Differentiating Feature Realization in Software Product Development

Aleksander Fabijan[1], Helena Holmström Olsson[1], Jan Bosch[2]

[1] Malmö University, Faculty of Technology and Society, Nordenskiöldsgatan 1, 211 19 Malmö, Sweden
{Aleksander.Fabijan, Helena.Holmstrom.Olsson}@mah.se
[2] Chalmers University of Technology, Department of Computer Science & Engineering, Hörselgången 11, 412 96 Göteborg, Sweden
Jan.Bosch@chalmers.se

Abstract. The software is no longer only supporting the mechanical and electrical products. Today, it is becoming the main competitive advantage and the enabler of innovation. However, in the companies that we study, the way in which software features are developed still reflects the traditional 'requirements over the wall' approach. Consequently, individual departments prioritize what they believe is the most important and are unable to identify which features are regularly used – 'flow', there to be for sale – 'wow', differentiating and that add value to customers, or which are regarded commodity. In this paper, and based on case study research in five large software-intensive companies, we develop a model in which we depict the activities for working with different types of features and stakeholders. The model (1) helps companies in differentiating between the feature types, and (2) selecting a methodology for their development (e.g. 'traditional' vs. 'Outcome-Driven development').

Keywords: data, feedback, outcome engineering, data-driven development, goal-oriented development

1 Introduction

Rapid delivery of value to customers is one of the core priorities of software companies [1–3]. Consequently, the amount of software added to products with attempts to deliver the value is rapidly increasing. At first, software functionality was predominately required in products to support tangible electrical, hardware and mechanical solutions without delivering any other perceptible value for the customers. Organizations developed the software as a necessary cost, leaving the prioritization part to the responsible engineering, hardware, and mechanical departments, without exploring the value of software features as such. Today, products that were once built purely from hardware components such as e.g. cars and household appliances, contain functionality that allows them to connect to the Internet, exchange information and self-improve over time. Software functionality is rapidly becoming the main competitive advantage of the product, and what delivers value to the customers [4].

However, the way in which software features are being developed, and how they are prioritized is still a challenge for many organizations. Often, and due to immaturity and lack of experience in software development, companies that transitioned in this way (e.g.

from electrical to software companies) treat software features similarly to electronics or mechanics components. They risk being unable to identify what features are differentiating and that add value to customers, and what features are regarded commodity by customers. As a consequence of this, individual departments continue to prioritize what they find the most important and miss the opportunities to minimize and share the investments into commodity features [5], [6]. Companies that are becoming data-driven, are exploring this problem and trying to learn from the feedback that they collect to optimize the prioritization decisions. The focus of software companies and product team that recognized the benefits of being data-driven is to develop the product that delivers the most value for the customer [1–3]. For every feature being developed, the goal is to create clarity in what to develop (Value Identification), develop it to a correct extent (Value Realization), and accurately measure the progress (Value Validation).

In our work, we perform an exploratory case study in which we were interested in (1) identifying how companies differentiate between different types of features and (2) how they prioritize the development activities with respect to the type of the feature that they are developing. In our work, we identify that the lack of distinguishing between different types of features is the primary reason for inefficient resource allocation that, in the end, make innovation initiatives suffer. In our previous work [7], we confirmed that differentiating between the different levels of functionality is indeed a challenge that software companies face today. In this paper, we further detail the feature differentiation model and evaluate it in two additional case companies.

The contribution of this paper is twofold. First, we provide detailed guidelines on how to distinguish between different types of features that are being developed and we provide empirical evidence on the challenges and implications involved in this. Second, we detail a conceptual model to guide practitioners in prioritizing the development activities for each of the feature types identified in our differentiation model. We identify in our work that for the successful development of innovative 'Wow' and 'Flow' features, a different methodology is required. We label this methodology 'Outcome-Driven development' and demonstrate how it differs from 'Traditional development' approach. With our contribution, companies can develop only the amount of feature that is required for commoditized functionality and, on the other hand, frees the resources to maximize their investments in innovative features that will deliver the most value.

2 Background

Software companies strive to become more effective in delivering value to their customers. Typically, they inherit the Agile principles on an individual development team level [8] and expand these practices across the product and other parts of the organization [9]. Next, they focus on various lean concepts such as eliminating waste [10, 11], removing constraints in the development pipeline [12] and advancing towards continuous integration and continuous deployment of software functionality [13]. Continuous deployment is characterized by a bidirectional channel that enables companies not only to deliver new updates to their customers in order to rapidly prototype with them [14], but also to collect feedback on how these products are used. By evaluating ideas with customers, companies learn about their preferences. In this process, the actual product

instrumentation data (for example, features used in a session) has the potential to identify improvements and make the prioritization process of valuable features more accurate [15]. In fact, a wide range of different techniques is used to collect feedback, spanning from qualitative techniques capturing customer experiences and behaviors [16], [17], [18], to quantitative techniques capturing product performance and operation, [18]. However, this development of getting rapid feedback in real time is only possible today when software products are increasingly becoming connected to the internet. And as such data focuses on what customers do rather than what they say, it complements other types of feedback data [19] and improves the understanding of the value that the product provides [18], [20]. For example, knowing which features are used in a certain context helps companies in identifying customer preferences and possible bottlenecks. The intuition of software development companies on customer preferences, can be as much as 90% of the time inaccurate [21–23]. In most companies, customer feedback is collected on a frequent basis in order to learn about how customers use products, what features they appreciate and what functionality they would like to see in new products [16]. This enables the companies to very quickly test new variants with the customers and explore how they perform with the respect to the metrics that are important. This concept is known as Continuous Experimentation and is even though continuous integration, deployment, and data collection are well developed across different organization types, the way in which companies work to identify what to develop, to what extent, and how to measure the progress differs substantially [5, 18].

2.1 Business and Design Experimentation

Davenport [24] suggests the design of smart business experiments to emphasize the need to bridge business needs closer to software design and engineering. The importance of running experiments to learn more about customer behavior is one of the core principles of the Lean Startup methodology [14, 25]. This idea is emphasized also in the product management literature by Bosch [18] who propose the need for constant generation of new ideas that should be evaluated with customers. Fagerholm et al. [1] suggest the RIGHT model for continuous experimentation and support constant testing of ideas. In their view, this is essential to create the evaluation of product value as an integral part of the development process. The strength of experimentation is further accelerated with controlled experiments. In controlled experiments (also known as A/B test), users of a software product or a feature are randomly divided between the variants (e.g., the two different designs of a product interface) in a persistent manner (a user receives the same experience at multiple software uses). Users' interactions with the product are instrumented and key metrics are computed [26, 27]. Research contributions with practical guides on how to create controlled experiments in software product development have previously been published both by Microsoft [28, 29] and Google [30]. The Return on Investment (ROI) of controlled experiments has been demonstrated and discussed a number of times in the literature [27, 28].

2.2 Feature Differentiation

Already in 1984, Kano [31] has identified the need to differentiate between product attributes (e.g. attractiveness, satisfaction, must-be, indifferent and reverse attributes) based on their importance for the product and the stakeholders. One of the key characteristics of the Kano model itself is the notion of attribute drift (for example, an attribute such as camera resolution is expected to improve over time). Attribute drift is driven by customers' expectations and changes over time. In this context, Bosch [32] developed the 'Three Layer Product Model' (3LPM) on the idea that a return on innovation requires differentiation [33]. The model provides a high-level understanding of the three different layers of features, i.e. commodity, differentiating and innovative. And although these models continue to be used in many industries today, they do not provide guidance on how to differentiate between the different types of features, neither how to prioritize software development activities based on the differentiation. At the same time, the pace at which we are becoming to be able to learn from the customers in the software industry is faster than anywhere else [11, 34]. The feedback that is available today (e.g. feature usage data while the product in the hands of the users) opens new possibilities for understanding and differentiating features in products. Using existing models (e.g. such as the Kano model or the 3LPM model) and accommodating new dimensions of learnings that we discussed above is challenging for software companies, and it blocks them from fully benefiting from the new possibilities [7, 35]. The implications of limited customer learning are substantive and can cause companies to inaccurately prioritize feature development by e.g. develop the software features to the wrong extend in either direction [11].

In our previous research [7], we developed a model in which four fundamentally different types of features are being developed. We name them "duty", "wow", "checkbox" and "flow" types of features. With "duty", we label the type of features that are needed in the products due to a policy or regulatory requirement. "Checkbox" features are the features that companies need to provide to be on par with the competition that provides similar functionality. With "wow", we label the differentiating features that are the deciding factor for buying a product. Finally, and with "flow", we label the features in the product that are regularly used.

In this paper, we address the new possibilities of learning from customers to (1) detail thea differentiation of features for software products, and (2) to suggest on how to prioritize the development activities for each of them. Our model (1) helps companies in differentiating between the feature types, and (2) selecting a methodology for their development (e.g. 'traditional' vs. 'Outcome-Driven development').

3 Research Method

In this section, we describe our research method. This case study [36] builds on an ongoing work with five case companies and it was conducted between January 2015 and June 2017. The goal of this study was to identify a state-of-the-art differentiation of software features. Companies A-C were participating for the full period of the ongoing research while companies D-E joined in this research in April 2016.

We present the case companies in Table 1 below.

A	**Company A** is a manufacturer and supplier of transport solutions construction technology and vehicles for commercial use. The systems that they develop require stable and fine-defined operation with no margin for misunderstanding. Companies' portfolio is extensive and includes both products for B2B as well as B2C types of customers.
B	**Company B** is a provider of telecommunication systems and equipment, communications networks and multimedia solutions for mobile and fixed network operators. Companies' portfolio is extensive and primarily consists of products developed for B2B customers.
C	**Company C** is a software company specializing in navigational information, operations management and optimization solutions. Their portfolio includes products and services for the B2B market.
D	**Company D** develops software systems for several different domains, including healthcare, energy and infrastructure operations. Their portfolio is extensive and includes products and services for both B2B as well as B2C types of customers.
E	**Company E** produces software and hardware systems for various surveillance purposes. Their portfolio includes products mostly for B2B types of customers and it operates in many market segments such as transport, infrastructure, retail, banking, education, government etc.

Table 1. Case company descriptions.

3.1 Data Collection

We collected our data using three primary types of data collection activities. First, we conducted **individual workshops** with all the companies involved in this research. Second, we conducted **a joint workshop** with participants from multiple companies participating and discussing the focus of our research. Finally, we conducted semi structured **interviews** following a guide with pre-defined open-ended questions with participants from each of the companies for in-depth analysis. The data collection activities were conducted in English and lasted in average three hours (individual workshops and joint workshops), and one hour (individual interviews). Each of the data-collection activities was conducted with practitioners in different roles (software engineers, project/product managers, sales specialists, and line/portfolio managers). In total, we performed 14 individual workshops, 7 joint workshops and 45 interviews.

3.2 Data Analysis

During analysis, the workshop notes, interview transcriptions and graphical illustrations were used when coding the data. The data collected were analyzed following the conventional qualitative content analysis approach [37]. We read raw data word by word to derive codes. In this process, we first highlighted individual phrases that captured our attention in the transcriptions. We used color coding to highlight different topics that emerged during the analysis process. In this process, we reflected on the highlighted text several times and took notes. As this process continued for a few iterations (e.g. by analyzing the first few workshop notes and interviews), codes emerged from the highlights. In the next step, we sorted these codes into categories (e.g. grouping them by color). This

type of design is appropriate when striving to describe a phenomenon where existing theory or research literature is limited. After we emerged with a few codes, we created definitions for those codes and continued to code the rest of the data using the definitions.

3.3 Threats to Validity

Internal Validity: As the participants were familiar with the research topic and expectations between the researchers and participants were well aligned, this can be a potential source of bias and thus a possible threat to internal validity. We mitigate this threat by providing unbiased overviews of the research area at the beginning of every data collection session, and avoiding suggestive interrogation during the workshops and leading questions during the interviews.

Construct validity. To improve the study's construct validity, we used multiple techniques of data collection (e.g. workshops, interviews) and multiple sources of data collection (product managers, software engineers, sales representatives, program managers, etc.). Meeting minutes from the workshops and interview transcriptions were independently assessed by three researchers to guarantee inter-rater reliability. Since this study builds on an ongoing research, the participants were familiar with the research topic and expectations between the researchers and participants were well aligned.

External validity. Considering external validity [38] from the theory constructed in this paper [39], our results cannot directly translate to other companies. However, we believe that the results can be generalized to other large-scale software development companies. Specifically, the results of this research are valid for companies that are transitioning from mechanical or electrical towards software companies.

4 Empirical Findings

In this section, we present our empirical findings. First, and to understand the current development practices in our case companies to identify the type and the extent of the feature that they are developing, we outline the current customer data collection practices in the case companies. We recognize that the practices to collect customer data do not differ with respect to the feature being developed, but rather depend on the perceived need of information ad-hoc. Next, we identify the challenges that are associated with distinguishing between the different feature types. Finally, we explore their implications.

4.1 Current State of Feature Differentiation

In the case companies, products and features that are being developed are handed over from one development stage to another, together with their requirements and priorities. Although our case companies acknowledge that they are aware of the need for differencing between the types of features, i.e. commodity, differentiating and innovative, the information that would help them achieve that does not circulate with the requirements. The differentiation strategy is unclear to the practitioners developing the features. And consequently, setting the development investment level for a feature does not vary with respect to the type of the feature, allowing the prioritization strategy to be in favor of

commodity features. We illustrate and describe the current state with description and quotes from our case companies below.

"Should we go into Maintenance budget? Or should it go to investment budget and we prioritize there?",
-Product Strategist from Company C.

"There is a lot of functionality that we probably would not need to focus on."
-Technology Specialist from Company C.

"Customer could be more involved in prioritization that we do in pre-development. Is this featuring more important that the other one?" –Product Owner from Company B.

4.2 Challenges with Feature Differentiation.

The current state advocates a situation where features are not differentiated in the strategy between being innovative, differentiating or commodity and, consequently, development activities do not differ between the features. Based on our interviews, we see that there are several challenges associated with this situation. Our interviewees report that the collection of feedback, which serves as the base for developing a feature is conducted ad-hoc and not following the type of the feature being developed. Second, the directives to differentiate the feature on a granular level are incomprehensible for our practitioners. Therefore, it is very challenging for the R&D staff in our case companies to be effective in understanding which of the features are differentiating and innovative in the market.

"We want to understand what the customer wants to have and truly, what do they need."
-Product Manager from company A.

"Functionality is software basically and the features are more subjective opinions and things that we can't really… it is hard to collect data." –Function owner from Company C.

"Those are the things that worry us the most. All of us, since it is so hard, you need to gamble a bit. If it turns out that you are wrong, then you are behind." –Product Manager from Company A.

4.3 Implications

Due to an unclear differentiating strategy, our case companies experience several implications during the development process of a feature. The companies consider the stakeholders that the features are being developed for as a uniform group and not differentiating the development extent of the features based on the type of the feature and stakeholder. This leads to a situation where resources are used in developing and optimizing the features that might never get used by a customer to a similar extent as the ones that are regularly utilized. Consequently, commodity suppresses resources that could be better used to develop features in the innovation layer.

"If you are sitting in a team and then you talk to maybe 2 or 3 customers, then you see that this is the most important thing we need to do." -Product Manager from company A.

"*We tend to focus on the wrong things. We need to look at the benefit for their customers*."

"*In our organization is difficult to sew everything together, to make it work. That requires funding that is almost nonexistent*." –Software engineer from Company A.

"We do also our own tests of competitors. We do feature roadmaps for [feature name] for example. And then we have a leading curve… We measure are we on track or not." *-Product Strategist from Company C.*

4.4 Summary of our empirical findings

Vague differentiating strategy results in challenges with understating the stakeholder and the purpose of the feature being developed, implicating uniform treatment of features. Invariable investment levels into development activities result in incomprehensible directives on what to develop, implicating arbitrary investments in development activities. Favoring commodity results in a challenge of identifying what is innovation, implicating the suppression of the actual innovation and projecting competitors current state as the norm. We summarize this in Table 1.

Current state	Challenges	Implications
Vague differentiating strategy	Understanding the stakeholder and purpose of the feature	Stakeholders treated uniformly, not reflecting their differing business value
Invariable investment levels	Incomprehensible high-level directives	Arbitrary investments in development activities
Feature prioritization processes is in favor of commodity	Commodity functionality is internally considered to be innovative	Commodity suppresses innovation Projecting competitors current state is the norm

Table 2. Mapping of the current state, challenges, and implications.

5 Differentiating Feature Realization

In this section, and as a response to the empirical data from our case companies, we present and detail our model for feature differentiation. The contribution of our model is threefold. First, it provides four different categories of features and their characteristics to give practitioners an ability to better differentiate between feature types. Second, and as a guide for practitioners after classifying a feature, we provide a summary of development activities that should be prioritized for every type of feature.

5.1 Feature Differentiation

In our model, four fundamentally different types of features are being developed: "duty", "wow", "checkbox" and "flow" types of features. The differentiation is based on the characteristics in Table 3.

Stake-holder	We recognize four types of fundamentally different stakeholders that are targeted with new feature development. Product users (D), the competitor developing or already selling a similar feature (C), the customer purchasing and asking for a feature (B), or a regulatory entity imposing it (A).
Feature Engagement.	This characteristic describes the level of engagement expected with the feature. Features are primarily used by the end-users (D) and occasionally by the customers directly (B). The engagement with the features is therefore expected for these two groups. Regulators and competitors, however, typically do not use the features directly. Instead, and what we see in our case companies is that verify the documents or tests demonstrating the existence or compliance of the feature. The expected exposure to the feature for regulators and competitors is therefore low.
Feedback	Feedback data that is collected about features under development is of various types. For example, the "Flow" features' (D) will be regularly used by users and should be equipped with automatic feedback collection mechanisms to retrieve customer data about feature usage. The "Checkbox" features'(C) source are the competitors that are being analyzed for a similar offering. In the case of "Wow" features, (B), the primary stakeholders are customers, whom companies study extensively through market analysis and available reports. Finally, and in the case of the regulation and standardization services (A), companies query regulation agencies for regulatory requirements that are necessary for the offering of the future.
Focus	"Flow" features (D) focus on the user and maximizing user value. Practitioners, in this case, develop and iterate features that are validated with the users. For "Wow" features (B), practitioners use their own technical ability to maximize the technical outcome of the feature. Here, for example, companies use their simulators to test the speed of the software or its stability. For "Checkbox" features (C), practitioners compare the feature under development towards the ones from the competitors. In this case, the objective of the development organization is to develop a feature that is matching or improving the competitors' feature. Companies continuously compare towards competition and intentionally slows down if they are performing better as expected. Similarly, in the case of "duty" features (A), it is the regulator's compliance that the companies are required to satisfy by developing the feature.
Sales Impact.	The greatest influence on driving sales have the features that focus on the customer - "Wow" features (B) and features that focus on the user - "Flow" features (D). Both types are differentiating or innovative. However, in B2B markets, the user of the product is different from the customer and hence "Flow" features are less of a deciding factor for potential customers. As an example, users communicate and show satisfaction with a product to their managers and departments that purchase the features and products. Their opinions have the possibility to indirectly influence the opinion of the customer. "Duty" and "Checkbox" features have very low or no impact at all for choosing a product over the one from a competitor.

Table 3. Feature Differentiation.

By going through the process of articulating different views on the characteristics below, product teams can make the most of their collective insight on classifying them into "duty", "wow", "checkbox" or "flow" type. What is important, however, is how they prioritize the development activities for each of the features types. We present this in the next section.

5.2 Activity Prioritization

In this section, we present the activities that should be prioritized for each of the four feature types that we identified in the previous section. We suggest how to set the extent of the feature that should be developed. Here, the extent of the feature can be either defined once (constant) or dynamically adjusted during development and operation (*floating* - alternates, *following*- follows the competitors *or open* - no limitation*)*. Second, the sources that contain the information required to set the development extent need to be defined. Next, we suggest the most important activities for feature realization. They are followed by the activities that do not deliver value and should be avoided. Finally, we suggest how to set the deployment frequency. We describe the approach in Table 4.

Duty features	The development extent for this type of features is constant and defined by the regulators. To identify it, practitioners can use research institutions, standardization industries and industry publishers as a source of feedback to get access to various standardization reports, vendor communications and obtain the regulatory requirements. For this type of features, identifying regulatory requirements and developing them until they satisfy the requirements are the two main activities. UX optimization, investments in marketing activities developing infrastructure and other similar activities for this type of features should be minimized. Deployment of this type of features is single.
Wow features	For this type of feature, development extent is dynamically set using the feedback from the market. Practitioners query social media, marketing agencies, customer reports, requests, and interviews to identify business cases and sentimental opinions about the product. The two most important activities of this type are the identification of technical selling points and selling characteristic of the product, and maximizing investments in the technical development of the feature and marketing it. The feature's quantifiable value should be maximized with periodically scheduled deployment increments.
Checkbox features	This type of features should be developed following a “sliding bar” extent that follows the competitor’s trend. Practitioners should guarantee to be on par with the indicative trend collected from industry test magazines and internal evaluation of customer products. It is essential to read articles in the media, consolidative question the customers, participate in customer events and perform trend analysis. Practitioners should perform competitor analysis to determine feature characteristics and develop the feature incrementally following the trend. Since features of this type will not be used extensively by the actual users, investments in improving user experience and interaction with the feature can be minimized. Deployment should be frequent and scheduled. Although products typically require this type of features to be even considered by the customers, they are not the decisive reason for customers to churn or select a product over a competitor.
Flow features.	With this type of features and the ambition to discover new and innovative concepts, practitioners should continuously deploy changes to their products, collect product feedback, analyze it, and perform A/B test to rank alternatives. The two most important activities for this type of feature are the defining of evaluation criteria and maximizing the ability to experiment with them. Interviewing the stakeholders or gathering qualitative information should be of secondary value and used for interpreting results. Also, and due to the continuous deployment, there is a high impact on the infrastructure.

Table 4. Activity prioritization.

6 Model Evaluation

We detailed our model and evaluated its feasibility with the five case companies in recent workshop and interview sessions. Our validation criteria were to identify whether companies now identify the feature types that they develop (based on our model), and whether it helps the mitigate any challenges. Based on the evaluation, we identified that four of the five companies develop all types of features, whereas company E does not develop "duty" features. They do, however, acknowledge that they developed duty features in their early stages of building the key products. In addition to this, we identified that our model helps mitigate three challenges and implications that large software companies experience with feature differentiation and that we previously identified:

Effortless Differentiation: One of the key challenges for companies is to identify whether a feature is a commodity, differentiating or innovative. With the feature differentiation model, practitioners felt empowered to perform this distinction based on the five key characteristics. By passing through the characteristics, practitioners can determine in which of the four quadrants their feature under development belongs to. We illustrate this with the two quotes below.

"Now we talk about these differentiators. This is known now, the wow features... One of the wow features, that we are selling is that we change rules and we demonstrate it in front of the customers."

– Product Owner from Company C

"We have a lot of checkbox features in our products that are only there because the competitors have them, but it is not directly the end user that wants them."

- Product Manager from Company E

Directive Comprehension: With the differentiation between the features ('Duty', Checkbox', Wow and 'Flow') and the development process (Traditional vs. Outcome-Driven), we give practitioners in large software companies the ability to define the right ambition level for a certain type of the feature, the preferred methods of collecting customer feedback, and provide them with instructions on which development activities to focus on.

"More experimentation is done in outcome-driven development. Absolutely in the Flow type."

- Product Manager from Company E

"For Duty and Checkbox features you know what you need to do, and you just have to do enough. You need to say that you have it, but it does not need to work that nicely."

- Product Manager from Company E

Distinguishing Innovative Features: With a clear separation between different types of features, our model enables practitioners to prioritize innovative functionality and invest in relevant activities, e.g. running continuous controlled experiments with customers for "Flow" features, or prioritizing investments into identifying regulation requirements for "Duty" features.

"You define the outcomes and see what activities contribute to that."

- Product Owner from Company C

7 Traditional Vs. Outcome-Driven Development

In the previous sections, we provided guidance on how to differentiate between the different types of features and which development activities to prioritize for each of them. In this section, we illustrate the differences in the development approaches (based on additional learnings during the validation sessions) for the three stages of feature development (Value Identification, Value Realization, and Value Validation). For each of the stages, we provide indications on what is beneficial and a drawback for this type of development. For example, in the Value Identification phase (illustrated with 1 in Fig. 1), we discuss requirements freedom (the extent to which feature teams can interpret and change requirements) of both approaches. Next, we contrast 'Value Realization' phase (illustrated with 2 in Fig. 1) by briefly presenting the differences in the autonomy of the development teams. In the Value Validation phase (illustrated with 3 in Fig. 1 below, we compare the extent to which a completion goal is known to a development team, and how distant they are from the customer data (e.g. feedback that can be used to validate how well the feature satisfies the objectives given by the stakeholders).

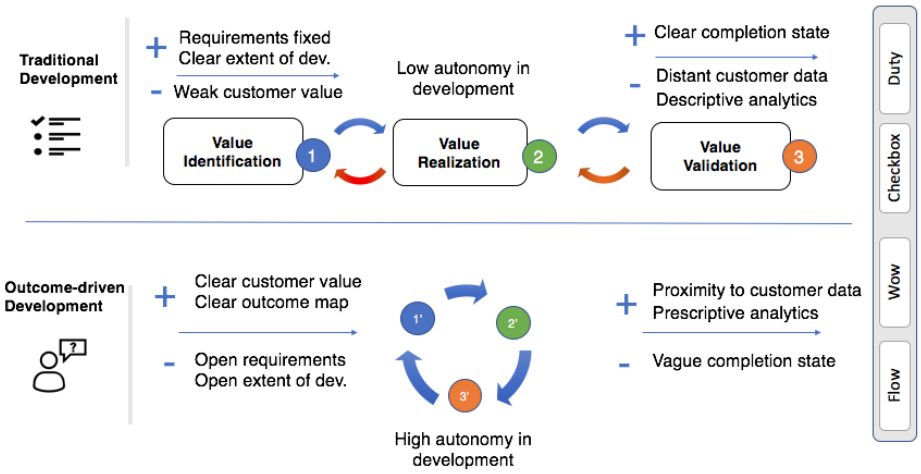

Fig. 1. Feature development in Traditional and Outcome-Driven development.

Traditional Development. Our findings indicate that 'Duty' and 'Checkbox' features development follows a common process which we label 'Traditional Development'. We select this label to emphasize the long-lasting tradition that our case companies followed in their transition from electronics to software companies. Companies that develop features in this way typically spend a considerable amount of time studying the requirements and/or competitors to determine what to develop in the Value Identification phase. Because of known policies, defined standards or competitor's development

outcomes (for example, a feature that was realized), they fix the requirements on what they should develop and information to which extent is given (for example, a safety feature should reach in time under 10ms). This enables teams in the Value Realization phase to focus on meeting the requirements and following the standards and sparing the team from customer representative roles. The teams that develop these types of features have limited autonomy and they do not re-prioritize development activities at execution. This is not necessarily a drawback as certain modern approaches to feature development reject the autonomy of teams and consider it an overhead. Minimizations of the number and significance of the prioritization decisions that the development teams must do makes teams more focused on the development activity at hand. Generally, the development team should be deciding on feature priority only after the priorities of regulations, the business, and the customer have already been addressed. Because of the Stakeholder objectives, the teams in the 'Value Validation' phase can benefit from knowing what exactly satisfies the objectives. The resulting feedback used for evaluation, however, can be very distant. As described in the previous section, features developed this typically are not instrumented with real-time feedback collection. This implies that development teams depend on the time distance of feature integration and deployment to the customer and the time distance of feedback collection, which can be (and in several of features developed by our case companies is) significant. In this development process, our case companies define the following question and strive to answer it: "Are the customers using the system and how?" and answer it using typically qualitative feedback such as observations and interviews, and quantitative raw logs.

Outcome-Driven Development. We contrast the approach above by presenting how 'Wow' and 'Flow' features are being developed. In what we label 'Outcome-driven development' development teams work significantly more iteratively. In the Value Identification phase, and due to the nature of 'Wow' and 'Flow' features, teams invest into identifying what customers and users expect as an outcome and outline a feature idea around this. As a result of studying the value as it will be experienced by the customer and not a requirement, it is very challenging to quantitatively describe the extent to which the feature should be developed. In the Value Realization phase, agile teams embed a customer representative within the development team and the customer representative determines the priorities for development. Therefore, the team has a high autonomy and ownership of the features that they are developing. This is, in principal different than traditional agile teams which follow the backlog items as defined and prioritized by a product owner. This is possible as the Value Validation phase is closely connected with the first two phases due to proximity to customer data. In this development process, our case companies define the following question that they try to answer: "Are the customers efficiently achieving desired outcomes with minimal blocking?" and measure their success with prescriptive analytics (for example experimentation) on customer value metrics (for example task success, time needed to result).

8 Conclusions

In this paper, and based on case study research in three large software-intensive companies, we provide empirical evidence that companies do not distinguish between different types of features, i.e. they don't know what is innovation, differentiation or commodity, which is the main problem that causes poor allocation of R&D efforts and suppresses innovation. To address this challenge, we developed and detailed a model in which we depict the activities for differentiating and working with different types of features and stakeholders. Also, we evaluated the model with our case companies.

With our model, which differs from existing models and similar models (e.g. the Kano model [31]) in that it focuses on software products with rapid customer feedback capabilities, practitioners can (1) categorize the features that are under development into one of the four types and invest into activities that are relevant for that type, (2) maximize the resource allocation for innovative features that will deliver the most value, and (3) mitigate certain challenges related to feature differentiation.

Our model, however, still requires validation on a larger scale to claim its general applicability. The current evaluation is based on qualitative impressions of the practitioners, which is certainly a limitation. In future work, we plan to expand this model by studying how mature online companies differentiate between the different types of features that they develop, how their activities are prioritized, and validate the model using quantitative metrics (e.g. counting the number of features of individual type in each of the case companies).

References

1. Fagerholm, F., Guinea, A.S., Mäenpää, H., Münch, J.: The RIGHT model for Continuous Experimentation. J. Syst. Softw. 0, 1–14 (2015).
2. Denne, M., Cleland-Huang, J.: The incremental funding method: Data-driven software development. IEEE Softw. 21, 39–47 (2004).
3. Boehm, B.: Value-based software engineering: reinventing. SIGSOFT Softw. Eng. Notes. 28, 3– (2003).
4. Khurum, M., Gorschek, T., Wilson, M.: The software value map - an exhaustive collection of value aspects for the development of software intensive products. J. Softw. Evol. Process. 25, 711–741 (2013).
5. Lindgren, E., Münch, J.: Software development as an experiment system: A qualitative survey on the state of the practice. In: Lassenius, C., Dingsøyr, T., and Paasivaara, M. (eds.) Lecture Notes in Business Information Processing. pp. 117–128. Springer International Publishing, Cham (2015).
6. Olsson, H.H., Bosch, J.: Towards continuous customer validation: A conceptual model for combining qualitative customer feedback with quantitative customer observation. In: Lecture Notes in Business Information Processing. pp. 154–166 (2015).
7. Fabijan, A., Olsson, H.H., Bosch, J.: Commodity Eats Innovation for Breakfast: A Model for Differentiating Feature Realization. In: Product-Focused Software

Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings. pp. 517–525. Springer International Publishing, Trondheim (2016).
8. Martin, R.C.: Agile Software Development, Principles, Patterns, and Practices. (2002).
9. Olsson, H.H., Alahyari, H., Bosch, J.: Climbing the "Stairway to heaven" - A mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In: Proceedings - 38th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2012. pp. 392–399 (2012).
10. Mujtaba, S., Feldt, R., Petersen, K.: Waste and lead time reduction in a software product customization process with value stream maps. In: Proceedings of the Australian Software Engineering Conference, ASWEC. pp. 139–148 (2010).
11. Sedano, T., Ralph, P., Sedano, T.: Software Development Waste. In: Proceedings of the 39th International Conference on Software Engineering - ICSE '17. pp. 130–140. IEEE Press, Buenos Aires, Argentina (2017).
12. Goldratt, E.M., Cox, J.: The Goal: A Process of Ongoing Improvement. (2004).
13. Rodríguez, P., Haghighatkhah, A., Lwakatare, L.E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J.M., Oivo, M.: Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study. J. Syst. Softw. (2015).
14. Ries, E.: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. (2011).
15. Fabijan, A.: Developing the right features: the role and impact of customer and product data in software product development, https://dspace.mah.se/handle/2043/21268, (2016).
16. Fabijan, A., Olsson, H.H., Bosch, J.: Customer Feedback and Data Collection Techniques in Software R&D: A Literature Review. In: Software Business, ICSOB 2015. pp. 139–153. , Braga, Portugal (2015).
17. Williams, L., Cockburn, A.: Introduction: Agile Software Development: Its About Feedback and Change, (2003).
18. Bosch-Sijtsema, P., Bosch, J.: User Involvement throughout the Innovation Process in High-Tech Industries. J. Prod. Innov. Manag. 32, 1–36 (2014).
19. Cao, L., Ramesh, B.: Agile requirements engineering practices: An empirical study. IEEE Softw. 25, 60–67 (2008).
20. Olsson, H.H., Bosch, J.: From Opinions to Data-Driven Software R&D: A Multi-case Study on How to Close the "Open Loop" Problem. In: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications. pp. 9–16. IEEE (2014).
21. Castellion, G.: Do It Wrong Quickly: How the Web Changes the Old Marketing Rules by Mike Moran. J. Prod. Innov. Manag. 25, 633–635 (2008).
22. The Standish Group: The standish group report. Chaos. 49, 1–8 (1995).
23. Manzi, J.: Uncontrolled : the surprising payoff of trial-and-error for business, politics, and society. Basic Books (2012).
24. Davenport, Thomas, H.: How to Design Smart Business Experiments,

https://hbr.org/2009/02/how-to-design-smart-business-experiments, (2009).
25. Blank, S.: Why the Lean Start Up Changes Everything. Harv. Bus. Rev. 91, 64 (2013).
26. Kohavi, R., Longbotham, R.: Online Controlled Experiments and A/B Tests. In: Encyclopedia of Machine Learning and Data Mining. pp. 1–11 (2015).
27. Cossio, M.L.T., Giesen, L.F., Araya, G., Pérez-Cotapos, M.L.S., VERGARA, R.L., Manca, M., Tohme, R.A., Holmberg, S.D., Bressmann, T., Lirio, D.R., Román, J.S., Solís, R.G., Thakur, S., Rao, S.N., Modelado, E.L., La, A.D.E., Durante, C., Tradición, U.N.A., En, M., Espejo, E.L., Fuentes, D.E.L.A.S., Yucatán, U.A. De, Lenin, C.M., Cian, L.F., Douglas, M.J., Plata, L., Héritier, F.: A/B Testing - The most powerful way to turn clicks into customers. (2012).
28. Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M.: Controlled experiments on the web: survey and practical guide. Data Min. Knowl. Discov. 18, 140–181 (2009).
29. Kohavi, R., Deng, A., Frasca, B., Walker, T., Xu, Y., Pohlmann, N.: Online controlled experiments at large scale. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1168–1176 (2013).
30. Tang, D., Agarwal, A., O'Brien, D., Meyer, M.: Overlapping experiment infrastructure. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '10. p. 17. ACM Press, New York, New York, USA (2010).
31. Kano, N., Seraku, N., Takahashi, F., Tsuji, S.: Attractive quality and must-be quality. J. Japanese Soc. Qual. Control. 14, 39–48 (1984).
32. Bosch, J.: Achieving Simplicity with the Three-Layer Product Model. Computer (Long. Beach. Calif). 46, 34–39 (2013).
33. Moore, G.A.: Dealing with Darwin: How great companies innovate at every phase of their evolution. Penguin (2005).
34. Fabijan, A., Dmitriev, P., Olsson, H.H., Bosch, J.: The Evolution of Continuous Experimentation in Software Product Development: From Data to a Data-driven Organization at Scale. In: Proceedings of the 39th International Conference on Software Engineering. pp. 770–780. IEEE Press, Piscataway, NJ, USA (2017).
35. Lindgren, E., Münch, J.: Raising the odds of success: The current state of experimentation in product development. Inf. Softw. Technol. 77, 80–91 (2015).
36. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14, 131–164 (2008).
37. Mayring, P.: Qualitative content analysis - research instrument or mode of interpretation. In: The Role of the Researcher in Qualitative Psychology. pp. 139–148 (2002).
38. Wieringa, R., Daneva, M.: Six strategies for generalizing software engineering theories. Sci. Comput. Program. 101, 136–152 (2015).
39. Gregor, S.: The nature of theory in information systems. MIS Q. 611–642 (2006).