Hypotheses Engineering: first essential steps of experiment-driven software development

Jorge Melegati
Faculty of Computer Science
Free University of Bozen-Bolzano
jmelegatigoncalves@unibz.it

Xiaofeng Wang
Faculty of Computer Science
Free University of Bozen-Bolzano
xiaofeng.wang@unibz.it

Pekka Abrahamsson
Faculty of Information Technology
University of Jyväskylä
pekka.abrahamsson@jyu.fi

***Abstract*—Recent studies have proposed the use of experiments to guide software development in order to build features that the user really wants. Some authors argue that this approach represents a new way to develop software that is different from the traditional requirement-driven one. In this position paper, we propose the discipline of Hypotheses Engineering in comparison to Requirements Engineering, highlighting the importance of proper handling hypotheses that guide experiments. We derive a set of practices within this discipline and present how the literature has tackled them up to now. Finally, we propose a set of research questions that could guide future work towards helping practitioners.**

***Index Terms*—hypotheses, assumptions, experimentation, experiment-driven software development, hypotheses engineering, requirements engineering**

I. Introduction

Recently, several studies focused on the use of experiments to guide software development [1]–[3] as a way to build features that indeed create value to customers [3]. By experiments we mean testing product assumptions applying scientific methods with the purpose of supporting or refuting these assumptions [3]. Several methods could be used to run experiments including problem or solution interviews, mock-ups, and A/B tests [3]. According to Olsson and Bosch [2], the lack of experiments systematic use and customer feedback while developing high-quality features could lead teams to frustration and fear when users do not appreciate or use these features. Nevertheless, while exploring the state of practice of experimentation, Lindgreen and Münch [3] realized the need of further studies to integrate experimentation into the development process.

Bosch et al. [4] proposed a framework identifying three different approaches to software development: the "traditional requirements driven development", the "outcome/data driven development", and "artificial intelligence (AI) driven development." In the first, software is built based on specifications provided by the product management or the clients. In the second, decisions are based on data collected through experiments. Lundgren and Münch [3] called it "experiment-driven development." In the third approach, the team uses "artificial intelligence techniques [...] to create components that act based on input data and that learn from previous actions."

In the requirement-driven development, an important component of the process is requirements engineering (RE). According to Nuseibeh and Easterbrook [5], RE is the process of discovering the purpose to what the software was intended, "by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation." In their Requirements Engineering textbook, Kotonya and Sommerville divide RE in the following activities: elicitation; analysis and negotiation; documentation and validation [6].

The rise of agile methodologies brought the need of discussing RE in agile contexts [7] and a large number of studies were performed. In a systematic literature review on the subject, Inayat et al. [8] found 21 studies on practices and challenges for agile requirements engineering. Among them, we highlight user stories that "are created as specifications of the customer requirements" [8]. Therefore, requirements are still the artifact to be created and user stories are a simple way to describe a feature needed by the user.

In requirement driven development, it is widely agreed that poorly-performed RE activities put the project at risk [9]. Based on the similarities between requirements and hypotheses, and the well-known practices for scientific experiments, we argue that badly defined hypotheses leads to waste of time and resources. Up to now, the literature on experimentation and guidelines to identify, prioritize and verify if those hypotheses could be answered by the proposed experiments are still scarce. To start filling this gap, in this position paper, we propose the discipline of Hypotheses Engineering and a set of research questions about its activities.

II. Experimentation models

Based on a scientific literature review, we identified two models for the experiment-driven software development: the HYPEX (Hypothesis Experiment Data-driven Development) proposed by Olsson and Bosch [2] and the RIGHT (Rapid Iterative value creation Gained through High-frequency Testing) model created by Fagerholm et al. [10].

The first was developed based on challenges observed on three case studies in software development companies and consists of six practices. The first practice is to generate features in the backlog that could be valuable to the users

and be experimented with. The second consists of selecting the highest priority feature and creating hypotheses on how the company believes that the feature creates value to the customer. During the third practice, the team extracts the feature smallest part that adds value to the customer (the so-called minimum viable feature or MVF) and instrument it to collect data. In the fourth practice, gap analysis, the company analyze the difference between expected and actual behavior, and, if they are different, the team develop hypotheses about the reason to be used in the following steps. Then, in the next practice, the team analyzes the hypotheses created and prioritize which ones they should test further. If the most probable hypothesis is that the feature does not meet users needs, the next practice will be to create an alternative implementation.

The RIGHT process model follows a similar path. Based on learning obtained in previous cycles and company's business model and vision, the team identifies and prioritizes hypotheses. Then, based on a subset of hypotheses, the team design an experiment implementing an MVP (Minimum Viable Product) or MVF and updating the instrumentation. Later, the experiment is executed and its results are analyzed leading to a decision making stage. From that, the team draws new pieces of learning, taking it to persevere in the idea (implementing/optimizing, scheduling for deployment) or pivot it/change assumptions, etc.

It is important also to highlight that both studies cite the Build-Measure-Learn cycle proposed by Ries [11]. The cycle is one of the building blocks of the author's Lean Startup methodology. According to him, startups should work in these cycles where they first take their assumptions as hypotheses and build the minimum solution to test a hypothesis (Build). Based on metrics (Measure), the team should accept or reject the hypothesis (Learn), that is, persevering or pivoting.

From the models above described, in experiment-driven software development:

- the main artifact is not a requirement but a **hypothesis** or an assumption that could be validated or not in the end of the experiment;
- based on **metrics** collected during the execution of the experiment;
- leading to learning (validated learning in the Lean Startup methodology) that **updates** the original set of hypotheses.

These common elements imply a similar approach to experiment-driven software development. Figure 1 presents this approach comparing it to a requirement-driven one in a simplified manner to highlight the differences between them.

Therefore, it is reasonable to improve methods and techniques to handle hypotheses in a similar way to what has been done to requirements. In the next section, we propose the Hypotheses Engineering discipline and show what has been discussed up to now in the literature that addresses the concerns represented by these activities.

III. Hypotheses engineering

In requirement-driven software development, the success of a software is determined by to which degree it fits the users' desires [5]. That is, it is important to understand users' needs, model, analyze, negotiate and document them, and validate that the documented requirements fit the users' desires [12]. On the other hand, in experiment-driven software development, hypotheses guide the creation of experiments to allow the team to learn about its user or customer. That is, the final goal is not the code itself but learning about the market and user. Therefore, given the limited time and resources, teams should be able to find out the hypotheses which exploring at that moment would take to the most relevant piece of learning.

Teams should be able to identify hypotheses from the business goals, vision and learning previous obtained. Then, prioritize these hypotheses in order to minimize waste of time and resources. Besides that, a hypothesis should have not been evaluated before in the same context and it should be meaningful to the product roadmap, i.e., the team should analyze it. Finally, the hypotheses should be communicated to the development team to have the related experiment performed. Figure 2 summarizes the activities.

In the following sections, we discuss each activity and how they have been presented in the literature. Then, we discuss how different development stages may influence these activities. As for traditional RE, a breakdown may infer a waterfall-like process [9] but this is not the case. In agile RE, all steps are mingled together [13] and it is reasonable to think that the same happens to hypotheses, especially given that experiment results influence following experiments.

A. Hypotheses generation

Similar to RE elicitation, hypotheses generation should describe techniques to gather assumptions to which experiments will be built. Until now, authors have discussed this issue very briefly. In HYPEX, the first practice is feature backlog generation where product management and product development staff "based on their understanding of customer needs and strategic business goals" generate features that may bring value to customers [2]. In the RIGHT model, "analyst and product owner work with a data scientist role [...] to communicate the assumptions of the roadmap and map the areas of uncertainty which need to be tested." Gutbrod and Munch [14] mention that since the Business Model Canvas or the Lean Canvas contain all relevant aspects of a product or business idea, they could be used to get the important assumptions. In this paper, the authors list 22 assumptions for the Airbnb product but they ranged from business viability issues or, as Fagerholm et al. [10] call, high-level questions, like if travelers were willing to rent from strangers and if Airbnb is legal, to more user interface optimization in a later stage of optimization [10] like "the price selection is easier with a sliding scale". In Bosch and Olsson's outcome/data driven development [4], the team "receive a quantitative target to realize and are asked to experiment with different solutions to improve the metric."

Although not deeply explored, a common theme is the guidance of business goals and vision to determine which assumptions have to experimented with. These can range from

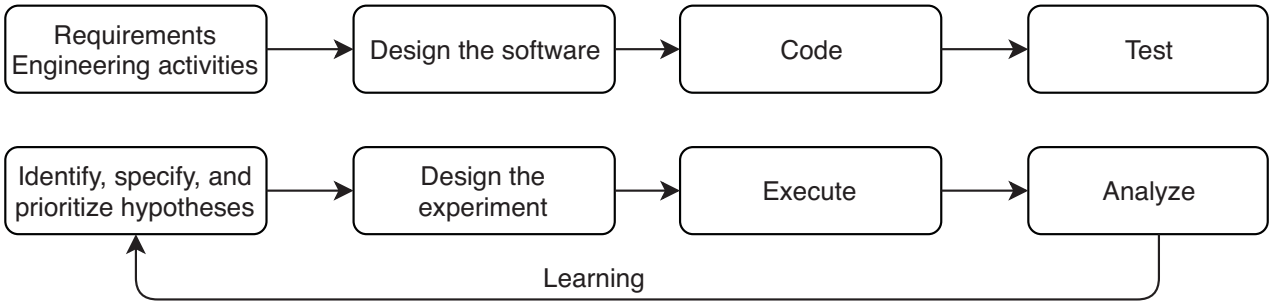

Fig. 1. Comparison between requirement-driven (upper) and experiment-driven (bottom) software development approaches.

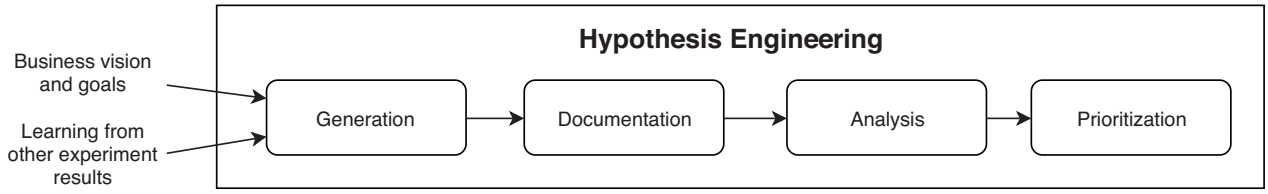

Fig. 2. Activities in Hypotheses Engineering.

new products to new features or feature optimization [3]. The lack of clear guidelines could make hard to practitioners to elicit the most critical assumptions that their products will be based on. Therefore, an important research question is:

RQ1: How can software development teams systematically define hypotheses based on business goals and vision, and own previously accumulated learning?

B. Hypotheses documentation

In traditional requirement-driven development, the documentation stage is essential to guide the implementation and, after that, to check if a feature certainly fulfills the demand [5]. In agile RE, there is no formal documentation and long requirements documents are replaced by user stories [8] that are a way to foster face-to-face communication [13]. Similarly, it is reasonable to think about artifacts, like user stories in agile RE, that could help people involved in experimentation.

Until now, the discussion about artifacts to experimentation is very limited. The RIGHT model mentions experiment plans and learning as information artifacts not necessarily formally documented [10]. In HYPEX, still in the feature selection and specification stage, the team should specify how the feature adds value to the customer and supports the business goals. It also should describe the expected behavior the user will take with the feature and this will guide analysis after the experiment is concluded [2]. Therefore, a valuable study could answer the following research question:

RQ2: What artifact could be useful to represent hypotheses and support experiments creation?

Besides that, the team should update the hypotheses after the experiments results, record the conclusions, and keep them to further queries. Then, another research question would be:

RQ3: How could a hypothesis artifact be used to keep experiment result information?

C. Hypotheses analysis

In a traditional RE process, during the analysis and negotiation phase, the team should analyze, detail and, in the case they are not feasible or beyond project scope, discard requirements [6]. Besides that, before implementing, there is a validation stage when analysts check requirements [6]. They should be complete, that is, they should have all the details to allow the correct implementation of the functionality and be consistent in the sense that they meet the users' expectations [12]. These practices were employed to ensure that requirements implemented really represented users' desires and, consequently, to avoid implementation effort waste. In agile methodologies, this concern is tackled by continuous face-to-face communication between development team and customers and the iterative RE [13]. The use of prototypes is also a way to get customer feedback before moving ahead [8].

While doing experiments, similar concerns can arise: if the hypotheses is not well-explained it could lead to an experiment that will take a lot of resources or users attention and not bring valuable learning. Prototyping experiments could be one possible solution to that. Teams should analyze hypotheses to check if they are consistent and not-duplicated, for instance, another experiment already performed could have answered it. That is, given a series of experiments, a hypothesis could have been already tested previously and doing another experiment will be waste. However, timing is a critical aspect that cannot

be ignored. For instance, a hypothesis valid in the beginning of a product life-cycle can be no longer valid towards the end of it. Then, some research questions arise:

RQ4: How could teams understand if a hypothesis can be practically tested using an experiment?
RQ5: How could teams understand dependencies among different hypotheses?
RQ6: How do hypotheses evolve over the time?

D. Hypotheses prioritization

Prioritization has been a more common theme both from scientific and commercial authors. Such interest could be explained by the fact that experimenting with less critical assumptions first could lead to waste of resources once the more important assumptions are later proven wrong. In a paper about teaching Lean Startup prioritization, Gutbrod and Munch perform a review on techniques to prioritize hypotheses [14]. Among others, they mention the Prioritizing Leap-of-Faith Assumptions (LOFA) by Ries [15], and the Prioritization Matrix by Gothelf and Seiden [16]. In their course, they used the two axes of LOFA, named "time to impact" and "magnitude of impact" in a experiment with students. In HYPEX model, the second step is feature selection and specification where "the team selects the highest priority feature for implementation" [2]. However, there were no systematic evaluation of these approaches.

Besides that, several authors propose different techniques to prioritize requirements like analytical hierarchy process (AHP) and MosCow, but they have several limitations [17]. Their use in experiment-driven software development has not been explored so far either. Then, several research questions arise:

RQ7: Are current assumption prioritization techniques effective?
RQ8: Could requirements prioritization techniques be adapted to hypotheses in experiment-driven development?

E. Development stages

In a literature review on customer feedback and data collection techniques, Fabijan et al. [18] distinguished between three development stages: pre-development, development and post-deployment. Even though they highlighted the iterative nature of most development processes, they identified different techniques used for different stages. In the first stage, companies are testing interesting for the product in the market. Then they evaluate early product concepts with techniques like prototyping and beta-testing. Finally, once the product is deployed, they use techniques to learn about customer behavior and optimize the product. Therefore, it is reasonable to think that different techniques will be used for Hypotheses Engineering in different stages.

IV. Conclusions

In this position paper, we compared requirement-driven and experiment-driven software development approaches. Based on that, we argued the need of a Hypotheses Engineering discipline to contrast with Requirements Engineering. We proposed a set of activities to be included and further developed so teams could find out the most critical assumptions to guide their experiments. For each activity, we suggested at least one research question creating a roadmap to further develop the discipline.

References

[1] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the "Stairway to heaven" - A mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software," *Proceedings - 38th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2012*, pp. 392–399, 2012.

[2] H. H. Olsson and J. Bosch, "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*. IEEE, aug 2014, pp. 9–16.

[3] E. Lindgren and J. Münch, "Raising the odds of success: the current state of experimentation in product development," *Information and Software Technology*, vol. 77, pp. 80–91, 2016.

[4] J. Bosch, H. H. Olsson, and I. Crnkovic, "It Takes Three to Tango : Requirement , Outcome / data , and AI Driven Development," in *Software-intensive Business Workshop on Start-ups, Platforms and Ecosystems (SiBW 2018)*. Espoo: CEUR-WS.org, 2018, pp. 177–192.

[5] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," *ICSE 2000 Proceedings of the Conference on The Future of Software Engineering*, vol. 1, pp. 35–46, 2000.

[6] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*. New York, NY, USA: John Wiley & Sons, Inc., 1998.

[7] A. Eberlein and J. Leite, "Agile Requirements Definition: A View from Requirements Engineering," *International Workshop on Time-Constrained Requirements Engineering, TCRE 2002*, pp. 1–5, 2002.

[8] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in Human Behavior*, vol. 51, no. 0, pp. –, 2015.

[9] I. C. Society, P. Bourque, and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014.

[10] F. Fagerholm, A. Sanchez Guinea, H. Mäenpää, and J. Münch, "The RIGHT model for Continuous Experimentation," *Journal of Systems and Software*, vol. 123, pp. 292–305, 2017.

[11] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.

[12] B. H. C. Cheng, J. M. Atlee, and M. Joanne, "Research Directions in Requirements Engineering," *Proceeding FOSE '07 2007 Future of Software Engineering*, pp. 285–303, 2007.

[13] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," *Information Systems Journal*, vol. 20, no. 5, pp. 449–480, nov 2007.

[14] M. Gutbrod and J. Münch, "Teaching Lean Startup Principles : An Empirical Study on Assumption Prioritization," in *Software-intensive Business Workshop on Start-ups, Platforms and Ecosystems (SiBW 2018)*, 2018, pp. 245–253.

[15] E. Ries, *The Startup Way: How Modern Companies Use Entrepreneurial Management to Transform Culture and Drive Long-term Growth*. Currency, 2017.

[16] J. Gothelf and J. Seiden, *Lean UX: Applying Lean Principles to Improve User Experience*, ser. Lean series. O'Reilly Media, 2013.

[17] P. Achimugu, A. Selamat, R. Ibrahim, and M. N. R. Mahrin, "A systematic literature review of software requirements prioritization research," *Information and Software Technology*, vol. 56, no. 6, pp. 568–585, 2014.

[18] A. Fabijan, H. H. Olsson, and J. Bosch, "Customer Feedback and Data Collection Techniques in Software R&D: A Literature Review," in *Lecture Notes in Business Information Processing*, 2015, vol. 210, pp. 139–153.