How to Support Customisation on SaaS: A Grounded Theory from Customisation Consultants

Hui Song, Franck Chauvel, Arnor Solberg
SINTEF
Email: first.last@sintef.no

Bent Foyn
VISMA Software International AS
Email: bent.foyn@visma.com

Tony Yates
SuperOffice AS
Email: tony@superoffice.com

***Abstract*—This paper reports the initial result of a qualitative research on how to support customization of SaaS (Software as a Service). The research follows the grounded theory method, and investigates the expectation of consultants who are specialized in customising enterprise software systems. The resulted theory contributes to the understanding of how customisation on SaaS differs from the traditional one, and provides a high-level guidance for SaaS vendors to prepare effective support for customisation.**

***Keywords*-Software Customisation, Grounded Theory, Software as a Service**

I. Introduction

Vital enterprise activities such as accounting, sales, human resource management or customer relationship management demand dedicated *enterprise software*. As every company is unique, such software typically can not be applied as is. In fact, more than 80% of companies are reported having moderate to extensive customisations [1]. Such *customisations* go beyond *configurations* in the way that they require *development work* to implement *ad hoc* extensions. Customisation is often implemented by a third-party consultant company, also known as the *partner* of the independent software vendors (ISV).

Customisation is traditionally based on the assumption that the software is deployed on the customer's own premise, so that the customer has full control of the system and is free to customise it. The assumption no longer holds when enterprise software moves to multi-tenant Software as a Service (SaaS), because software is running *in the cloud* and controlled by the SaaS vendor. As a result, some considers that SaaS customisation is too complicated and should be abandoned. The state of practice in SaaS advocates configuration over customisation [2]. The two software vendors that commissioned this research also started from configuration support on their SaaS solutions. However, it appears that many of their partners have a different view, thus, these vendors continuously receive requests to support customisation on their SaaS products.

In general it is crucial for SaaS vendors to understand *the partner's opinions* on customisation, *i.e.*, why customisation is necessary for SaaS, how it differs from the on-premises time and how the difference would reshape the way vendors support customisation. However, it is a challenging task, because these opinions are subjective expectations about an emerging object, and we need to summarize them into a usable theory to guide the vendor's future decisions.

This paper reports the initial result of a qualitative research towards a *grounded theory* [3] that formalises how partners see customisation on SaaS. The theory is elicited from interviews of partners responsible for doing customisation. In short, we found out that customisation is still important in a SaaS context due to the gap between customers and vendors, but it differs from traditional on-premises customisation, because multi-tenancy alters the responsibilities between vendors and their partners. As a result, SaaS vendors have to be more actively involved in the new customisation activities, providing effective tool support for customisation. The theory guides vendors in designing and evaluating the possible tool support from a high abstraction level. To illustrate how to use our theory, we applied it to scrutinise the customisation solutions provided by Salesforce and SAP.

The contribution of this paper is twofold. 1) We investigate software customisation as a neglected software engineering activity, and vision how it evolves and should be supported in SaaS. 2) We apply qualitative research to software engineering in a novel way, *i.e.*, to study the subjective expectations of human stakeholders in software development activities.

The remainder of the paper is structured as follows. Section II summarises our approach to qualitative research and Section III then presents our theory. Section IV applies it to inspect how Salesforce and SAP support customisation. Finally, Section V discusses selected related work before Section VI explains the limitations and the future plans.

II. The Research Approach

We undertook a qualitative research following Corbin's grounded theory method [4]. It is widely used in the social sciences to systematically analyse qualitative data, through the steps of *open coding* to extract concepts by reviewing the data, and the *theoretical sampling* to define the relationships and categories on the concepts. The steps are performed iteratively as new data is appearing, until a *theoretical integration* is reached, where no new concepts emerges.

We collected the data mainly through interviews. So far, we have interviewed in total 10 first line consultants from 5 partner companies. The partners have been working for many years (between 8 and 20) on customising the on-premises products from the two vendors, but have no experience on

The final publication is available at IEEEXplore at http://dx.doi.org/10.1109/ICSE-C.2017.136

customising SaaS. This avoids the interviewees from confining their opinions to specific customisations mechanisms. A team with three co-authors of this paper, from a research institute and the two vendors, attended all the interviews. We also used documents from the partners' websites, leaflets and advertisements, as supplemented data.

We did open coding after each interview by reviewing the meeting minutes and the voice records. After that, we compared the new concepts with the existing ones to identify equivalent concepts and possible categories. We finally identified four categories, and selected one central category to explain the main difference of SaaS customisation. The research team reported the progress and the periodic results twice to an extended group of 15 people from the institute and vendors, and integrated the feedbacks into subsequent analysis. We have not yet reached a *theoretical integration*, as new concepts still emerged in the last interview. The work is still on-going, and the theory we report here is initial result.

III. The Result

The Theory. Customisation remains important for SaaS, but multi-tenancy changes stakeholders' responsibilities. Partners are no longer only responsible to customers, but have to rely on the vendors to develop and deploy customisations. As a result, partners have to drop the old way of developing customisations freely, with *ad hoc* modifications scattered in the product. Instead, they should adopt development with more constraints and involvement from vendors, and focus more on business. This transformation conflicts with the partner's traditional interests, *i.e.*, to minimize the cost for customers, to integrate the custom code seamlessly with the main product, and to keep themselves flexible. Therefore, the critical success factor for a customisation ecosystem is that the vendors should offer not only highly customisable SaaS, but also effective supporting tools that fit the new customisation activities and in the same time compensate on these basic interests.

The theory is a summary of the *concepts* we elicited from the interviews, which are classified into four *categories*. Hereafter, we highlight the name of each concept or category in the sans-serif font. Figure 1 illustrates these four categories and their relationship. The driving forces behind customisation shape the traditional responsibilities between stakeholders, but multi-tenancy changes them. The typical customisation activities are directly determined by the responsibilities, and the new ones may conflict with the fundamental driving forces. The effective customisation capabilities and supports from the vendors should fit the new activities and in the same time compensate them on the driving forces. The categories and their relations explain the theory at a high level, and we further elaborate the theory by explaining the main concepts.

Driving forces. Customisation arises from the gap between customers and vendors: Vendors focus on generic enterprise software but lack the knowledge and resource to understand every customer's business. The loose relationship and the geographic distances between the vendors and customers

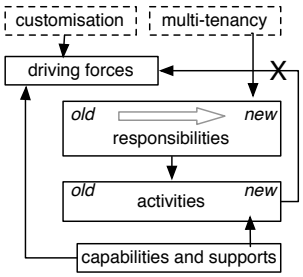

Fig. 1. Categories and their relationships

widen this gap. Partners bridge this gap: they have a deep business background, a long-term relationship with customers, and usually have consultants working locally with customers. Therefore, partners can investigate *together* with the customers on their specific needs. As long as this gap stands—SaaS itself does not narrow it—it is necessary to have third-party partners customise enterprise software. *Configuration cannot replace customisation* because it disregards this gap by implying that vendors can foresee and implement all possible features.

When doing customisation, partners follow three principles: low cost, one product and flexibility. Low cost is the customer's main interest, and dominates the way partners work on customisation. Second, customers should feel as if they were using one product, instead of a loose composition of functions or services. Last, partners want to be flexible and agile, so that they can keep adapting themselves to new customers with very different businesses and backgrounds.

Responsibilities. The three major stakeholders have different responsibilities, as illustrated in Figure 2.

For on-premises systems, partners act as a complete delegation of the vendors: They retail the product to customers, adapt or complement its functions. They then help customers to use it and fix any subsequent problems. The partner and their customer have a shared quality responsibility on the customisation, which minimises the cost during customisation. Upgrades are also a shared responsibility: Partners may do some debugging and fixing work on their customisations when upgrading the main product, but many customers choose to keep the old versions for a long time. There is no direct responsibility from vendors to customers.

For SaaS however, the vendors host the product for all customers together with their customisations. This forces the vendors to directly face the customers, and to take direct responsibility for the function and quality of both their main product and any customisations. When customers meet a problem, they would first contact the vendor instead of the partner, because the partner may even not have access to their running product instance. Under this setting, the partners have to take the new responsibility for the quality of their customisation to the vendors. A customisation with bad quality not only harms the vendor's reputation, but also disturbs other customers who are hosted by the same cloud premises.

The change of responsibility is the core category [4]. It explains how SaaS customisation is different from the traditional one, and is the root rationale behind our theory

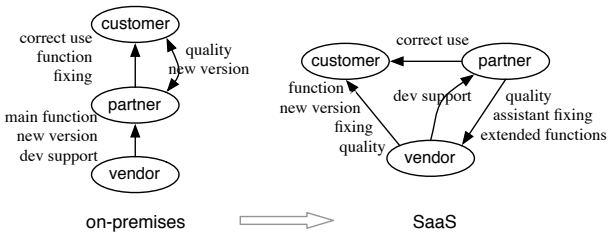

Fig. 2. Responsibilities between stakeholders in customisation

about how vendors should react to the change, which we will explain by the following two categories.

Activities. Customisation comprises typical software development activities. The change of responsibilities will dismiss some customisation activities and encourage new ones.

For on-premises systems, customisation is an *ad hoc* development. A customisation project starts from bidirectional training, where the customer learns how to use the vendor's product while the partner learns the customer's business. This training exposes customisation needs that the partner will then implement. When possible, the partner directly modifies the product code or works on the database. Before delivering the customised product, partners and customers would do a joint-testing together, in order to share the quality responsibility. The result, so called "custom code", includes numerous pieces scattered over the code base of the vendor's product. Partners prefer to keep minimal legacy between different customisation projects, without caring about reusable code, documents, license, *etc.*

For SaaS however, customisation is more like a supplement to the main product, because it is the vendor who now directly faces the customers. Partners have to work under the constraints set up by the vendor, and deliver custom code as wrapped components that the vendor can easily integrate into the running instance. The customisation needs to go through a vendor-involved testing process to enforce the partner's responsibility for quality. SaaS also provokes new activities, *e.g.*, custom code reuse and non-programmer customisation: Since the custom code of different customers are running on the same product instance, in the same cloud premises, it is much easier to reuse the customisation between different customers or even different partners. Besides, as Cloud hides infrastructure details, partners can focus on the business aspects and, eventually, non-programmers such as sale or training staff may build customisations.

The change on activities is conflicting with the fundamental driving forces. The drop of *ad hoc* development would increase the cost. The vendor's involvement, either with their constraints or in vendor-involved testing, would disregard the gap, and harm the parnter's flexibility. The reuse of custom code will cause more legacy to partners which is bad for flexibility. Only customisation by non-programmers may help considering both the gap and the low cost.

Capability and support. Customisation capabilities define how vendors allow partners to customise their product. According to the level of intrusiveness, typical capabilities include code modification directly on the product, managed components running on an engine within the product, and external add-ins running on their own infrastructures.

All the three capabilities are possible for SaaS, but our theory indicates that additional support is required to fit the new activities and to avoid violating the fundamental driving forces. We briefly discuss some potential supports that emerged from the interviews. If a vendor want to allow code modification, then code analysis, rewriting and generation tools are needed to enforce the constraints, simplify the reuse and release the burden of vendor-involved testing. This in turn rescues partners from the pitfalls on cost and flexibility. Non-programmer customisation will be very difficult to achieve in this direction, but a set of powerful templates could be helpful. Following the direction of managed components, the tools on code analysis and generation are still important but can be lighter, because some constraints and reuse support can be built-in to the engine and the language. A well-designed Domain-Specific Language (DSL) with proper business abstraction is promising for non-programmer customisation. However, vendors should avoid embedding too much into the engines or the languages to disregard the gap. Following the direction of external add-ins, partners can have the flexibility to choose where the services are hosted and how to implement them. There is much less constraints to consider, and also less to be tested by the vendors. However, a pitfall could be that partners end up with paying too much cost on hosting and maintaining the small services, and violating the one product policy. A compromising option would be to host the "external" services on the vendor's cloud. There are also common tool supports that are important across capabilities. A powerful IDE with automatic building tools will lower the cost on wrap-up, reuse and constraints. An online development tool integrated with the main product will speed up development of minor customisation or experiment of writing small pieces of custom code. An on-demand testing environment will help the partners accept vendor-involved testing.

IV. Application of the Theory

We applied the theory to inspect the customisation support of two commercial SaaS solutions from Salesforce[1] and SAP[2].

Salesforce allows both "customisation within the cloud" in their Apex language, and "customisation outside the cloud" through their REST API, corresponding to managed components and external add-ins, respectively. The vendor recommends the first way, unless an external legacy system need to be integrated. Some advanced support tools are provided along with this capability. The Apex language and the limited libraries embed the constraints. In the same time, Salesforce provides strong compilers and warning systems to reduce the cost for developers to obey these constraints. The migration tool automatically wraps the scattered custom code into a manageable package, which reduces the cost for component wrapping and reuse. The disadvantage is that

[1] developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/
[2] https://discuss.asug.com/docs/DOC-42313/version/1

even a minor customisation will have to be implemented as an Apex class in a standard package, which harms the partner's flexibility. Salesforce has an online workbench with intuitive wizards for developers to compose and experiment with Apex. It is a promising step towards non-programmer customisation, but so far the composed pieces are not yet stand-alone customisation. Salesforce enforces developers to write test cases for their custom code, which are executed automatically in cloud to eliminate the manual testing by vendors. This increases the partner's flexibility, even though writing test cases has some cost. Salesforce lacks the support for the one-product principle. There is a clear separation between the main product and the custom code: They are not written in the same language nor executed by the same engine. A possible improvement would be to rewrite part of the standard product in Apex, and open them to the partners for direct modification.

SAP supports customisation of their products on the SAP HANA platform, with the "in-app extensibility" (managed components) and the "side-by-side extensibility" (external add-ins). The in-app extensibility on SaaS is limited to basic functions such as adding custom fields or simple business rules, whereas more complex customisation is still only available on the on-premises version. Therefore, SAP regards side-by-side extensibility as their by-default customisation capability, which is the opposite of Salesforce. By running the customisation in the customer's own premises or third-party cloud, SAP does not need to set constraints on resource consumption, which benefits the partner's flexibility. But such solution innately violates the one-product policy, and increases the cost for customers. As a compensation, SAP provides a UI framework (SAP fiori) to standardise the user experience between the main product and the custom extensions, as well as the HANA cloud connector to simplify and unify the connection between external clouds and SAP's own one. To the best of our knowledge, SAP does not provide dedicated tools for the wrapping and reuse of custom code, and the vendor-involved testing. Moreover, since the partners have to care about many technical issues, such as the API and the hosting environment, it is hard to foresee a way to support non-programmer customisation. As we have discussed before, a promising solution could start from hosting custom code directly on their own cloud.

Comparing the two solutions, Salesforce appears to better satisfy the partner's expectation on SaaS customisation. This conclusion is consistent with the market positions of the two solutions: Salesforce is a native cloud-based CRM system and develops the customisation support with cloud thinking from the beginning, whereas the SAP platform is still under the transformation from on-premises to cloud.

V. Related Work

This work relates to the qualitative research approaches in empirical software engineering. A close work is Rothenberger and Srite's investigation, combining grounded theory and case studies, on the factors that influence the degree of customisation on ERP systems [5]. More generally, researchers used the grounded theory method to investigate how an agile development team control the quality [6], how much architecting is proper for different agile teams [7], *etc.* More approaches can be found in a survey by Stol *et al.* [3]. These approaches investigates how developers work currently, in order to find best practices or to improve the current processes. This paper presents a novel attempt, *i.e.*, to understand and theorize developers' expectation of working *in new contexts*. The attempt is valuable considering the fast-changing technical trends.

There are research approaches to enabling customisation of multi-tenant SaaS. Walraven *et al.* [8] added customisation support to multi-tenant middleware based on dynamic dependency injection. Mietzner *et al.* [9] enables customisation by managing variability at runtime based on software product line techniques. Our work, at this stage, does not focus on any concrete technical solutions for the SaaS customisation, but rather on understanding the users expectations to guide the preparation of such solutions.

VI. Conclusion and Discussion

This paper reports the first step of an qualitative research to theorise how customisation, as a software development activity, is evolving when enterprise software is moving from on-premises deployment to multi-tenant SaaS. The initial result is a theory that provides a high-level guidance for SaaS vendors to decide how they should participate and support their partners in doing customisation.

The initial result has limitations due to the small data scale. The theory is based on only 10 interviewees, who are from the same country and work with only two vendors. The result is not complete, and can be specific to the products or the market. However, the work is just started, and we will collect and analysis more qualitative data from a wider scope in the next step.

The result is not well evaluated. A long term evaluation plan is to apply the theory to guide the two vendors in designing their customisation environment. Finally, the acceptance and feedback from their partners on this environment will evaluate the correctness and usefulness of the result.

References

[1] "Maintaining ERP Systems: The Cost of Change, White Paper by IDC: International Data Corporation," Tech. Rep., may 2013.

[2] W. Sun, X. Zhang, C. J. Guo, P. Sun, and H. Su, "Software as a service: Configuration and customization perspectives," in *Congress on Services Part II, 2008. SERVICES-2. IEEE.* IEEE, 2008, pp. 18–25.

[3] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: a critical review and guidelines," in *International Conference on Software Engineering*, 2016, pp. 120–131.

[4] J. Corbin and A. Strauss, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, 2014.

[5] M. A. Rothenberger and M. Srite, "An investigation of customization in ERP system implementations," *IEEE Transactions on Engineering Management*, vol. 56, no. 4, pp. 663–676, 2009.

[6] Y. Lindsjørn, D. I. Sjøberg, T. Dingsøyr, G. R. Bergersen, and T. Dybå, "Teamwork quality and project success in software development: A survey of agile development teams," *Journal of Systems and Software*, vol. 122, pp. 274–286, 2016.

[7] M. Waterman, J. Noble, and G. Allan, "How much up-front? A grounded theory of agile architecture," in *International Conference on Software Engineering*, vol. 1, 2015, pp. 347–357.

[8] S. Walraven, E. Truyen, and W. Joosen, "A middleware layer for flexible and cost-efficient multi-tenant applications," in *International Conference on Distributed Systems Platforms and Open Distributed Processing*, 2011, pp. 370–389.

[9] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl, "Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications," in *ICSE Workshop on Principles of Engineering Service Oriented Systems*. IEEE Computer Society, 2009, pp. 18–25.