Continuously Experiment to Assess Values Early On

The troubles began when Tom, the business analyst, asked the customer what he wants. The customer came up with good ideas for software features. Tom created a brilliant roadmap and defined the requirements for a new software product.

Mary, the development team leader, was already eager to start developing and happy when she got the requirements. She and her team went ahead and created the software right away. Afterwards, Paul tested the software against the requirements. As soon as the software fulfilled the requirements, Linda, the product manager, deployed it to the customer. The customer did not like the software and ignored it. Ringo, the head of software development, was fired. How come?

Most Ideas Fail to Show Value

Nowadays, we have tremendous capabilities for creating nearly all kinds of software to fulfill the needs of customers. We can apply agile practices for reacting flexibly to changing requirements, we can use distributed development, open source, or other means for creating software at low cost, we can use cloud technologies for deploying software rapidly, and we can get enormous amounts of data showing us how customers actually use software products.

However, the sad reality is that around 90% of products fail, and more than 60% of the features of a typical software product are rarely or never used.

But there is a silver lining – an insight regarding successful features: Around 60% of the successes stem from a significant change of an initial idea. This gives us a hint on how to build the right software for users and customers.

Many software projects fail to deliver or only deliver little value due to the wrong assumptions made on requirements. A questionable assumption is, for instance, that customers or experts can come up with the right requirements. In consequence, projects usually have an upfront business analysis phase before the development starts. There are of course projects such as large-scale contract software projects in well-understood domains where upfront analysis is feasible and successful. But we should consider that these projects represent a very small percentage of all software projects.

“If we’re not solving the right problem, the project fails.” – Woody Williams

Nowadays, nearly all software projects are conducted in complex environments where the relationship between cause and effect with respect to features and their success can only be understood in retrospect. Nobody “knows” upfront if and how features will create value for customers. Making decisions on what to develop based on opinions is highly risky in dynamic and non-predictable environments. Developing wrong features creates cost for development and maintenance as well as opportunity cost representing the missed opportunity to develop something of value instead.

A promising way to create software products in complex environments is to quickly and systematically iterate an initial product idea towards success before running out of time and other resources. Simply speaking, this means that you need to create a plan A that describes the scope of the software, identify the underlying assumptions of this plan, test the riskiest assumptions, and iterate until you have a plan B that works. The initial ideas we come up with are seldom successful. Identifying, testing and refining multiple options helps to discover better ways to provide value for users or customers.

Every Idea can be Tested with an Experiment

A means for doing this is continuously conducting experiments to test assumptions and making being wrong cheaper. Insights from experiments directly influence what is given to the users. This process of continuous experimentation consists of three meta-steps:

1. Break down your product idea into a product roadmap that can be efficiently tested. Be aware that the roadmap changes over time and is basically a list of assumptions. Constantly reprioritize the assumptions.
2. Run frequent and additive experiments to test assumptions. This includes systematically observing users' behavioral responses to stimuli such as features. An example for a hypothesis is “The new posting feature will increase sign ups of new users by 5% in two weeks“. If an experiment does not deliver the expected result, do not test another option at random. Carefully choose what to test next.
3. Use results from experiments to iteratively modify your product roadmap. This might lead to an improvement of a product or a significant change of the strategy. It might also mean that you need to stop the project.

Success cases from companies such as Etsy, Amazon, and Supercell show that an hypothesis-driven development approach helps companies to gain competitive advantage by reducing uncertainties and rapidly finding product roadmaps that work. However, experimentation is hard.

How do we Find Good Hypotheses and Conduct the Right Experiments?

Customers and users are a questionable source for novel ideas. What they say often does not match what they will do. Consider the wish of users for privacy and the way they use Facebook. However, customers often have a good understanding of problems and asking the right questions can help reveal good hypotheses in the problem space.

Developers are usually good at coming up with solution proposals. They are familiar with technical options for solving a problem and can be a good source for revealing hypotheses in the solution space. Creating a “UserDevs” community by intensifying the communication between users and developers promises to be another good source for hypotheses.

A further source for identifying hypothesis is usage data. It can be used to gain insights and new ideas on what to develop if the right data is collected and appropriately analyzed. Further hypotheses to test, often hidden and not directly visible, can be found in the respective business models.

"One test is worth one thousand expert opinions." - Wernher von Braun

What about the HiPPOs? HiPPOs are the highest paid person's opinions. HiPPOs currently dominate decisions about what to develop. However, there is no guarantee that their ideas are better or successful. Listen to HiPPOs and take their ideas into account when prioritizing what to test. But make development decisions based on validated assumptions.

"It's not an experiment if you know it's going to work.” - Jeff Bezos

The experimentation process follows the scientific method. It is important that you state upfront what you expect. Otherwise you just see what is going on. And many people are excellent at rationalizing what they see and would be surprised if they would have stated their expectations upfront.

There are many techniques available that support experimentation such as multivariate tests, prototyping, or customer interview techniques. But consider that choosing the right experiment technique requires that you know what you want to learn. Do you want to better understand the problem? Do you want to test the feasibility of a solution? Do you want to compare solution alternatives? Do you want to understand a behavior change? Do you want to test the efficiency of a distribution channel? All these questions lead to different experiments.

Overall, continuous experimentation requires a deep integration of testing critical assumptions in the overall development process. It emphasizes rapid and constant learning by empirical means in order to create software that provides value for users, customers, and the developing organization. Success with software is not luck. We all have the opportunity to deliver high-value software. What is your most critical assumption?

Key Takeaways

- It’s more important to do the right thing than to do things right. – Peter Drucker
- Success in highly dynamic application domains traces back to disciplined experimentation.
- Defining and running the right experiments is hard.
- Experimentation must be deeply integrated in the design and product development process.
- Platforms for experimentation can be seen as a core part of future development environments.
- Data Scientists can play a critical role by supporting the planning and execution of experiments as well as by ensuring their quality.

References

Eveliina Lindgren, Jürgen Münch. Software Development as an Experiment System: A Qualitative Survey on the State of the Practice. In Proceedings of the 16th International Conference on Agile Software Development (XP 2015). Springer-Verlag, 2015. Sketchnotes.

Fabian Fagerholm, Alejandro Sanchez Guinea, Hanna Mäenpää, Jürgen Münch. Building Blocks for Continuous Experimentation. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering (RCoSE 2014), Hyderabad, India, pages 26-35, June 2014.

Ash Maurya. Running Lean. O'Reilly, 2012.

Helena Holmström Olsson, Jan Bosch, From Opinions to Data-Driven Software R&D: A Multi-case Study on How to Close the 'Open Loop' Problem. 40th Euromicro Conference on Software Engineering and Advanced Applications, pp. 9-16, 2014.

Eric Ries. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Publishing, 2011

Scott Anthony, David Duncan, and Pontus M.A. Siren. The 6 Most Common Innovation Mistakes Companies Make. Harvard Business Review, June, 2015.