddressed by the provided manuals.

Key Learnings

Returning to the initial problem, our article results in two major contributions: first, we have used a qualitative case-study to derive a taxonomy of ten functional and nonfunctional characteristics which can be helpful to come to a well-informed decision. Second, we applied this taxonomy to assess three current PaaS solutions and derive general statements about the current stage of development of this new kind of technology. Although the case study was conducted by students, the qualitative data indicates a deep and thorough analysis of the assigned platform. We therefore think that the findings also apply in a professional setting and constitute valuable help for developers.

With regards to the current situation on the market for PaaS technology, we come to the following conclusion: Current offers diverge significantly, both in quantity and quality of the implemented features. Developers should thoroughly consider those differences to make a well-informed decision. The taxonomy poses a sharp contrast to the marketing brochures of platform vendors. It allows developers to focus on the essential characteristics when assessing PaaS technology. In particular, they need to find answers to the four important questions:

1. What shared components are offered by the platform?
2. How extensible are these functionalities?
3. Which development tools are provided?
4. What knowledge is necessary to develop on the platform and how can this knowledge be acquired?

Summarizing our analysis, we are confident to make the following statements which capture some key characteristics of PaaS technology at its current stage of development:

- Shared components offered by platforms are probably the key benefit of PaaS. They can lead to significant increases in developer productivity since they remove the need to reinvent the wheel over and over again. Furthermore, the provided components provide structure along which developers can create their own applications reducing complexity and cognitive load in the development process.
- Developers cannot expect the same degree of flexibility as regular programming frameworks provide. Whether or not PaaS applies to this is a question of the complexity of the use case, but also if the platform follows a configuration-centric

instead of a programming-centric approach. Not surprisingly, the platforms that turned out to be more flexible are those which follow a holistic coding-centric approach.

- The ideal of rapid application development partly applies with regards to minor adjustments or extensions. Configuration-centric platforms offer a great selection of reusable components while still maintaining some extensibility. Yet, with regards to major changes, application development is not much more efficient than local development. On the contrary, the limited functionality and usability of the web-based IDEs can turn developing into a painful, slow and frustrating activity. Developers are therefore advised to look for a PaaS that also provides local tools.
- Current PaaS solutions are extremely diverse. Every vendor follows a proprietary approach. Proprietary approaches are contradictory to the promise of rapid application development since they prevent the reuse of existing knowledge and experience. Developers must expect a significant period of adjustment when switching to a PaaS solution. Vendors are well advised to respect industry standards and practice for their products.
- Smaller vendors seem to lack the capabilities to provide sufficient training material. This contradicts their efforts to generalize their SaaS to a platform and allows other developers to use shared functionality. After all, the best available functionality is useless if developers have no way to learn how to use it.

In conclusion, from the point of view of developers, PaaS systems can be seen as a first step to provide a development environment as a utility as envisioned by McCarthy. Our study shows that use cases of easy-to-moderate complexity can be successfully implemented based on PaaS technology. However, to be able to use utility development environments to develop more than utility applications, existing PaaS sytems still need to advance significantly.

References

[1] M. Armbrust et al., “A view of cloud computing,” Communications of the ACM, vol. 53, no. 4, pp. 50-58, 2010.

[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” Future Generation Computer Systems, vol. 25, no. 6, pp. 599-616, Jun. 2009.

[3] I.Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared,” in 2008 Grid Computing Environments Workshop, 2008, pp. 1-10.

[4] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, “What’s inside the Cloud? An architectural map of the Cloud landscape,” in 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009, pp. 23-31.

[5] W. Sun, X. Zhang, C. J. Guo, P. Sun, and H. Su, “Software as a Service: Configuration and Customization Perspectives,” in 2008 IEEE Congress on Services Part II (services-2 2008), 2008, pp. 18-25.

[6] C. Weinhardt et al., “Cloud Computing – A Classification, Business Models, and Research Directions,” Business & Information Systems Engineering, vol. 1, no. 5, pp. 391-399, Sep. 2009.

[7] G. Lawton, “Developing Software Online With Platform-as-a-Service Technology,” Computer, vol. 41, no. 6, pp. 13-15, Jun. 2008.

[8] M. Sakal, “GUI vs. WUI Through the Prism of Characteristics and Postures,” Management, 2010.