BITS Pilani

SEZG507: Product Discovery & Requirements Engineering

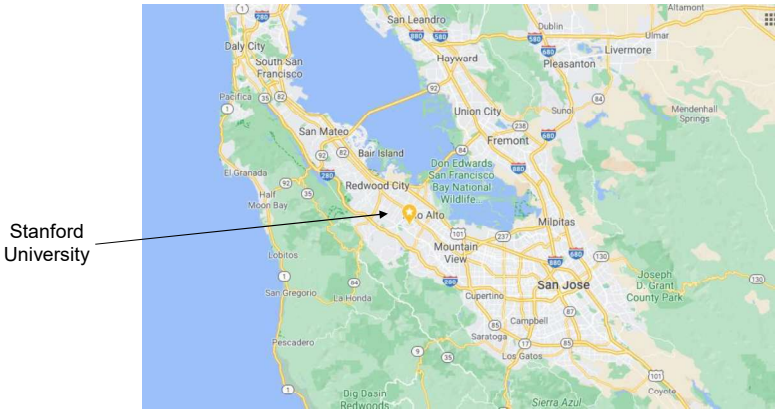
Session 01: Introduction

Contents

- Software products scenario
- What fuels the software product industry?
- Different product categories
- Project business vs product business
- What is product management?

Software products scenario

Software product revolution started in the Silicon Valley



Silicon valley

Early companies in Silicon Valley: HP, Xerox, Apple, Oracle,…..

How did Silicon Valley become successful?

- Convergence of academia (Stanford, UC Berkley), the private sector, and government
- High density of wealthy investors and funding institutions
- Inspiration from past success stories
- Cultural diversity: Half the startups belong to Indians and Chinese
- Level-headed approach to failure

Unicorns across the world

Country	# of Unicorns
United States	300+
China	140+
India	50+
United Kingdom	30+
Germany	12
South Korea	11

https://www.cbinsights.com/research-unicorn-companies

Unicorns by industry

Industry	# of Unicorns
Fintech	80+
Internet software & services	70+
E-commerce & direct-to-consumer	70+
Artificial intelligence	50+
Mobile & telecommunications	35+
Health	35+

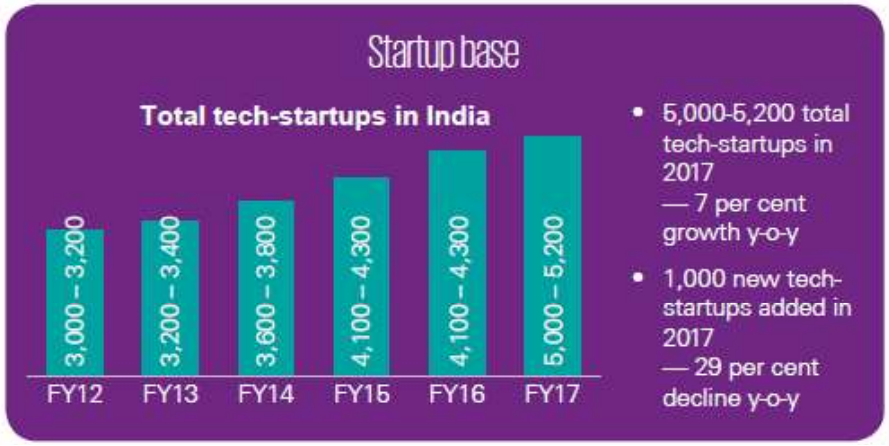
https://www.cbinsights.com/research-unicorn-companies

Growth of start-ups in India

The number of start-ups has grown from 7,000 in 2008 to 50,000 in 2017, according to a report by **KPMG on the startup ecosystem in india**

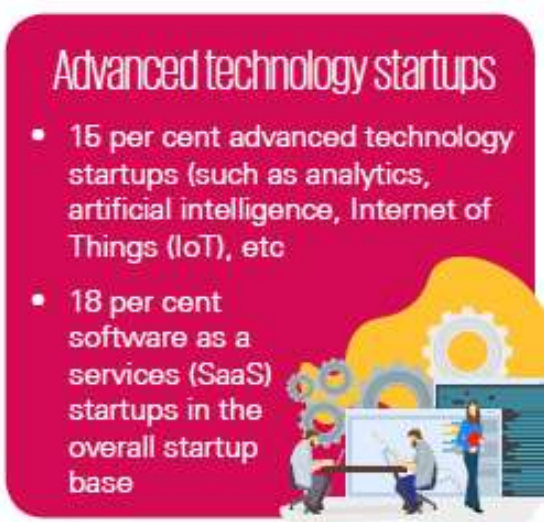
KPMG report: https://home.kpmg/in/en/home/insights/2019/01/startup-landscape-ecosystem-growing-mature.html

Tech start-ups growth in India



KPMG report: https://home.kpmg/in/en/home/insights/2019/01/startup-landscape-ecosystem-growing-mature.html

Tech start-ups using advanced technology (India)



KPMG report: https://home.kpmg/in/en/home/insights/2019/01/startup-landscape-ecosystem-growing-mature.html

Tech start-ups job creation (India)

KPMG report: https://home.kpmg/in/en/home/insights/2019/01/startup-landscape-ecosystem-growing-mature.html

Examples of products

- Zoom – Simplified Cisco Webex
- Ola – Built a platform
- Postman – Eco system for API development
- Slack – Simplified Team collaboration
- Twilio – Tool to Integrate messaging
- Kissflow – Business workflow implementation easily
- Rivigo – Innovation in logistics
- MyGate – Spotted an opportunity

What fuels the software product industry?

- Global market reach
- Cloud resources – Amazon AWS, Microsoft Azure, IBM, Google
- Funding - 100 angel investors in 2020
- Talent pool

Product categories

Product categories

- By industry – Finance, Health, Retail, Travel
- By technology – AI/ML, Analytics, Robotics, IoT
- B2B vs B2C
- SaaS vs On-premise
- Mobile vs Web
- Regular vs API products (Payment gateway, Google Maps, SMS gateway, Banking API)
- Product vs Product-cum-service (Ola, Uber, Flipkart)
- Product (Paytm), Product platform (Ola), Product family (Office on Windows, Office on Mac, Office on Android), Product Line (Rockwell Collin Avionics)
- Any other?

Industry segments

- E-Commerce – Amazon, Flipkart
- HealthTech – Practo, Tata Health, CogniAble
- FinTech – Paytm, Wealthy
- EdTech - Byju
- TravelTech – MakeMyTrip, Tripadvisor
- Logistics – Ecom express, Dunzo, Delhivery
- Consumer services – Swiggy
- EnterpriseTech – Zoho, Kissflow, Wooqer
- DeepTech - Niflr, Logically, AskSarkar
- Software Development – Postman, WorkDuck

Product platform

Product platform: Amazon AWS, Android, Uber, PayPal, Facebook

- The technical foundation or ecosystem on which several software products are based.

Product platform

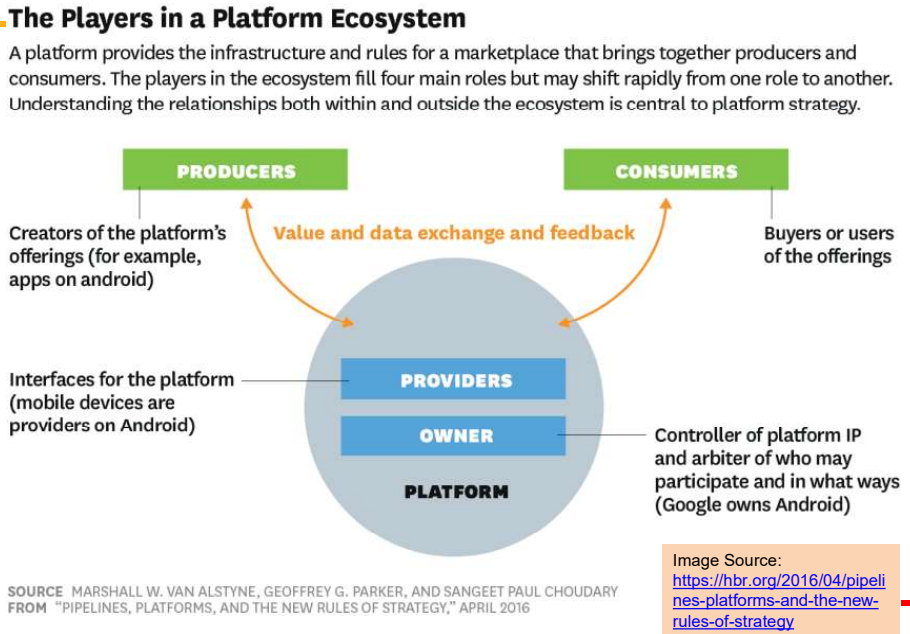


Image Source: https://hbr.org/2016/04/pipelines-platforms-and-the-new-rules-of-strategy

Product family

Product family: Microsoft Office (Word, Excel, PowerPoint, OneNote, Outlook)

- A group of software products that are marketed as belonging together under a common family name

Product line

Product line: Rockwell Collins Avionics systems for different helicopters

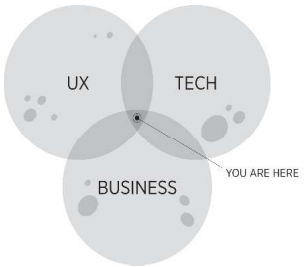
- a collection of similar software systems from a shared set of software assets using a common means of production.

Product business and project business

Dimension	Product	Project
Risk	High	Low
Returns	High	Low
Duration	Ongoing	Pre-determined
Customers	Many	One
Objective	Discovered	Given
Funding	Internal & external	Internal
Marketing effort	High	Low
Management	Strategic	Tactical/operational

What is product management?

- "The job of a product manager is to discover a product that is valuable, usable and feasible." – Marty Cagan, Author of 'Inspired'
- "Product management is an intersection between business, user experience, and technology" – Martin Eriksson, Author of Product Leadership



- "Product management is the glue that holds together all the various functions" - Ken Norton, Product Partner at Google Ventures

Product management role

- You need to be really good at strategy, be inspirational, and understand the long-term picture.
- At the same time, you have to be really good at the operational side and making things happen
  - Setting a vision
  - Creating a roadmap
  - Build the product
  - Talk to customers
- You need the soft skills of persuasion, negotiation, storytelling, vision setting and communication

Product management role

Ref: Product Plan.com

Sharing thoughts

- Name one product company you admire.
- What is the reason you admire this company?

Evolution of product organizations

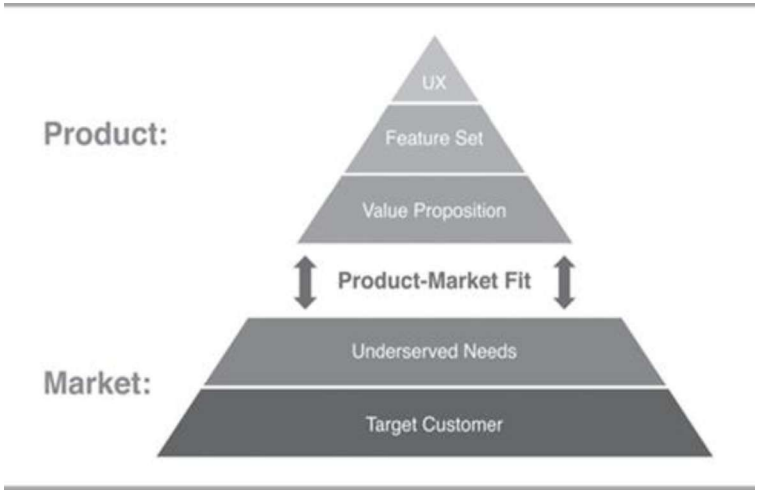
- A product organization goes through the following stages:
  - Startup
  - Growth stage
  - Enterprise
- Let us see what are the characteristics of each stage

Startup stage

- Trying to achieve product-market fit
- Limited funding
- Learns quickly
- Little bureaucracy
- Many fail
- Those that succeed are good at product discovery
- Risky but rewarding if things go well

Examples: WhiteHat Jr, Simpl

Product-Market fit concept



Thank you

BITS Pilani

SEZG507: Product Discovery & Requirements Engineering
Session 02: Overview of Product Management

Start Recording

Please read the course handout, if you haven't already ☺

Contents

- Evolution of product organizations
- Why products fail?
- What do best product teams do?
- Product management: Relationship with rest of the company
- Product lifecycle
- Technology adoption lifecycle
- Journey of some product companies
- Multi-faceted role of a product manager

Evolution of product organizations

A product organization goes through the following stages:

- Startup
- Growth stage
- Enterprise

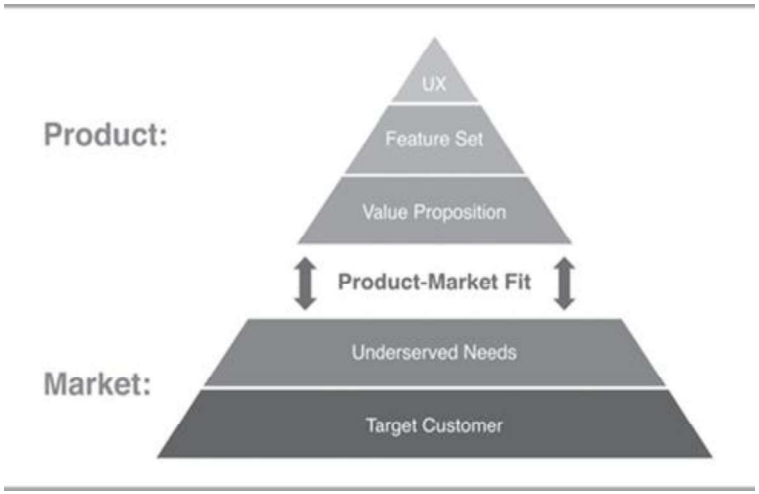
Let us see what are the characteristics of each stage…

Startup stage

- Trying to achieve product-market fit
- Limited funding
- Learns quickly
- Little bureaucracy
- Many fail
- Those that succeed are good at product discovery
- Risky but rewarding if things go well

Examples: WhiteHat Jr, Simpl

Product-Market fit concept



Startup stage examples

WhiteHat Jr

- Founded in 2018
- Offers coding & AI courses to children aged 6 to 14 years.
- Aims to empower children to become creators
- BYJU's acquired it for $300 million

Simpl

- Started 2016
- Online payment method that allows a consumer to buy now and pay later
- Digitalizing the old Khata system of payment to grocer, milkman, etc
- Simpl under-writes customer payments based on machine learning
- USP: Transparent financial services and single click payment

Growth stage

- Scale up – more customers
- Replicate earlier successes with new, adjacent products and services; e.g. MakeMyTrip - flight, train, hotel
- Technology infrastructure is stretched (Netflix during the growth stage)
- There is technical debt (Amazon monolithic to microservices)
- Goes for IPO or gets sold (MakeMyTrip IPO, WhatsApp sold to Facebook)

- Examples: Bounce (2016), Postman (2014), KissFlow (2013).

Growth stage example

Kissflow

- Business process management software
- Self-service setup/configuration
- 50 process templates to choose from – employee on-boarding, travel reimbursement
- Strong after sales support
- Product led growth - leading to pull rather than push
- 10,000-plus clients, including biggies like Airbus, Danone, Michelin and Pepsi
- Competitors - Pega, Appian, Outsystems
- 200 employees

Enterprise stage

- Focus is on consistent product innovation, stay ahead
- But many companies are satisfied with leveraging the value created and brand created, leading to slow death (ex. Kodak)
- They work hard to protect what they have created and less on new ventures and initiatives
- There is lack of vision, increased bureaucracy, resorts to acquisitions or creating separate innovation centres to incubate new business or products (example Cisco).

- Companies that failed to innovate: Xerox, AoL, Motorola
- Strong enterprise companies: Adobe, Amazon, Apple, Facebook, Google, and Netflix

Enterprise stage: Examples of consistent innovation

Netflix	Amazon	Facebook
• DVD movie sales • DVD rentals • Online booking of DVD, delivered via Post • Streaming video • In house production of serials and movies • Movie/serial award function (akin to Oscar)	• Books • Electronics, Others • Recommendation feature • Amazon Prime • Alexa • Kindle • AWS • Firestick • Amazon Pay	• Wall & messaging • News Feed - streams friend's activity • Sell stuff to other Facebookers • Tagging and attachments • 'Like' button • Timeline feature • Buys Instagram, WhatsApp

Why products fail?

- Most companies start with ideas generated internally or got from existing or potential customers.
  - Example: HP's AI-enabling technology on a low-cost, general-purpose workstation developed by Marty Cagan & team (1980s), DB designer – I worked on (1989)
- Based on these ideas they create a business case, roadmap, build the product and deploy
- It is then that they realize that there are no takers

- More examples of failed products:
  - Apple Watch Gold edition
  - Google+ social media
  - The Daily - Digital newspaper in collaboration with Apple

What do best product teams do?

- Tackle risks early
- Define and design products collaboratively – Product Manager, Designer, Engineering
- Solve problems, not just implement features

Tackle risks early

There are four types of risks:

- Value – Does customer find value in the product?
- Usability – Is the product easy to use?
- Feasibility – Is the product technically feasible to build?
- Viability - Will the business be viable, can it break even?

Tackle risks early - Examples

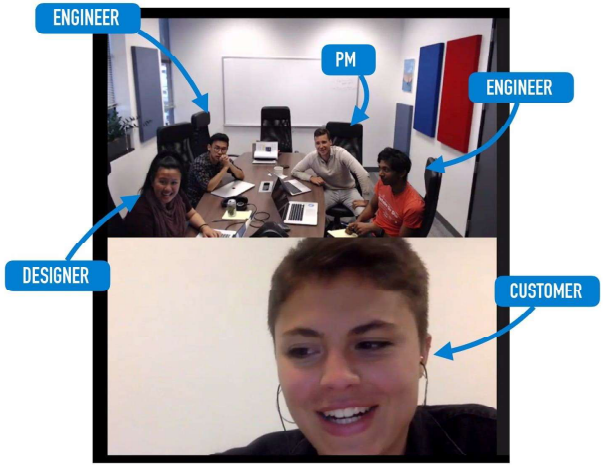
- Bounce
  - Bounce spotted an opportunity in Bangalore: Provide scooter to reach the nearest metro station
  - Bounce experimented their concept with a few scooters to determine value. Once the demand/value was established, they expanded
- AirBnB
  - AirBnB rented their house to test value. A conference was being held in their city and people would be looking for accommodation
- Slack
  - Slack requested friends and cajoled 6-10 companies, to use their product and give feedback to determine usefulness/value and usability and improved the product based on user feedback.

Define and design products collaboratively

- Product, design analytics engineering work side by side in a give-and-take manner
- Leads to better solution ideas and higher ownership

Example: Amplitude

- A product analytics software
- Engineers stay connected with customers by participating in client calls



Solve problems, not just implement features

Example: Kissflow

- Workflow automation improves employee productivity
- Provides 50 ready-to-use workflows from travel reimbursements to employee on-boarding
- Easy diagramming helps model a company's process just as it appears in the business manager's mind.

Solve problems, not just implement features

Example: Wobot Intelligence

- Helps organizations in the Food, Retail, and Manufacturing sectors to reduce risk of non-compliance & pilferage
- Has process compliance modules like hygiene, workforce & workplace safety, customer SOPs, and more
- Uses deep learning video analytics to identify people, objects and their activities
- Customers - IRCTC, Rebel Foods, CureFit, Kitopi, Travel Food Services, Burger Singh, G4S, Max Estates, Blue Tokai, Apparel Group and Smartworks

Solve problems, not just implement features - Example

Example: Logically

- Detects fake news & inaccurate news using AI & ML
- Finds out who is spreading misinformation to enable authorities to take action

- Examples:
  - Detected misinformation during the death of a Bollywood actor Sushant Singh, during conflict with China in Ladakh, and during the Kashmir issue with Pakistan.
  - Detected bots originating in Pakistan that were interfering with geopolitical and sensitive issues within India

- Customers: Indian Election Commission, pharma companies to prevent anti-vaccine information, Mysore Police

Product Management: Relationship with rest of the company

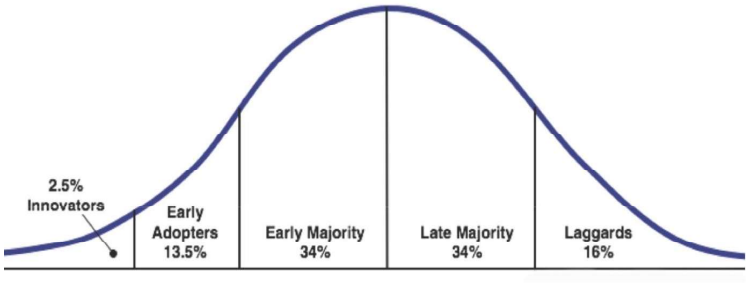
- Development team relies on Product Management to define a plan and write user stories, requirements, and acceptance
- Marketing team relies on Product Management for product information, value proposition definitions. They collaborate to define product position, launch product, define go-to-market strategy
- Sales team relies on Product Management for demo cases, answering detailed inquiries, and helping to close deals.
- Finance and Product Management rely on each other to build the business through determining pricing, margins, discounting, and so forth.

Product lifecycle

- Determine your target customers
- Identify underserved customer needs
- Define your value proposition
- Assess value through customer interaction
- Specify your Minimum Viable Product (MVP)
- Create your MVP prototype
- Test your MVP with customers
- Iterate
- Launch product and support
- Grow and build adjacent products
- End of life

Technology adoption lifecycle

Products using new technology such as AI, NLP, Blockchain, Robotics are adopted gradually



Technology adoption lifecycle…

- **Innovators** are the first to get interested in new products and novelties. They even accept incomplete or defective products just for the pleasure of being the first ones to use this new product.
- **Early adopters**, also known as visionaries or enthusiasts, who accept the risks of testing a new product, but not for the pleasure of coming first but **because they see the potential in it**. Usually, they are influencers within organizations and communities in which they participate.
  - IBM Watson was adopted by Memorial Sloan–Kettering Cancer Centre, Cleveland Clinic, MD Andersen Cancer Centre, to get advise on cancer
- **Early majority**, also called pragmatic, buy new products only after they got references.
  - Manipal Hospital Bangalore, Georgia Tech teaching assistant, H&R Block for tax preparation, several startups use it for developing cognitive apps
- **Late majority** are the conservatives, in other words, those who buy only after the price has dropped substantially. Example late majority users of SalesForce
- **Laggards**, who only buy a new product if this is the only option available.

Technology adoption lifecycle…

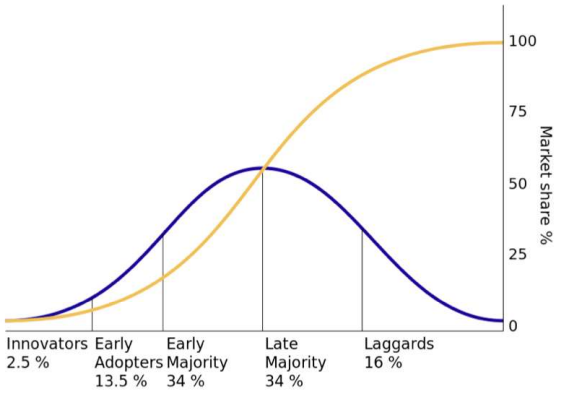
Example:

- IBM Watson and Robotic surgery (Da Vinci) used by one or two hospitals.
- In 1999 Salesforce.com was the first to use Cloud to offer applications on the Cloud. Three years later the industry grew massively with video, music and other media being hosted and delivered online.

https://www.scality.com/solved/the-history-of-cloud-computing/

Technology adoption lifecycle…

S-curve: By calculating the integral (who remembers the calculus classes?) we can obtain the famous S-shaped technology adoption curve.



Multi-faceted role of a Product Manager

- Deep knowledge of customer, your business, market & industry
  - Nium - money transfer to foreign countries
  - Had deep knowledge of money transfer markets in Singapore, Indonesia, Japan, etc.
  - Had good knowledge of forex – how it works, who are the players, banking
- Engage with customers, understand their business, process, pain points
  - Slack understood the collaboration needs of teams
  - Twilio understood the messaging needs of companies
  - Wobot understood the process compliance needs of food, pharma, retail industries
- Prioritize ideas, features and projects
  - Slack focused on Search, synchronization, file sharing
- Collaborate with Design, Engineering, Marketing, Legal, Finance
- Recruit, train and develop the product team
- Manage upward and outward: Tell a story, sell a vision, get funding
- Align and focus the organization

Journey of some product companies: Exercise

Study the journey of Netflix and identify:

- Key milestones?
- Challenges faced?
- What did they do right and what did they do wrong?
- Key product management learnings

- Courtesy: https://www.businessmodelsinc.com/exponential-business-model/netflix/

Thank you

Contents

- Principles of product management
- Characteristics of a holistic product
- Product-market fit
- Problem space vs. solution space
- User vs. buyer
- Continuous discovery and delivery
- Product ecosystem
- Critical success factors
- Introduction to product research

Principles of product management

- Establish compelling value; examples:
  - MakeMyTrip – A one stop shop for travel
  - Postman – Make API testing easy
- Many of the ideas won't work out, and the ones that do will require several iterations; examples:
  - Slack – Initially developed a multi-player online game which did not succeed, but the inbuilt messaging feature became successful
  - MakeMyTrip – Initially targeted Indian travellers, but was not successful; later targeted NRIs
- We must validate our ideas on real users and customers; examples:
  - Bounce – Validated the 'Rent-a-bike' idea by investing in a few scooters
  - AirBnB – Rented their apartment to conference attendees
- Validate ideas fast and with minimal cost – the more we delay, we may be expending more effort and cost on an idea that does not have a market

Discussion point

What has been *your* experience with applying any of these principles?

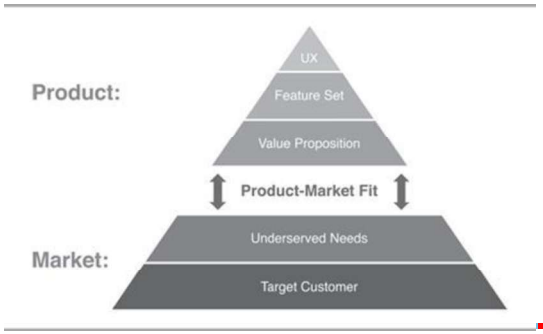
- Establish compelling value; examples:
  - MakeMyTrip – a one stop shop for travel
  - Postman – Make API testing easy
- Many of the ideas won't work out, and the ones that do will require several iterations; examples:
  - Slack – Initially developed a multi-player online game which did not succeed, but the inbuilt messaging feature became successful
  - MakeMyTrip – Initially targeted Indian travellers, but was not successful; later targeted NRIs
- We must validate our ideas on real users and customers; examples:
  - Bounce – Validated the 'Rent-a-bike' idea by investing in a few scooters
  - AirBnB – Rented their apartment to conference attendees
- Validate ideas fast and with minimal cost – the more we delay, we may be expending more effort and cost on an idea that does not have a market

Different aspects of a product

- Functionality; Example: Booking tickets is one function of MakeMyTrip
- Technology; Example: Microservices architecture used by Amazon, encryption used by WhatsApp, AI/ML used by Logically
- User experience (UX); Example: Tally's ease of use for non-finance people)
- How do we monetize? Example: Through transaction fee of something like payment gateways or subscription fee of SalesForce
- How we attract & acquire customers? Example: Freemium of Zoom, cash back of Paytm, search engine optimization, ads
- Offline experience; Example: Merchandise fulfilment experience and merchandise return experience of Amazon & FlipKart, support experience by call centre personnel, self help material on website

Product-market fit

- It is about how well the product meets the needs of the customer (market)
- Good product-market fit results in happier customers, lower churn rates, shortened sales cycles, and rapid organic growth.
- You can always feel when product/market fit isn't happening.
  - The customers aren't quite getting value out of the product, word of mouth isn't spreading, usage isn't growing that fast, press reviews are not enthusiastic, the sales cycle takes too long, and lots of deals never close.



Product-market fit

- Marc Andreessen coined the term *product-market fit* in a well-known blog post titled "The only thing that matters." (https://pmarchive.com/guide_to_startups_part4.html)
- In a great market -- a market with lots of real potential customers -- the market *pulls* product out of the startup; example:
  - eCommerce, EdTech, FinTech
- Conversely, in a terrible market, you can have the best product in the world and an absolutely killer team, and it doesn't matter -- *you're going to fail*; example:
  - Video conferencing (2007), Iridium satellite phone
- Great products sometimes create huge new markets: examples:
  - Virtual machine by VMWare, smart phone by Apple
  - Any other?
- The only thing that matters is getting to product/market fit.

Discussion point

Do *you* know of any great product that failed?

- Marc Andreessen coined the term *product-market fit* in a well-known blog post titled “The only thing that matters.” (https://pmarchive.com/guide_to_startups_part4.html)
- In a great market -- a market with lots of real potential customers -- the market *pulls* product out of the startup; example:
  - eCommerce, EdTech, FinTech
- Conversely, in a terrible market, you can have the best product in the world and an absolutely killer team, and it doesn't matter -- *you're going to fail*; example:
  - Video conferencing (2007), Iridium satellite phone
- Great products sometimes create huge new markets: examples:
  - Virtual machine by VMWare, smart phone by Apple
- The only thing that matters is getting to product/market fit.

Problem space vs solution space

- Problem space consists of customer needs and pain points.
- However problems are not always easy to know:
  - Customers express their needs in terms of existing solutions
  - For example they say “I need a cab in 5 minutes”, because they think cab is the only solution
  - The real need is to go from A to B.
- There can be many solutions for this:
  - Hire a cab
  - Use self-driving scooter or car
  - Hail a bike taxi
  - Any other solution you can think of?
- Therefore before finding a solution, we need to understand the real need/problem
  - Understand what the customer needs and why
  - Observe what (s)he does, why (s)he does it etc. (Persona)
  - “If I had only one hour to solve a problem, I would spend up to two-thirds of that hour in attempting to define what the problem is.”

Problem space vs solution space contd.

What differentiates one product from another is the quality of solution; examples:

- Space pen: Need is to write in space. US designed an ink pen that works in zero gravity; Russians used a simple pencil.
- Progressive auto insurance: Customer wanted quick settlement of car insurance claim; a process that took 6-7 days was cut down to 1 day through innovative solution.
- MoveWorks: Users need quick IT support to install say a project management software. Solutions can be: Raise a ticket, Call IT support, Use MoveWorks bot which will check your eligibility and download the software and install it instantly.
- Application maintenance service: Need to decide whether faster problem resolution is the need or minimal problems is the need.

Discussion point

What do you think can be another example?

What differentiates one product from another is the quality of solution; examples:

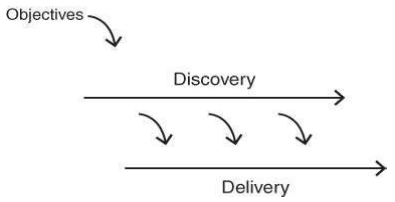
- Space pen: Need is to write in space. US designed an ink pen that works in zero gravity; Russians used a simple pencil.
- Progressive auto insurance: Customer wanted quick settlement of car insurance claim; a process that took 6-7 days was cut down to 1 day through innovative solution.
- MoveWorks: Users need quick IT support to install say a project management software. Solutions can be: Raise a ticket, Call IT support, Use MoveWorks bot which will check your eligibility and download the software and install it instantly.
- Application maintenance service: Need to decide whether faster problem resolution is the need or minimal problems is the need.

User vs. buyer

- In large enterprises the decision makers are not the end users
- Decision makers are usually part of the senior management. They want to solve a business problem or pain point.
- Their concerns are delivering functionality that brings business value (increase customer satisfaction, customer growth, reduce customer churn), productivity, security, reliability / stability / quality of solution
- The end users typically do not have the authority to commission the product. But ultimately they are the ones who are going to use the product.
- Hence the product needs to be user friendly and efficient in performing its functions.
- Example
  - Lotus Notes: It was a very secure team database and email system. But not very user-friendly.
  - Cisco WebEx – very reliable but not very user friendly. But corporations prefer it.
  - Do you know of any other example?
- But this is changing with SaaS product management becoming more aware of UI/UX

Continuous discovery and delivery

- Discovery and delivery are our two main activities for a cross-functional product team, and they typically go on simultaneously.
- We are always working in parallel - to both *discover* the necessary product to be built—which is primarily what the product manager and designer work on every day—while the engineers work to *deliver* the-quality product.
- The engineers are also helping daily in discovery (and many of the best innovations come from that participation, so this is not a minor point), and the product manager and designer are also helping daily on delivery (mainly to clarify intended behaviour).
  - Examples: Postman, Slack
- Does this happen in a company you know of?



Product ecosystem

Product should address the total customer experience (the whole offer)

- Kaagaz & MS Office Lens (document scanner app on mobile) does not only scan but allows us to share the image via email, WhatsApp etc. Because the customer is not just interested in scanning and storing, (s)he wants to share with others
- Xerox started with photo copying facility but soon realized people need to staple the pages, need cover page in different colour, etc. So they enhanced the machine to address the total customer experience
- Clarify: customer support software that involves tracking customer interaction, product details, knowledge base, workflows
- No Broker.com: Find house, pay advance, get painter, get packer & mover
- Have you come across other products that address total customer experience?

Product ecosystem contd.

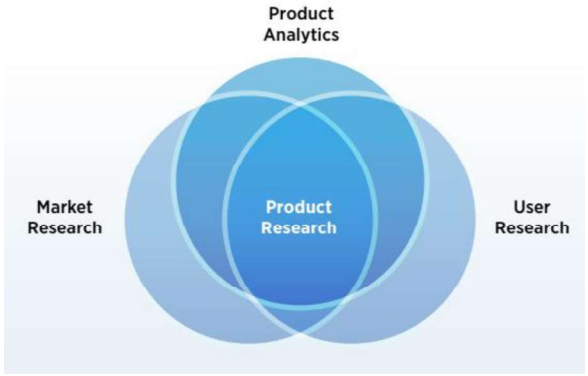
Creating partnerships & alliances

- Xerox tied up with paper manufacturers to ensure steady supply of paper
- SAP partners: DataXtream for POS solutions, DocuSign for eSignature integration with SAP
- Netflix tied up with telecom service providers such as Verizon, Airtel to host their content at ISP gateways, so as to ensure fast response time to customers
- MakeMyTrip built alliances with airlines, hotels, etc.
- Have you come across any other examples?

Critical success factors

- Differentiation
  - Intuit: UI and features
  - Apple: UX
  - Citibank: Reliability & infrastructure
  - .Net: Ease of use
  - Toyota: Quality
  - ISRO: low cost satellite launches for world-wide customers
- Entry barrier
  - Google Earth: Entry barrier due to technology
  - Da Vinci Robotic Surgery: Technology
  - Microsoft HoloLens: Mixed reality technology for doctors

Product research



Product research is an approach that draws from user research, market research, and product analytics to help any product team arrive at insights in a timely, continuous manner.

Thank you

BITS Pilani

SEZG507: Product Discovery & Requirements Engineering

Session 04: Introduction to Product Research

Contents

- Product research
- Excuses for not doing research
- When do you do product research?
- Building on different research disciplines
- Market research
- Product analytics
- A set of rules for product research
- Rule 1: Prepare to be wrong

Start Recording

Product Research

Product research is an approach that draws from user research, market research, and product analytics to help any product team arrive at insights in a timely, continuous manner.

Excuses for Not Doing Research

- It takes too long
- We don't have the budget
- We decided what the user needs
- We're not researchers
- The product is completely new
- It's just a small change
- We need the features first
- It's not the right time
- We don't have many users to test with
- We have enough data
- We'll learn during the pilot

When Do You Do Product Research?

- **Stage 1** is exploring the value of products or features in the market. This is the phase where you are discovering deeper needs in a broader context. In many cases, you don't even have a plan to build something: you're just trying to find out whether it's a good idea. At this stage, you are trying to understand the problem space. Have you understood the problem correctly? Are you considering the right solutions? Are you planning to build the right solutions for the problem you understand?
- **Stage 2** is the development of the product or feature. Here, research helps you stay on course and allows you to assess the right approach. Your results might invite you to explore alternatives. Now that you're immersed in the problem, why are you struggling with certain aspects? Do the assumptions you made at the beginning still hold true? Are you building the solution the right way?
- **Stage 3** comes after you have released your product or feature or when you're working on refining existing features. Research at this stage helps you observe the change in your users' behavior. Now you can check your assumptions directly with the users and see how their needs are changing because of your product or service.

Different Research Disciplines

- Product research draws from different research disciplines, namely
  - User research
  - Market research
  - Product analytics
- There is some overlap between these disciplines, each discipline has a different focus
- Each discipline has subdisciplines that are specialized for particular types of research

User Research

- User research studies what a user does with and surrounding the context of a product's use
- It is about working with real humans to understand their motivations, behavior, and needs
- It aims to understand how that person employs your product and what happens before, during, and after that experience
- User research can be broken down into three categories
  - **Generative user research**: Aims to get a deep, rich understanding of user needs and desires: users' behaviors, attitudes, and perceptions; uses methods like ethnography and contextual interviews
  - **Descriptive user research**: Aims to uncover how something works and describe a phenomenon in detail; uses methods such as interviews, contextual interviews, and diary studies
  - **Evaluative user research**: Aims to find out how something compares to a known set of criteria; usability studies and A/B testing are common evaluative research methods

Market Research

- Market research involves gathering data about what people want and analyzing that data to help make decisions—for example, about strategies, processes, operations, and growth
- Market research is usually split into four areas
  - **Exploratory market research**: Used when the research problem has a lot of unknowns. It identifies avenues for new and existing product growth. Market exploration usually makes use of secondary data from inside and outside the company, as well as observational studies, expert opinions, and user feedback.
  - **Descriptive market research**: Concerned with finding out how things occur, how often, and how they're connected. Interviews and surveys are popular descriptive market research methods.
  - **Causal market research**: Establishes the cause-and-effect relationship between a set of variables. It relies on statistical methods and large data sets, therefore it requires rigor.
  - **Predictive market research**: Helps you predict certain market variables. It forecasts what users will want and when they want it, and findings can affect future sales, growth projections, or the development of a product.

Product Analytics

- Product analytics is about discovering how your audience uses your product from the data trails they leave. Product analytics can be used to find answers to questions regarding the behavior of a large number of users.

Types of Product Analytics

- Product analytics can be classified into four subtypes: descriptive, diagnostic, predictive, and prescriptive.
  - **Descriptive analytics**: Describes what you know from your data, whether that's the number of downloads recorded or the percentage of users who leave the site within a minute. It paints a numerical picture of what happened.
  - **Diagnostic analytics**: Helps you discover why something is happening. It uses techniques like data discovery, drilldowns, data mining, and correlations.
  - **Predictive analytics**: Asks what might happen in the future, based on what has happened in the past. You take the data you have and employ statistical techniques, usually involving machine learning, to predict how users might behave.
  - **Prescriptive analytics**: Asks what your next steps should be, based on what you know and what you think users will do in the future. Prescriptive analytics is thus firmly rooted in predictive analytics but is more advanced.

Rules for Product Research

- Rule 1: Prepare to be wrong.
- Rule 2: Everyone is biased, including you.
- Rule 3: Good insights start with a question.
- Rule 4: Plans make research work.
- Rule 5: Interviews are a foundational skill.
- Rule 6: Sometimes a conversation is not enough.
- Rule 7: The team that analyzes together thrives together.
- Rule 8: Insights are best shared.
- Rule 9: Good research habits make great products.

Rule 1: Prepare to Be Wrong

Case Study

- Some years ago, the online marketing company Constant Contact experienced an increase in call volume from its customers, mostly small businesses. More and more callers wanted answers to their marketing questions, and data showed an increase in visits to its support pages and forums from mobile devices. This led the VP of customer success to believe that Constant Contact customers weren't getting the answers they wanted.
- Do you agree with the VP's point of view?
  - If yes, why?
  - If no, why not?

Continuing With the Story

- Some years ago, the online marketing company Constant Contact experienced an increase in call volume from its customers, mostly small businesses. More and more callers wanted answers to their marketing questions, and data showed an increase in visits to its support pages and forums from mobile devices. This led the VP of customer success to believe that Constant Contact customers weren't getting the answers they wanted.
- An executive had the idea to package all the company's content into a mobile app called Marketing Smarts. This executive allocated more than $200,000 to build this app and send it to the app stores. Fortunately, it never became a reality.
- We say "fortunately" because the mindset in the company at the time was "go build it for scale," which meant this project went to the newly formed innovation team, the Small Business Innovation Loft.
- The innovation team decided to run a design sprint to test concepts with customers and observe their reactions. They found that customers were often using a desktop when they had a question, so they didn't need a mobile app at all.

Ego Is the Enemy of Product Research

- Challenging this egocentric mindset is a key part of doing product research well
- Each time we disprove a theory or belief, we in fact discover more validation that we can be right, because we know the process we are undertaking is real.
- Instead of expecting to be right, we need to expect to be wrong, even seeking to be wrong.
- The whole team needs to adopt the right mindset for product research.

Different Mindsets in Research

- Transactional Mindset
- Confirmatory Mindset
- Problem-Finding Mindset
- The Right Mindset, the Insight-Making Mindset

Transactional Mindset: "How Can I Sell This?"

The *transactional* mindset is all about the transaction and limits itself to whether or not a customer would buy a product. An example is asking, "Would you like to buy...?" or "What if you had...?" If this sounds more like a sales pitch than research to you, you are correct. The transactional mindset doesn't take into account the nuances of a customer journey or the complex needs of the customer. It explores the topic at a surface level, avoiding the depths at which the researcher may find evidence that they are wrong. This approach is often seen in market research that solely focuses on sales performance, and it is not useful.

Confirmatory Mindset: "Am I Not Right?"

The *confirmatory* mindset is where you try to get the answers you want. If you beat up the data, it will tell you what you want to hear. The problem with this approach is that it's about confirming the ideas or beliefs you already have rather than listening to what the customer has to say. If you are in the confirmatory mindset, you might find yourself asking subtly leading questions, usually focused around the feature you are working on. These questions are laced with a secret desire to be liked by the customers, and their wording reflects your own product development world, not the customers' world.

Problem-Finding Mindset: "How Can I Improve This?"

Some teams focus on finding things to fix. These teams carry out research just to find problems. They'll treat a usability study like a driving test: there are right and wrong answers, and the student either passes or fails. If you have a problem-finding mindset, you will be watching the sessions with a very keen eye on what the participant *cannot* do. This drive to find an underlying problem causes you to be aggressive in interviews, which sometimes look more like interrogations. You assume that the participants are hiding a problem, which you are there to extract from them. You believe the users will give you a great insight if you continue asking, "Why? Why? Why? Why!"

The Right Mindset, the Insight-Making Mindset: "I Want to Understand"

We talked about teams that focus on the sales motivation, teams that focus on being right, and teams that focus on mistakes and improvements. There is a fourth type of team that focuses on learning. They are aware of their assumptions and biases, they work very hard to avoid leading their users during research, and they check their egos at the door. Their sole goal is to learn from their users. They seek to learn about users' good and bad experiences, their favorable and critical thoughts, their suggestions and complaints. When they hear an unpleasant surprise, they stop and listen, giving users space to express themselves in their own terms. These are the teams that assume the insight-making mindset. They are the ones that are most successful with product research.

The insight-making mindset is where you are open to being wrong.

The insight-making mindset adopts a diagnostic approach.

The *insight-making* mindset focuses on the positive *and* negative aspects of your product. Researchers in this mindset work with an open mind, trying their best to withhold judgment and focus on the research question at hand. Unlike researchers with a transactional mindset, they are not just interested in that "switch" moment when the purchase decision is made. Unlike researchers with a confirmatory mindset, they do not lead their participants to affirm their product's features. Unlike researchers with a problem-finding mindset, they are not looking to fix an immediate problem. They just want to listen and understand without bias.

Steps for Good Insights

Step 1: Focus on a research question.
The goal of product research is finding actionable insights for product development in a timely manner. Therefore, it is very important to start with a research question. A research question is a single focused question that frames your research endeavor: what is it that you want to learn? Finding one is not hard, and it guarantees quality and focus throughout the process.

Step 2: Identify your research method and participants.
This is the step where you decide on your method and who you will work with. There are hundreds of methods that you can use to answer your research question. But each method answers a different type of research question.

Step 3: Collect data.
It may come as a surprise that collecting data comes at such a late stage in the process! Depending on your research question and your selected method, this step can entail talking to participants, asking them specific questions, doing work together, watching them use your product, or analyzing historic data from users.

Step 4: Analyze as a team.
Your data will gain meaning if you analyze it from different perspectives. The best way to do this is as a team—not necessarily your immediate team but a group of people who offer different, even opposing, viewpoints. This is almost seeking to be wrong about your initial ideas! Involving fellow team members, peers on other teams, stakeholders, and sponsors will help you arrive at richer insights in a very short time.

Step 5: Share findings.
Sharing your research findings is just as important as the previous four steps, and it is a separate step because of the effort involved. It is so sad to see teams do a very good job with the first four steps and then write a report that no one will ever read. Sharing your findings is an opportunity to share stories, judge the business impact, and show what you suggest through prototypes. Successful product research teams don't do this just once; they do it over and over to have meaningful discussions with all affected parties.

Step 6: Plan the next cycle.
No, you're not done yet! Good product research is about continuously learning from the market and from your customers. One research endeavor leads to another, and you never stop discovering new insights.[4]

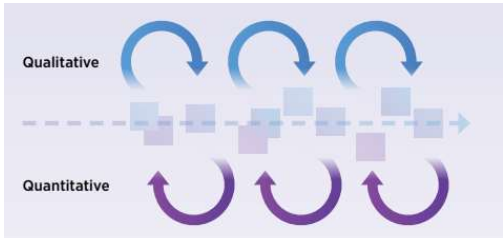
Continuous Learning in Real Life: Mix of Qualitative and Quantitative Approaches

Example 1: Start with quantitative data

Quant:	Start with sales data. Look at won/lost numbers and reasons for these. Focus on the top reason sales are lost.
Qual:	*Interview lost prospects and/or look-alikes.*
Quant:	Based on insights around won/lost reasons and interviews, examine product analytics to compare what you heard with what actually happened.
Qual:	*Prototype a product feature that solves an identified problem and get customer feedback.*
Quant:	After feature release, track analytics to check whether you are seeing the expected behavior.

Example 2: Start with qualitative data

Qual:	*On-site customer visits/day-in-the-life*
Quant:	Analytics on those customers' product usage
Qual:	*Video interviews of customers*
Quant:	Market analysis of new opportunity
Qual:	*Prototype of new product area*



Summary

- The foundation of product research is being open to being wrong. The insight-making mindset gives you the space to learn and to arrive at genuine insights.
- Good product research consists of six steps: focusing on a research question, identifying your research method and participants, collecting data, analyzing as a team, sharing findings, and planning the next cycle.
- Planning the next cycle in product research is a key behavior. Without it, you risk making research a one-time, disposable showpiece.
- Product research is an ongoing endeavor where you cycle between different types of research approaches.

Thank you

Contents

- Rule 2: Everyone Is Biased, Including You
  - What Are Biases?
  - Assumptions: What Do You Think You Know?
  - Rules in the Real World: SME Interviews
  - Summary

In 2016, product leader Hope Gurion (now of Fearless Product) took a job at Beachbody, a multilevel marketing company with a network of over 250,000 fitness coaches. The company was preparing for its annual coaching summit, which would bring 50,000 people together in a stadium in Nashville, Tennessee. It would be a great opportunity for the product team to interact with coaches. Hope and her team were responsible for managing an app called Coach Office, which the coaches used to organize their administrative work.

Up until then, Beachbody's research on this app had focused solely on its top coaches: a high-performing 1%, or about 2,500 people, who had been with the company for a long time and were expert users. They had learned their way around the product and had no issues navigating around it and getting things done. This led the internal Beachbody stakeholders, who had come to know the top coaches over the years, to believe that everything was fine with the app.

Was everything fine?

In 2016, product leader Hope Gurion (now of Fearless Product) took a job at Beachbody, a multilevel marketing company with a network of over 250,000 fitness coaches. The company was preparing for its annual coaching summit, which would bring 50,000 people together in a stadium in Nashville, Tennessee. It would be a great opportunity for the product team to interact with coaches. Hope and her team were responsible for managing an app called Coach Office, which the coaches used to organize their administrative work.

Up until then, Beachbody's research on this app had focused solely on its top coaches: a high-performing 1%, or about 2,500 people, who had been with the company for a long time and were expert users. They had learned their way around the product and had no issues navigating around it and getting things done. This led the internal Beachbody stakeholders, who had come to know the top coaches over the years, to believe that everything was fine with the app.

Was everything fine?

In 2016, product leader Hope Gurion (now of Fearless Product) took a job at Beachbody, a multilevel marketing company with a network of over 250,000 fitness coaches. The company was preparing for its annual coaching summit, which would bring 50,000 people together in a stadium in Nashville, Tennessee. It would be a great opportunity for the product team to interact with coaches. Hope and her team were responsible for managing an app called Coach Office, which the coaches used to organize their administrative work.

Up until then, Beachbody's research on this app had focused solely on its top coaches: a high-performing 1%, or about 2,500 people, who had been with the company for a long time and were expert users. They had learned their way around the product and had no issues navigating around it and getting things done. This led the internal Beachbody stakeholders, who had come to know the top coaches over the years, to believe that everything was fine with the app. But what about the other 247,500 coaches—were they using the app well?

Further, the app, which had been developed by a third party, had not been built with smartphones in mind. (Remember, it was 2016!) To make matters worse, it wasn't instrumented for usage analytics, either. The internal team had no data to help them understand app usage. So all they had to go on were the experiences and comments of these top coaches. They were biased toward one small segment of their users—and they didn't even know they were biased.

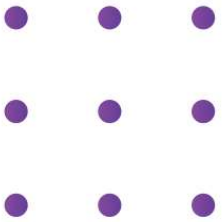
What are Biases?

- Biases are shortcuts our brains take to make things easy for us to process, allowing us to draw conclusions and make decisions faster.
- Biases can be healthy.
  - For example, if you have a bias toward eating fruit over cookies and cake, your overall health likely benefits.
- However, biases have an underlying prejudice that can be harmful.
  - They induce us to gravitate toward what we want to be researching instead of what we should be researching.
- Biases also oversimplify the phenomena you are trying to understand and may lead you to limited or downright wrong insights.
- Understanding the different types of biases and learning to identify them before you begin analysis will help you reduce or eliminate their effects on your research.

Categories of Biases

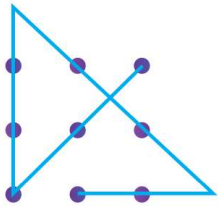
- The NeuroLeadership Institute (https://neuroleadership.com) has identified 150+ different types of bias
- The institute has also organized biases into the following categories
  - Similarity: "People like me are better than others."
  - Expedience: "If it feels right it must be true."
  - Experience: "My perceptions are accurate."
  - Distance: "Closer is better than distant."
  - Safety: "Bad is stronger than good.
- These high-level categories can help frame how biases affect you and your product research efforts.
- It is almost certain that you will exhibit biases during your product research initiatives.
- Biases can be conscious, such as in the fruit example previously given, or they can be unconscious.
- They can take the form of assumptions you don't even realize you're making.

A Fun Exercise



Connect the dots by drawing four or fewer straight lines, without lifting your pen

A Fun Exercise: Solution ☺



To solve the puzzle, you have to go quite literally "out of the box" by drawing a line that goes far outside the assumed box around the 3 × 3 grid. You might have assumed that you weren't supposed to break that invisible box, but there's no real boundary there—it's all in your head! If you had a really giant Sharpie, you might be able to connect all of the dots in one short, thick line. Were you making assumptions about the width of the pen? Maybe you considered it, but our assumption (see what we did there?) is that you didn't. This means you were making assumptions you didn't even know you were making. You likely assumed that the pen was a common ballpoint pen, which would not be incorrect, but that assumption closes the door on another insight before you even start.

Let's apply this to a real-world example. In early 2001, a company known for breakthrough inventions (such as the first drug-infusion pump and an all-terrain wheelchair) told the press that it would soon unveil a revolutionary new vehicle that would "be to the car what the car was to the horse and buggy."[3] Steve Jobs said it would be as important as the invention of the personal computer.[4] It drew $38 million from prestigious Silicon Valley investors. The project, code named "Ginger,"[5] would be revealed live on *Good Morning America*. What would it be? A hover board? A flying car? A teleportation device?

It was the Segway scooter.

The Segway was a strange-looking battery-powered scooter with no seat that had a range of about 15 miles (shown in Figure 2-3). It weighed about 70 pounds (32 kilograms) and cost about five thousand dollars. Not only was it not a threat to the automotive industry, but the average consumer was simply not interested. The company expected to sell 100,000 Segways in the first year. Nearly 20 years later, in 2020, approximately 130,000 have been sold. *Time* eventually named the Segway one of the 50 worst inventions.[6]

Let's apply this to a real-world example. In early 2001, a company known for breakt
an all-terr
olutionary
the horse
invention
tigious Sil
would be
hover boa
It was the
The Segwa
that had a
about 70
Not only
age consu
sell 100,0

pump and
veil a rev-
ar was to
ant as the
rom pres-
Ginger,"[5]
d it be? A
th no seat
weighed
d dollars.
the aver-
pected to
in 2020,
approximately 130,000 have been sold. *Time* eventually named the Segway one of the 50 worst inventions.[6]

Make no mistake, the Segway was an amazing piece of hardware, but the team made assumptions about its market acceptance that had little grounding in how the market would receive it. This vehicle fit into none of the standard categories. Was it a motorcycle? A bicycle? And what if the motor vehicle bureau required a license to operate a Segway?

From a bias standpoint they exhibited a bias to technology development. It was more interesting to the Segway team to solve the problem with technology, and that caused them to be blind to broader social context. For example, cars take up a lot of space, yet making a smaller vehicle isn't necessarily the solution. They might have taken this vehicle to the local grocery store, but would consumers? It turns out, consumers weren't interested.

To help you identify biases, we have divided them into three categories: biases that occur on the researcher side, external biases (those that stem from the participant or the data they generate), and biases that involve both.

innovate achieve lead

Researcher Bias

- Observer expectancy bias
- Confirmation bias
- Attribution errors
- Group attribution effect

Observer Expectancy Bias

Have you ever started something with an expectation about how it will end? *Observer expectancy bias* is just that. It's the tendency of a researcher to see what they expect to see in a study. (It falls into the "expedience" category of bias types). This might be because you have prior knowledge of the group of participants, an expectation about how they'll behave, or subjective feelings about the people you are studying. If you don't check your observer expectancy, you risk contaminating your research with "data" that isn't there. This can happen because you unintentionally influence participants during your sessions or cherry-pick the results that confirm your hypothesis. Are you assuming that one user segment is more intelligent than another? Is your verbal or nonverbal language influencing how they behave?

Confirmation Bias

Sometimes you can subconsciously gravitate to data that confirms what you already think (see the "experience" category of bias types). This is *confirmation bias*. If you have a hypothesis you are particularly close to or you think you know what the problem is already, you might be drawn to data that confirms those beliefs. This might even mean that you subconsciously discard or discredit data that challenges your beliefs or proves them wrong. Even if you don't think you are doing this, it's a very common unconscious bias that's hard to eliminate from research.

This bias goes hand in hand with the observer expectancy bias. Observer expectancy bias can prevent you from asking the right questions and listening openly; confirmation bias can prevent you from doing analysis with an open mind.

Attribution Errors

Another way you introduce bias into research is through *attribution errors*. This is where you attribute certain behaviors to participants' characteristics or situational circumstances unwarrantedly, and often erroneously. Human beings tend to associate negative and undesired behavior with a personal characteristic of the other person, not themselves. For example, you might think that a driver who didn't yield to you in traffic is selfish and rude, when in fact they may be in a hurry for a good reason. These errors happen very frequently when analyzing product usage data. For example, if a meal-plan dieting app has a low user retention rate, you might attribute that to users being unmotivated or unwilling to lose weight, but the real reason may be that the recommended ingredients are hard to source and the recipes are confusing.

When you are trying to understand how and why users behave, try to make sure you're considering both their personal characteristics and their circumstances. This will help you avoid attribution errors and arrive at more broadly applicable insights.

Group Attribution Effect

Another kind of attribution error is *group attribution effect*. This is when you associate your participant with a group, then assume that the group is homogeneous and that the participant has all the attributes of that group. A common example is generalizing about nationalities: even though France has a great culture of gastronomy, not every French person is a good cook. Another example: just because someone works out six days a week, you can't assume that they eat protein powder and take supplements. In fact, we see group attribution effect every time we experience or witness racism and similar types of bigotry.

Group attribution effect can arise when you are trying to build rapport with participants and make them feel like you know them. If you make wrong assumptions, you risk damaging the connection you worked hard to create.

Types of External Bias

- Availability bias
- Biased participant: The know-it-all
- Biased participant: The existing customer

Availability Bias

Availability bias (which falls into the "distance" category of bias types) is when you focus on the data or participants that are fastest and easiest to obtain. It tends to enter after you've identified a research question and planned your project, as you're selecting your participants, data, and methods. You are particularly likely to do this in areas where your product is already performing well, and you may even have a confirmatory mindset about the problem you're trying to solve. Asking existing customers how they feel about a particular feature may be easy, but it's unlikely to elicit feedback that will lead to growth. You should resist the temptation to use only the data that you can gather or speak only to people you can easily get to and be open about new things you can learn with an insight-making mindset.

Biased Participant: The Know-it-all

You have probably spoken to a customer like this. They have an answer to everything, and they are overly eager to tell you what you should do with your product. Those who shout loudest are often heard first, but that doesn't mean those voices are the only ones you should be listening to. This is not to say you should ignore them—they may have a valuable insight! But don't view them as representing your whole customer base, either. The more a customer knows about the product (the more expert they are), the more likely they are to provide overly complicated feedback. While this can be valuable, such customers often represent a tiny segment of the customer base, not the mainstream.

Listening to expert buyers at the expense of the average customer can result in narrow messaging that then results in an overengineered product. Expert buyers are often early adopters, so it's easy to fall into this trap when you are starting to build your product research practice, when those customers don't represent your target users.

Biased Participant: The Existing Customer

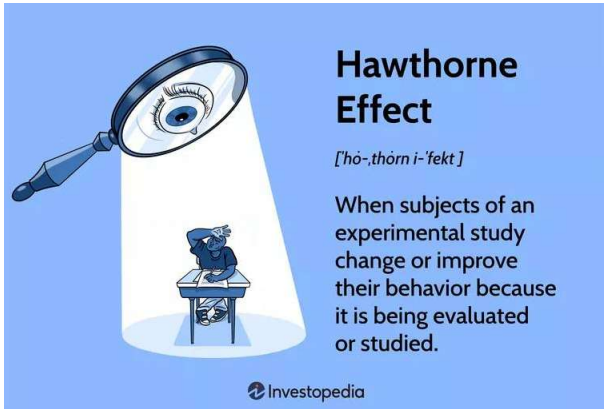
Attracting new customers and retaining existing ones can look very different. Your existing customers might want something as straightforward as simpler navigation, whereas new customers need to be drawn in by a unique design element. In product research, it's easy to lean too heavily on making improvements for your existing customers because, frankly, that's where the dollars are. But when focusing on growth, remember that not losing one existing customer isn't the same as winning an entire market.

General Biases

- Hawthorne effect (observer bias)
- Social desirability bias
- Recall biases

Hawthorne Effect

Between 1924 and 1932, researchers studied the Hawthorne Works, a factory that produced electrical equipment in Illinois, to determine the effect of certain working conditions on productivity. They split the workers into two groups: a control group that worked in the same lighting and another group that worked under brighter lights. When they increased the lighting intensity, workers' productivity increased. But what was surprising was that worker productivity increased in all groups involved in the study— even in the control group that had no increase in brightness at all.



https://www.investopedia.com/terms/h/hawthorne-effect.asp

Social Desirability Bias

Your presence affects participants' task performance; does it affect other aspects of your study? *Social desirability bias* is the tendency for participants to give responses they feel would be acceptable for the general population. Participants may avoid answering questions truthfully for fear of being judged, or they may inhibit their usual behavior because they think that it is not socially acceptable. For example, factory personnel may exaggerate how important safety is to them, even while ignoring safety precautions. Novice users may deliberately navigate to advanced features in your app to hide the fact that they may not be competent with computers.

Recall Biases

There are four common *recall biases* that affect what we remember. First, there's the *primacy* or *recency effect*: people tend to remember the first and last things we hear better than the rest of a conversation. Second is the *anchoring effect*: we tend to give more significance to the first thing we hear and use that as a reference point to evaluate everything else after it. Third, the *Von Restorff effect* holds that we tend to better recall those things that stand out from the rest. Finally, there's the *peak-end rule*: people tend to recall the end and the most unusual parts of past episodes.

Recall biases mean that your participants will not be able to share with you entire episodes of their experiences accurately because they may forget parts of the story and their memory may fill in the gaps incorrectly. What's more, you will not be able to recall everything that your participants share with you. Taking notes helps, but you may still be inclined to treat the first interesting thing you hear as a reference point for everything else, due to the anchoring effect. Or a particularly striking detail in an anecdote may surprise you so much that you miss other important details, due to the Von Restorff effect.

What Can You Do About These Biases?

- Take a good look in the mirror
  - Try a healthy dose of self-critique
  - Challenge your own motives, thoughts, and hypotheses as you plan your research, conduct it, and analyze it
- Find an independent set of eyes
  - When someone looks at a problem with a fresh set of eyes, they can often spot the bias that closer observers missed and call it out
- Be on the lookout for bias
  - During your sessions, capture the moments when you feel you or your research partners might be biased; recording these moments and discussing them afterward will improve your awareness.
- Watch your conversation style
  - How are you communicating?
  - What form do your questions take?
  - Could you be leading your participants with your language and phrasing?
  - How you communicate with your participants is key to how they perceive you.

Exercise: Assumption Storming ☺

- Look at any product
- Let's say it's a chair you're sitting on
- Ask yourself the following questions and write down the answers:
  - Why does the chair have armrests?
  - What assumptions does that imply?
  - The person has two arms?
  - If it had only one armrest, what assumptions might that be due to?
  - Or if there are wheels, is the floor it is placed on smooth enough to roll on?
  - What about the dimensions? Is the chair intended for an "average" adult human?
  - And who might that even fit?
- Craig Launcher, a designer at Medtronic, calls this process *assumption storming*.
- In the medical device industry, if you are wrong about an assumption, someone's life could be at risk.
  - While designing and developing products, his team spends days assumption storming about a problem area so that they have a fuller mental grasp of the situation.

Common Reasons for Making Assumptions

REASON	EXAMPLE
You lack knowledge.	You don't know why customers abandon a shopping cart, so you assume it's because that's when they see the total price.
You want to simplify the problem.	You assume that all customers are using your software on the latest iPhone device.
You want to standardize the problem.	You assume that what you need to do is just like something you did in a past project, and thus the solutions (or framework or standards) you used then will work here, too.
You want to make a general statement rather than a specific one.	You assume that left-handed customers are no different from right-handed customers.
Different tools encourage different assumptions.	During sketching, you might think more abstractly, leading you to make assumptions about your general approach as well as visuals. Flowcharts are very process oriented and discrete, so your assumptions will be about decision making. Design mockups are more detailed and interface focused, so your assumptions might deal more with the visual aspects of the project. Different ways of thinking lead you to focus differently.
You're responding to cultural pressure.	You make assumptions based on the latest trend. Remember Skeuomorphism? (Good for you if you don't!) It was an aesthetic trend Apple used in many of its mid-2007 designs that made things look like they did in the real world.

Common Reasons for Making Assumptions contd.

REASON	EXAMPLE
You fall into the trap of expert arrogance.	"I'm not making assumptions!" you assume.
Your project requirements are ambiguous.	You don't know which customer you're designing for, and your requirements don't mention it, so you assume that you don't need to think about accessibility for hearing-impaired or visually impaired customers.
You follow rules, norms, and conventions.	You learn the UX rule that "every additional step in a flow increases the drop-off rate" and assume that it's true in your case.
You've already formed expectations.	You expect a particular outcome and then see that outcome in the data because you're looking for it (confirmation bias).
You want to break away from routine.	You feel the urge to do something differently, making an assumption that it must be different even though the requirements don't state that at all.
You assume the normal in everyday activity.	You assume that the air in a building you enter has the right mix of oxygen, carbon dioxide, and nitrogen to support human breathing, without thinking much about it.

Rules in the Real World

SME Interviews

One way to reduce bias is to bring in experts. Yes, experts can be biased, but they also bring intimate knowledge of a topic. The trade-off can be worthwhile. The product development practitioners at BCG take a somewhat different approach to product development than a typical SaaS company does. Boston Consulting Group (BCG) is a large company that employs experts in many fields, so, researchers reasoned, why not use them? Iuliia Artemenko, a product manager in BCG's practice, says that when BCG has an opportunity to build a product, while performing their initial market research they also interview their internal SMEs. What better way to inform a product team than to have an SME with deep industry knowledge help frame the problem? Only then does the team begin work on a prototype to test with users. This way, they level set the initial product direction with real users and incorporate their feedback into their product development efforts, since expert users can be biased, as we've seen. This real-world check helps BCG reduce any internal SME bias when they launch a new product or features.

Summary

- If you are a human being, you are biased.
  - Biases can be conscious prejudices or unconscious assumptions and can sway the results of your research.
- Avoid directing your research toward a particular set of users just because they are easily accessible, expert, or loyal.
- It is fine to make assumptions
  - Be clear on what you are assuming in your research and why.
- Analyze your assumptions.
  - There is a difference between what you think you know and what you actually know.

Thank you

Contents

- Rule 3: Good Insights Start with a Question
  - What's an Insight?
  - It's Too Easy to Start Research Without a Question
  - Going from Hunch to Research Question
  - The Usage Perspective
  - The Business Perspective
  - The Expertise Perspective
  - Rules in the Real World
  - Summary

The Meaning of "Research"

Daniel Elizalde had just started his new job as vice president and head of Internet of Things (IoT) at the telecommunications company Ericsson. His charge was to build and deliver end-to-end IoT solutions to market. He had access to market intelligence from various sources (McKinsey & Company, IDC, Gartner, and so forth), and his initial direction, following the market reports, was to support manufacturing via "Industry 4.0," which brings automation and digitization to traditional manufacturing practices. Daniel had to do some product research because he had a number of questions that needed to be answered.

The word *research* meant something very different at Ericsson than it did to Daniel, who spent his career in product management. At Ericsson, *research* means technology development: for example, while the company is bringing 5G products to market at the time of this writing, they have research teams conducting technology development on the next generation of network technology (you might call it "6G").[1] Contrast *technology* research with *product* research, which seeks to answer questions around viability, usability, and desirability of a product.

What's an Insight?

- An *insight* is a nugget of information you discover that makes you look at a situation from a different perspective. It's an observation about the behavior and psychology of a group of users. In short, it's like learning the secret to something.

Let's take this to a real product example: the MachineMetrics (MM) operator dashboard is a tablet that's mounted next to a machine in a factory. The screen shows data and information to the machine operator on how the machine is performing regarding the current job. If the machine is on track, the screen is green. If it is behind, the screen is orange or red depending on how far behind. The initial intent of this design choice was to enable factory workers to see from a distance how the machines were performing. If you've been inside a factory, they can be quite large spaces, and this visibility from a distance is very helpful. The insight that the MM team uncovered after some product research was that the machine operators were strongly affected emotionally by the red screen color and it would demotivate them, so they would not work hard to get back on track. Digging further, the inverse was true: if the screen remained green, the operators would work harder to keep it green! The team took this information into account when redesigning the interface. The question the MM team began with was: how do the machine operators consume the information and data from the tablet during their workday?

It's Too Easy to Start Research Without a Question

- It is surprisingly easy to start research without a research question.
- Teams that are new to product research often fall into this trap.
- In their excitement to find out how their product is being received, they dive into looking at data, asking users, and showing concepts—without a focus.
- We see three common traps here: starting with no focus at all, starting with an output instead of a question, and starting with a method instead of a question.
  - The Vagueness Trap: "Let's Do a General Check!"
  - The Output Trap: "We Need Personas"
  - The Method Trap: "Should We Do a Survey?"

The Vagueness Trap

- Teams that are just beginning with research may feel that they need to take a broad look at everything first.
- So they attempt a "general check" with their users, bombarding users with questions.
- The result is lots of information but maybe only a few genuine insights that would help them with their product.
- By asking too much, they risk gathering data that can't possibly be distilled into actionable results quickly—which is what product research is all about.
- Being excited to know about every aspect of your customers' experience is great; it shows that you care!
- Taking focused, iterative steps toward this goal is a better product research practice.

The Output Trap

- Teams who are new to research can fall into the trap of focusing on the output.
- It's easy to read a Medium article about user personas and think, "OMG, that's what we need!"
- It is better to take a few steps back and think about *why* you think you need personas (or a customer journey map, or another shiny output).
- Are you asking this question because of a recent change in your plans? Are you interested in a particular type of behavior, for a particular type of development? Are you trying to explore a particular area for new business opportunities?
- Focusing on the problem before the output yields better insights.

The Method Trap

- Similar to the temptation to start with an output, you might be inclined to start with a method instead of a research question.
  - You might say, "We should do a survey—why not? All customer-centric companies do surveys!"
- Then you'd try to figure out what questions you want to ask.
- This is research planning done backward.
- Teams who get good results from product research first decide on what they want to learn about, then decide on the method to use.

The Value of a Research Question

- It's essential to narrow your objectives to a single crisp research question.
- A research question provides focus to your research effort and ensures that it has an impact.
- You'll use a combination of product data, comparisons, market opportunity, and known best practices to frame the problem and create a single, pinpointed research question.
- A question that is well formulated for the purpose of the project helps everyone focus on the common problem, without getting distracted by the surrounding context.
- How can you distill everything you want to know into a single research question?

Going from Hunch to Research Question

- Your research may start with a hunch—something that bugs you about your product.
- For successful product research, you need to go beyond assumptions or "what if?" musings and find a research question.
- A research question is a single, focused question that guides your research.
- You can arrive at it iteratively by examining your hunch through different lenses to find the underlying problems, then formulating one research question to learn about those areas of interest.
- Sometimes your refinement leads to very interesting problems and multiple research questions.

From Hunch to Problem

- Who?
  - What type of person are you trying to learn about? Whatever your hunch is about, is it a problem for that person? How do you know? Do they think that it is a problem?
- What?
  - What is the nature of your hunch? What is driving you to explore this area? What evidence do you currently have? What information do you lack?
- Why?
  - Why is this worth exploring? What is its impact on the user? How significant is it? Why do you care about this now?
- Where?
  - Where do you see this issue occur? What is its natural habitat, and what is the broader context?
- When?
  - When does this happen? With what frequency? Are there exceptions to this frequency? Does the user's experience change as they use the product?
- How?
  - How did you arrive at this hunch? Does it manifest itself as a problem or a moment of delight for the user? Do they experience it differently in different channels?

Different Perspectives

- The usage perspective
  - When you find out what your users are doing with your product, you'll have a better understanding of the issues and opportunities in play. You will start to discover the problems they're having, which means you'll be able to incorporate their behavior and sentiments into your research.
- The business perspective
  - Product management is a complex domain, and one of its goals is to sustain financial growth. Delivering great experiences can cost you, but great experiences can bring great returns. Looking at your problems from a business perspective helps you determine what is valuable for future growth.
- The expertise perspective
  - Industry leaders, academics, internal SMEs, and resources created by them can help you look deeper into your problem and focus your effort on the most valuable parts. You shouldn't make up your own usability rules, campaign structures, or market trends; you can start with what is already out there.

Hunch to Research Question

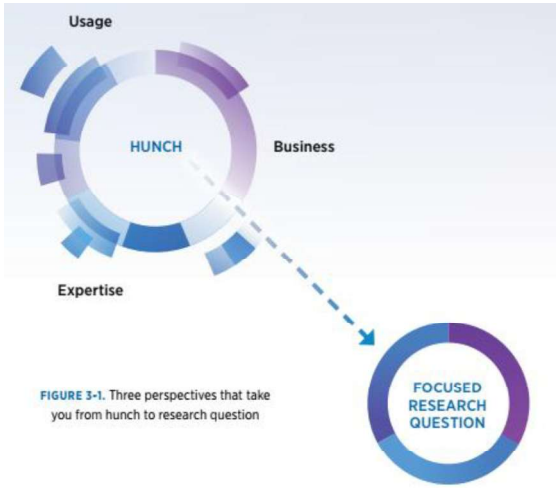


FIGURE 3-1. Three perspectives that take you from hunch to research question

Properties of a Good Research Question

- Focused and deliberate
  - A research question has a very specific focus, carefully chosen by the researcher.
  - Note that you can have a very broad research question that is also extremely focused
    - For example, "How do low income communities cope with COVID-19 risks?"
- Open-ended
  - A research question is not stated as a yes-or-no question.
  - Product research is about learning with an open mind.
  - Open-ended questions allow your participants to share experiences that you never thought of. They also allow you to ask about interesting moments as they arise.
- Free of prejudices
  - A research question is not leading; it is free of prejudices.
  - It does not come with a hidden agenda and is not designed to elicit the answers you want to hear.
  - The answers you get depend on how you frame the question.
  - That's why it is important to get rid of implicit biases while you turn your problem into a research question.

A Good Research Question

- A research question is different from an interview question.
- A good research question is informed by what you already know, not your assumptions.
- Examining your hunch from three different perspectives will help you reach a problem that is grounded in reality and worth spending time on.

The Usage Perspective

- Usage data from event tracking and feedback in the form of user voices are helpful in framing the problem so that you can form the right research question.
  - Event Tracking
    - A great place to start understanding how people use your product or service is by observing how they interact with it. One way of getting to this data is by examining the traces your users leave.
    - Segments and Cohorts
      - *Segments* are groups of users organized by certain criteria.
      - *Cohorts* are segments that are based on certain behaviors shown over a specific time period
    - User voices
      - It's important to look not only at how your users are *behaving* with your product but also at how they *feel* about it.

The Business Perspective

- Your Business Model
  - What you offer and how you offer it—in other words, your *business model*—will be a big input to your research question. Understanding your business model and how it affects the nature of your research question will allow you to pinpoint more accurately where your greatest opportunities are.
- The Market
  - Understanding the opportunities available in the market for your product is another important input in refining your problem. Here, data you already have can give you insights into where your product fits for your customers. Knowing about the market is especially important if your problem is related to new areas of growth.
- Operations
  - *Operations* refers to the important group of people who make sure that the users' digital experiences are running as intended. These people are usually the "nondigital" part of a digital product. Operations teams are responsible for functions like user support, quality of service monitoring, shipping and returns, IT, and accounting and finance, among others. Knowing about the operational background of your problem gives you a phenomenal advantage in framing your problem.

Different Markets

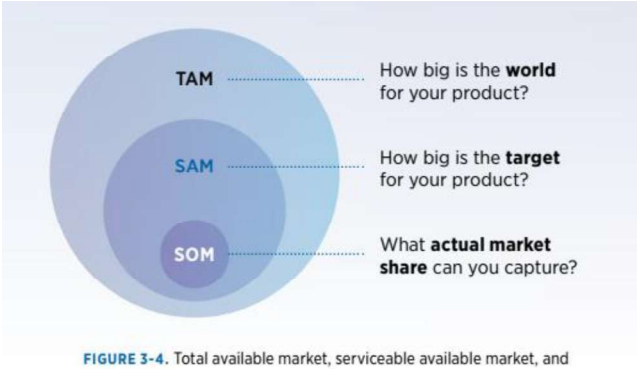


FIGURE 3-4. Total available market, serviceable available market, and serviceable obtainable market

Heuristic Analysis

- A *heuristic analysis*—often referred to as *expert review*—is a structured way of reviewing your product against known UX best practices.
- In its most common form, you can take your product (or a prototype) and ask three to five usability and/or design experts to offer their opinions on how well the design matches up to current best practices.
- Heuristic analysis will *not* be 100% accurate or complete. In fact, that's the whole point ☺
- Some typical questions for starting a heuristic analysis:
  - *How will the user attempt to achieve their intended outcome(s)?*
  - *How will the user see the correct action(s) available to them?*
  - *Will the user associate the correct action with the outcome they expect to achieve?*
  - *How will the user see progress toward their intended outcome?*

Existing Research

- One of the biggest goals of product research is to arrive at insights without having to wait a long time and put in a lot of resources.
- That is why it is critical to frame your problem in a way that makes research less challenging.
- In some cases, someone might have done it for you: the problem you are interested in, or a problem that is very similar, might have already been the subject of someone else's research.
- Therefore, looking at existing research can be a great way to focus your research question.
  - Existing internal research: In your organization, someone in a similar function might have looked at a similar problem a while ago and did some investigation.
  - Existing external research: This research may include publicly released research findings from other companies (usually in the form of a Medium article) or databases of research findings from research agencies.

Rules in the Real World

Paying tuition by bank wire can be costly, time-consuming, and opaque. It can be extra stressful for international students attending university in another country. One cross-border payment company made it their mission to make these large international money transfers a breeze.

A former product manager for the company was given a broad problem to solve once she arrived: find where the money is. (What a focused place to start!) She spoke to the support team in her first few weeks at the company to get their perspective. They told her about where in the process people were struggling, where they needed in-person help, and where they usually got confused.

She then met with the data science team to review recent transaction trends. The first quantitative insight they found was the small conversion rate: out of every one hundred transactions that started, only a handful finished. They found a second insight: successful payments tended to start on a mobile device and finish on a desktop.

Rules in the Real World contd.

This was interesting, but the data science team didn't know why. So the product manager pulled together a research sprint—a time-boxed learning and prototyping practice—to speak with international students. Luckily, the company's headquarters was near many large universities—so out of the building she and her team went! The team interviewed students about how they pay their tuition and learned about how they use the company's products in the process.

From that activity they gained two valuable insights. First, the reason for starting on mobile and finishing on desktop was that the student, usually about 17 or 18 years old, would receive the payment email from the school on their smartphone and then begin the process. Often they would get stuck because a large amount of information was needed that the student didn't have (tax ID numbers, bank numbers, and so forth). The student would then forward the payment email to their parents with the kind of loving note that all parents enjoy receiving: "Hi, Mom and Dad, can you pay this bill? Thanks! Love you!"

Rules in the Real World contd.

The second important aspect of this insight was that there were two different people involved in this process. At the time, the product's user experience didn't differentiate between student and parent users. Furthermore, parents would often go to the bank to initiate a wire transfer, thus completely cutting the company out of the transaction. This explained the low conversion rate and highlighted some serious problems with the product experience. It also indicated a significant opportunity in the higher education sector.

With these insights, the product manager took the initiative to summarize, form solutions hypotheses, and begin experimenting with her team. Their findings ultimately led to a significant increase in conversions and revenue from a segment the company had previously thought to be saturated. All it took was some brief but thoughtful and rigorous product research to "find where the money was!" She made this approach a habit at the company by doing variations of it over and over. She has since left the company, yet they still approach the product in a similar manner.

Summary

- All good research starts with a single question.
- That question should be based on what we already know—and that means data.
- Examine users' behavior data with event tracking, user voices, heuristics, and user segments and cohorts to find salient behavior.
- Frame your opportunity based on your business model and the market available to you.
- Look at the experience of people delivering the unseen parts of the service for hidden, high-impact areas.

Thank you

Start Recording

Contents

- Purpose of Product Discovery
- Core Principles of Product Discovery
- Discovery Iterations
- OKR Framework
- Discovery Techniques Overview
- Market Segmentation
- Personas
- Identify Underserved Customer Needs
- Customer Benefits
- Hierarchy of Needs
- Importance vs. Satisfaction

Purpose of Product Discovery

- The purpose of product discovery is to address these critical risks:
  - Will the customer buy this, or choose to use it? (Value risk)
  - Can the user figure out how to use it? (Usability risk)
  - Can we build it? (Feasibility risk)
  - Does this solution work for our business? (Business viability risk)
- And it's not enough that it's just the product manager's opinion on these questions. We need to collect evidence.
- When it comes to *how* we do product discovery, there are a set of core principles that drive how we work. If you understand these, you will understand not only how to work well today but also how to easily incorporate new techniques as they emerge in the future.

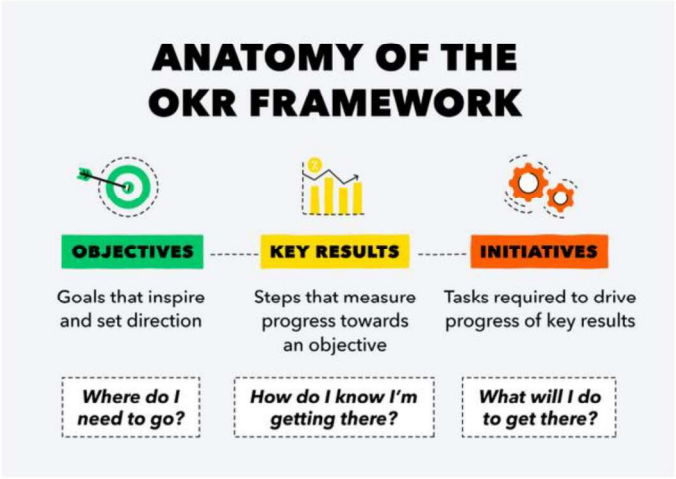
Core Principles of Product Discovery

1. We know we can't count on our customers (or our executives or stakeholders) to tell us what to build.
2. The most important thing is to establish compelling value.
3. As hard and important as the engineering is, coming up with a good user experience is usually even harder, and more critical to success.
4. Functionality, design, and technology are inherently intertwined.
5. We expect that many of our ideas won't work out, and the ones that do will require several iterations.
6. We must validate our ideas on real users and customers.
7. Our goal in discovery is to validate our ideas the fastest, cheapest way possible.
8. We need to validate the feasibility of our ideas during discovery, not after.
9. We need to validate the business viability of our ideas during discovery, not after.
10. It's about shared learning.

Discovery Iterations

- Most product teams normally think of an *iteration* as a delivery activity. If you release weekly, you think in terms of one-week iterations.
- It's true that ideas come in all shapes and sizes, and some are much riskier than others, but the purpose of discovery is to do this much faster and cheaper than we can do in delivery
- But we also have the concept of an iteration in discovery. We loosely define an *iteration* in discovery as trying out at least one new idea or approach.
- To set your expectations, teams competent in modern discovery techniques can generally test on the order of 10–20 iterations *per week*.
- Many iterations never make it beyond just you, your designer, and your tech lead.
- The very act of creating a prototype often exposes problems that cause you to change your mind.
- As a rule of thumb, an iteration in discovery should be *at least* an order of magnitude less time and effort than an iteration in delivery.

Objectives and Key Results (OKR)



https://mint.intuit.com/blog/planning-2/okr-framework/

Discovery Techniques Overview

- Discovery Framing Techniques
  - Framing techniques help us to quickly identify the underlying issues that must be tackled during product discovery.
  - If we're handed a potential solution, we need to clarify the underlying problem to be solved.
  - We need to tease out the risks and determine where it makes sense to focus our time.
  - We also need to ensure that we understand how our work fits in with the work of other teams.
- Discovery Planning Techniques
  - There are a few techniques that are useful throughout the product discovery effort and help with identifying the bigger challenges and planning how you'll attack this work.

Discovery Techniques Overview contd.

- Discovery Ideation Techniques
  - There are, of course, any number of ways to come up with ideas.
  - But some sources are better than others in their potential for keeping us focused on the most important problems.
  - Ideation techniques are designed to provide the product team with a wealth of promising solutions aimed at the problems we're focused on now.
- Discovery Prototyping Techniques
  - Our go-to tool for product discovery is typically a prototype.
- Discovery Testing Techniques
  - Testing Feasibility
  - Testing Usability
  - Testing Value
  - Testing Business Viability
- Transformation Techniques
  - Quantitative
  - Qualitative

Discovery Framing Techniques

- Much of our product discovery work doesn't require a lot of framing or planning.
- We need to come up with a solution to a particular problem, and often this is straightforward, and we can proceed directly to delivery work.
- But for many efforts, this is decidedly not the case, and some framing and true problem solving becomes critically important.
- Big projects—and, especially, *initiatives* (projects spanning multiple teams)—are common examples.
- Our discovery work to ensure alignment and to identify key risks. Two goals:
  - The first is to ensure the team is all on the same page in terms of clarity of purpose and alignment.
  - The second purpose is to identify the big risks that will need to be tackled during the discovery work.

Different Techniques for Different-Sized Efforts

1. An opportunity assessment is designed for the vast majority of product work, which ranges from a simple optimization to a feature to a medium-sized project.
2. A customer letter is designed for larger projects or initiatives that often have multiple goals and a more complicated desired outcome.
3. A startup canvas for those times you're creating an entirely new product line or a new business.

- Note that these techniques are not mutually exclusive. You may find it useful to do both an opportunity assessment and a customer letter, for example.

Problems versus Solutions

There is an underlying theme you'll see in all framing techniques, and the reason is that it's just human nature for people to think and talk in terms of *solutions* rather than the underlying *problems*. This applies especially to users and customers but also applies to stakeholders in our business, other company execs, and if we're honest with ourselves, it very often applies to us as well.

But one of the most important lessons in our industry is to *fall in love with the problem, not the solution*.

Why is this so important? Because, more often than not, our initial solutions don't solve the problem—at least not in a way that can power a successful business. It usually takes trying out several different approaches to a solution before we find one that solves the underlying problem.

However, there very likely is a legitimate problem behind that potential solution, and it's our job in the product organization to tease out the underlying problem and ensure that whatever solution we deliver solves that underlying problem.

A small amount of time up front framing the problem to be solved—and communicating this framing—can make a dramatic difference in the results.

This problem famously applies to startup founders. Founders will often stew on a potential solution for month, if not years, before they get the funding and the nerve to pursue it.

> *More often than not, our initial solutions don't solve the problem—at least not in a way that can power a successful business.*

This is another reason why typical product roadmaps are so problematic. They're lists of features and projects where each feature or project is a possible solution. Somebody believes that feature will solve the problem or it wouldn't be on the roadmap, but it's all too possible they are wrong. It's not their fault—there's just no way to know at the stage it's put on the roadmap.

Problem Space vs. Solution Space

Segment Your Target Market

- Demographic Segmentation
  - Demographics are quantifiable statistics of a group of people, such as age, gender, marital status, income, and education level. E.g. App for moms to easily share photos of their babies; demography of target customers women 20 to 40 years old who have one or more children under the age of three.
- Psychographic Segmentation
  - Psychographics are statistics that classify a group of people according to psychological variables such as attitudes, opinions, values, and interests. For the same app, you might describe your target customers as moms who enjoy using social media and like sharing pictures of their babies with friends and family.
- Behavioral Segmentation
  - You can also use relevant behavioral attributes to describe your target customer: whether or not someone takes a particular action or how frequently they do. You might define your target market as moms who currently share an average of three or more baby pictures per week on social media
- Needs-Based Segmentation
  - With this approach, you divide the market into customer segments that each have distinct needs. Let's take Dropcam, for example, which offers an affordable, easy-to-use wireless camera. A parent, may use Dropcam to monitor their children while they sleep:

PERSONAS

The persona is a useful tool for describing your target customer. Alan Cooper championed the use of personas as part of his "Goal-Directed Design" process. In his book *The Inmates are Running the Asylum*, he describes personas as "a precise definition of our user and what he wishes to accomplish." Cooper explains, "personas are not real people" but rather "hypothetical archetypes of actual users." Personas

What Info Should a Persona Provide?

Good personas convey the relevant demographic, psychographic, behavioral, and needs-based attributes of your target customer. Personas should fit on a single page and provide a snapshot of the customer archetype that's quick to digest, and usually include the following information:

- Name
- Representative photograph
- Quote that conveys what they most care about
- Job title
- Demographics
- Needs/goals
- Relevant motivations and attitudes
- Related tasks and behaviors
- Frustrations/pain points with current solution
- Level of expertise/knowledge (in the relevant domain, e.g., level of computer savvy)
- Product usage context/environment (e.g., laptop in a loud, busy office or tablet on the couch at home)
- Technology adoption life cycle segment (for your product category)
- Any other salient attributes

The Busy Mom

Lisa Bennett

Age:	32
Gender:	Female
Marital Status:	Married
Education:	Bachelor's degree
Job:	Teacher
Income:	$55,000

"My children's health is my top priority, but raising two kids is a full-time job, so I need an easy way to stay on top of their prescriptions and medical appointments."

Lisa is an elementary school teacher. She lives with her hard-working husband Dave and their two children Addison (12) and Caleb (9). Because Dave often works late, Lisa is the primary caregiver to her children.

Although her children are generally healthy, they both have to take important prescriptions. Addison has asthma and must always have her inhaler by her side. Lisa worries that she might forget to refill a prescription for the inhaler and potentially put Addison in danger.

Life never stops for Lisa, and she rarely has a moment to herself. Therefore, she needs an easy way to keep track of her children's prescriptions and medical appointments.

Goals
- Be reminded of children's medical appointments
- Be able to keep track of children's health info
- Have the ability to refill prescriptions easily

Technology Use
- Average
- Owns an iPhone
- Uses desktop PC
- Uses Facebook to keep up with family and friends

Interests
- Spending time with family
- Being involved with her children's extracurricular activities
- Tennis

Identify Underserved Customer Needs: An Example - Turbotax

Tax preparation software can go well beyond the IRS tax forms, which are just instructions for how to prepare your tax return. Tax software can check the accuracy of your return. TurboTax can also file your taxes for you electronically, which is more convenient than having to print out and mail your return. It can help you maximize your deductions and reduce your audit risk. It can even download your tax information from your employer, banks, and brokerages so that you don't have to enter it manually. Each of those items is a distinct customer benefit. Let's list them explicitly:

1. Help me prepare my tax return
2. Check the accuracy of my tax return
3. Reduce my audit risk
4. Reduce the time it takes me to enter my tax information
5. Reduce the time it takes me to file my taxes
6. Maximize my tax deductions

This is by no means an exhaustive list of the customer benefits that TurboTax provides. We could easily keep peeling the onion and identify many more benefits. For example, state tax returns are completely separate from federal returns. Also, TurboTax offers a service that lets you receive your tax refund more quickly. But for the purposes of this discussion, let's focus on the six benefits listed above.

Customer Discovery Interviews

You should share each of your customer benefit hypotheses with the customer during the interviews. You should ask a set of questions about each benefit statement, such as:

- What does this statement mean to you? (to check their understanding)
- How might this help you?
- If a product delivered this benefit, how valuable would that be to you?

(Possible responses: no value, low value, medium value, high value, or very high value)

- For a response of high or very high value: Why would this be of value to you?
- For a response of low or no value: Why wouldn't this be of value to you?

Understanding Customer Benefits

Customer Benefit	Typical Customer Comment
1. Help me prepare my tax return	"I don't really know much about taxes. I try to follow the instructions but they're confusing. I'm not sure which forms I should be filling out."
2. Check the accuracy of my tax return	"I'm not that great at math, so I know I'm probably making several mistakes when I'm adding and subtracting all those numbers on my tax forms."
3. Reduce my audit risk	"I'm worried about being audited but don't really know how risky my tax return is. It would be great to know if it would raise any yellow flags with the IRS so I could fix those parts."
4. Reduce the time it takes me to enter my tax information	"I spend lots of time each year entering data from all the tax forms I receive from my employer, bank, and brokerages."
5. Reduce the time it takes me to file my taxes	"I normally print my tax return and then go to the post office, wait in line, and mail it so I can get delivery confirmation. It would be great if I could avoid that hassle."
6. Maximize my tax deductions	"I don't know about all the deductions that I'm eligible to take. I'm probably leaving some money on the table."

Customer Benefit Ladders

Benefit at Top of Ladder	Detailed Customer Benefit
Feel confident	1. Help me prepare my tax return 2. Check the accuracy of my tax return 3. Reduce my audit risk
Save time	4. Reduce the time it takes me to enter my tax information 5. Reduce the time it takes me to file my taxes
Save money	6. Maximize my tax deductions

Summary

- Product discovery process has many facets
- Understanding customer needs plays a central role in successful product discovery
- Customer benefits can be understood from different perspectives

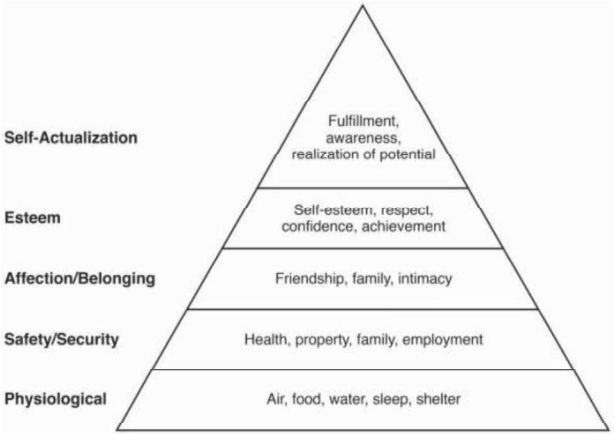
Thank you

Start Recording

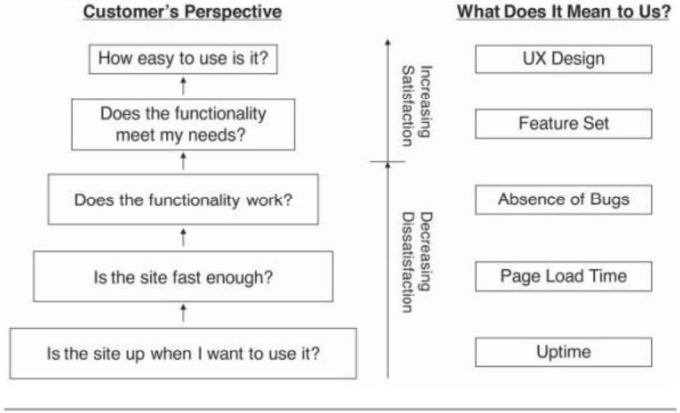
Contents

- Hierarchy of Human Needs
- Hierarchy of Web User Needs
- The Importance vs. Satisfaction Framework
- Discovery Framing Techniques
  - Opportunity Assessment Technique
  - Customer Letter Technique
  - Startup Canvas Technique
- Discovery Planning Techniques
  - Story Map Technique
  - Customer Discovery Program Technique
  - Case study: Martina Lubchenco of Microsoft [Reading]
- Discovery Ideation Techniques
  - Customer Interviews
- Summary

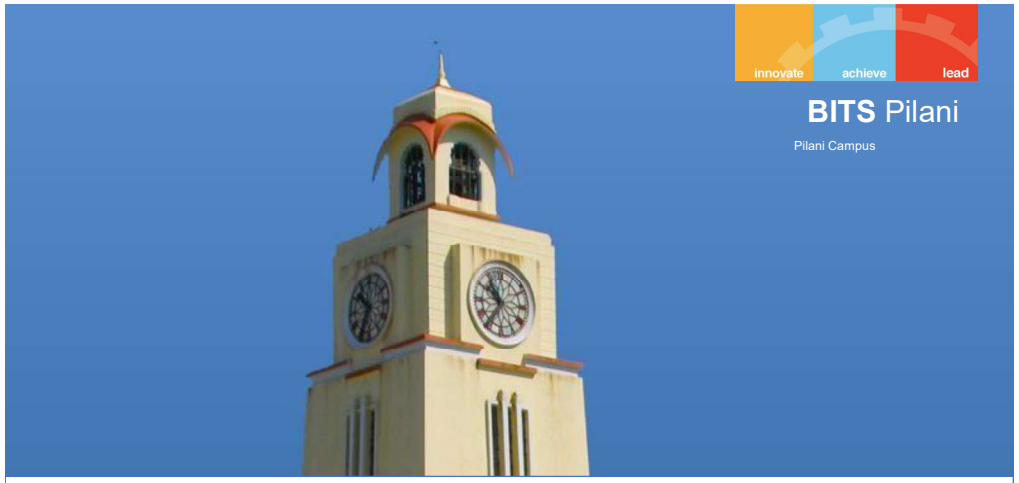
Maslow's Hierarchy of Human Needs



Olsen's Hierarchy of Web User Needs



The Importance vs. Satisfaction Framework

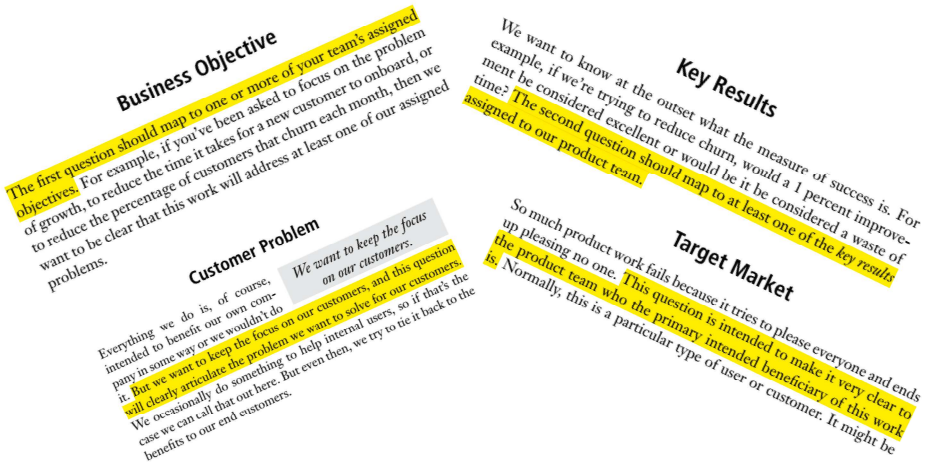


Discovery Framing Techniques

Opportunity Assessment Technique

- An opportunity assessment is an extremely simple technique but can save you a lot of time and grief.
- The idea is to answer four key questions about the discovery work you are about to undertake:
  - What business objective is this work intended to address? (*Objective*)
  - How will you know if you've succeeded? (*Key results*)
  - What problem will this solve for our customers? (*Customer problem*)
  - What type of customer are we focused on? (*Target market*)
- Answering these questions is the responsibility of the product manager, and it normally takes a few minutes to prepare these answers.
- But then the product manager needs to share them with her product team and with key stakeholders to ensure you are on the same page.

Opportunity Assessment Technique contd.



Customer Letter Technique

- For smaller and more typically sized efforts, the opportunity assessment is usually sufficient.
- But when embarking on a somewhat larger effort, there may in fact be multiple reasons, several customer problems to be solved, or business objectives to be tackled.
  - A typical example of an effort of this size would be a redesign.
  - Maybe it is intended to both improve the experience for current customers and perform better for new customers.
- To communicate the value effectively, it may take more than the four questions listed in the previous technique.
- In the format of a customer letter written from the hypothetical perspective of one of your product's well-defined user or customer personas, the following points are addressed:
  - How does the planned redesign improve the life of our customers?
  - What are the real benefits to them?
- The letter—sent to the CEO from a very happy and impressed customer—explains why he or she is so happy and grateful for the new product or redesign.
- The letter also includes an imagined congratulatory response from the CEO to the product team explaining how this has helped the business.

Startup Canvas Technique

- The techniques explored so far are for typical-sized, smaller efforts like adding a new feature, or medium to large-sized efforts like a redesign.
  - Those cover most of what product teams actually work on.
- However, another especially difficult situation requires a more comprehensive framing technique.
  - An early stage startup, where you are trying to figure out a new product that can power a new business
  - For those that work at an enterprise size company, when you're asked to tackle an all-new business opportunity for the company.
- You're not being asked to improve an existing product, you're being asked to invent an entirely new product ☺
- In such situations, a wide array of risks:
  - Validating your value proposition
  - Figuring out how you intend to make money
  - Figuring out how you plan to get this product out to your customers and sell to them
  - Deciding how much it will cost to produce and sell this product, and what you will measure to track your progress
  - Determining whether the market is large enough to sustain a business.

Startup Canvas Technique contd.

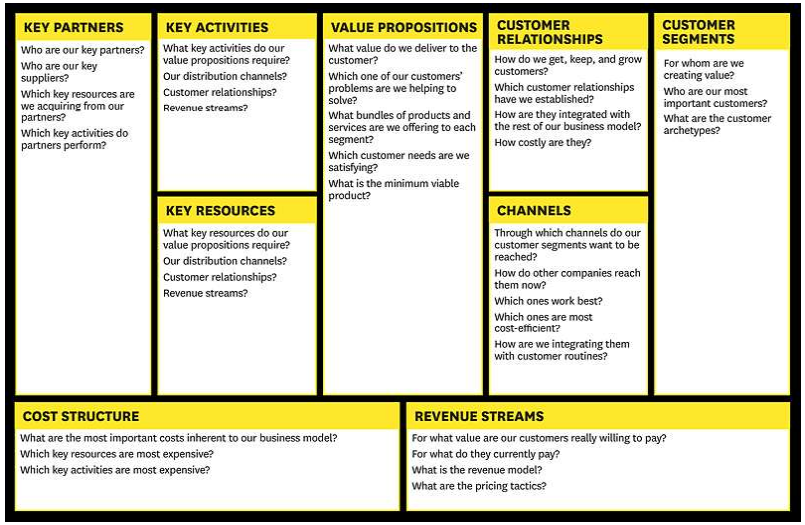
- A startup canvas, its close cousins the business model canvas, and the lean canvas are intended to be lightweight tools to call out these risks early and encourage the team to tackle them up front.
- You can use a canvas for any product change, no matter the size, but you would likely quickly find that, once you have an existing product and business, the majority of the canvas doesn't change and is only duplicated.
- The startup canvas for simpler work, especially if you have a new product manager. The startup canvas can help that new product manager get a good holistic understanding of her product and understand the key areas of the affected business.

Business Model Canvas

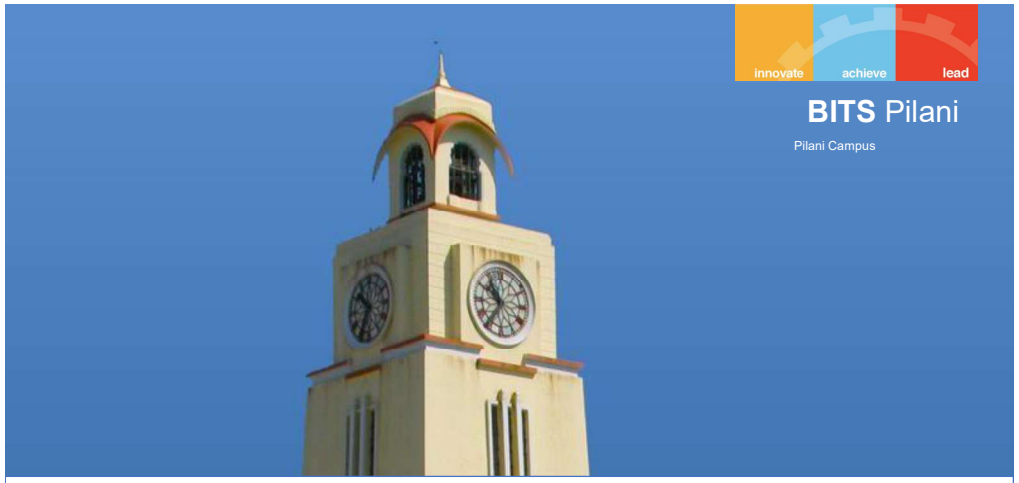
- The business model canvas — as opposed to the traditional, intricate business plan — helps organizations conduct structured, tangible, and strategic conversations around new businesses or existing ones.
- Leading global companies like GE, P&G, and Nestlé use the canvas to manage strategy or create new growth engines, while start-ups use it in their search for the right business model.
- The canvas's main objective is to help companies move beyond product-centric thinking and towards business model thinking.

https://hbr.org/2013/05/a-better-way-to-think-about-yo

Business Model Canvas contd.



https://hbr.org/2013/05/a-better-way-to-think-about-yo

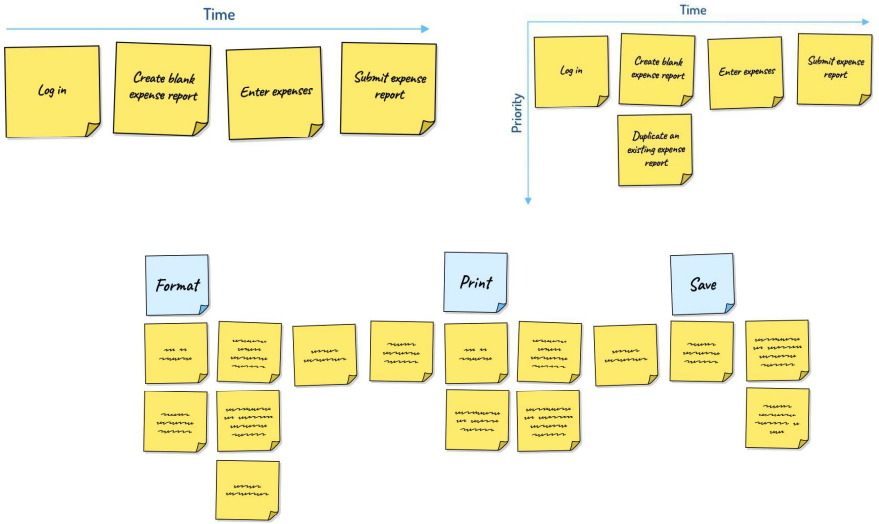


Discovery Planning Techniques

Story Map Technique

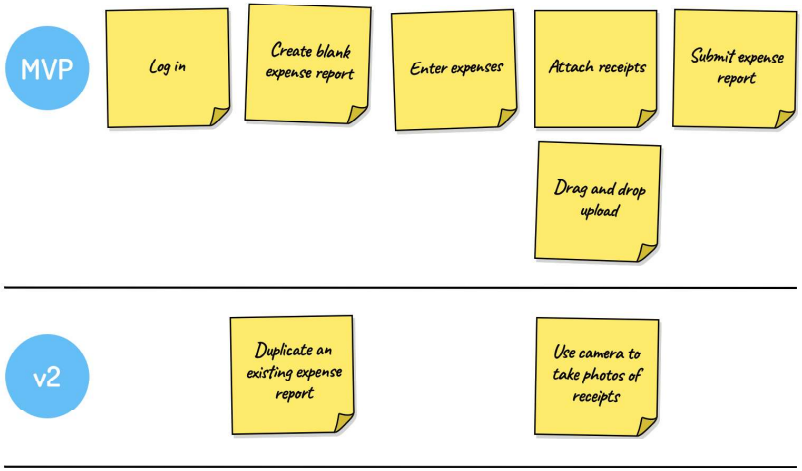
- The origin of story maps came from frustration with the typical flat backlog of user stories.
  - There's no context, just a prioritized list of stories.
  - How can the team know how one story fits in with the big picture?
  - What does it mean to even prioritize at that granularity with so little context?
  - And what set of stories constitutes a meaningful milestone or a release?
- Story maps are two-dimensional maps, in which major user activities are arrayed along the horizontal dimension, loosely ordered by time from left to right.
- Along the vertical dimension, we have a progressive level of detail. As we flesh out each major activity into sets of user tasks, we add stories for each of those tasks. The critical tasks are higher vertically than the optional tasks.
- If you lay out your system this way, you can, at a glance, get the holistic view and consider where to draw the line in terms of different releases and their associated objectives.

Story Map: Example



https://www.mountaingoatsoftware.com/blog/user-story-mapping-how-to-create-story-maps

Story Map: Example contd.



https://www.mountaingoatsoftware.com/blog/user-story-mapping-how-to-create-story-maps

Story Maps: Utility

- Note that each story has context. The entire team can see how it fits in with the other stories. And not just as a snapshot in time. The team can see how the system is expected to grow over time.
- We can use this story map to frame our prototypes, and then as we get feedback on our prototypes and learn how people interact with our product ideas, we can easily update the story map to serve as a living reflection of the prototypes.
- As we finalize our discovery work and progress into delivery, the stories from the map move right into the product backlog.

The Power of Reference Customers

First, we need to talk about the nearly magical power of a happy reference customer.

Let's be clear about what it means to be a *reference customer*: This is a *real* customer (not friends or family), who is running your product in *production* (not a trial or prototype), who has paid *real money* for the product (it wasn't given away to entice them to use it), and, most important, who is willing to *tell others* how much they love your product (voluntarily and sincerely).

There are few things more powerful to a product organization than reference customers.

Please believe me when I say that there are few things more powerful to a product organization than reference customers. It is the single best sales tool you can provide to your sales and marketing organization, and it completely changes the dynamics between the product organization and the rest of the company.

Ask any good salesperson the single best tool you can provide to help her do her job, and she'll say, "happy reference customers."

Customer Discovery Program Technique

We are discovering and developing a set of reference customers in parallel with discovering and developing the actual product.

I will warn you that this technique takes substantial effort, primarily on the part of the product manager. I wish it were easier. But I will also say that if you do this technique, I consider it the *single best leading indicator of future product success.*

I will also say that this technique is not new, although every few years some influential person in the product world rediscovers its power and it gets attention once again. It also goes by multiple names. In any case, I'm convinced that everyone would do the technique if it didn't require so much actual work.

There are four main variations of this technique for four different situations:

1. Building products for businesses
2. Building platform products (e.g., public APIs)
3. Building customer-enabling tools used by employees of your company
4. Building products for consumers

Self Study for Future Discussion

https://www.svpg.com/team/martina-lauchengco/

Profile: Martina Lauchengco of Microsoft

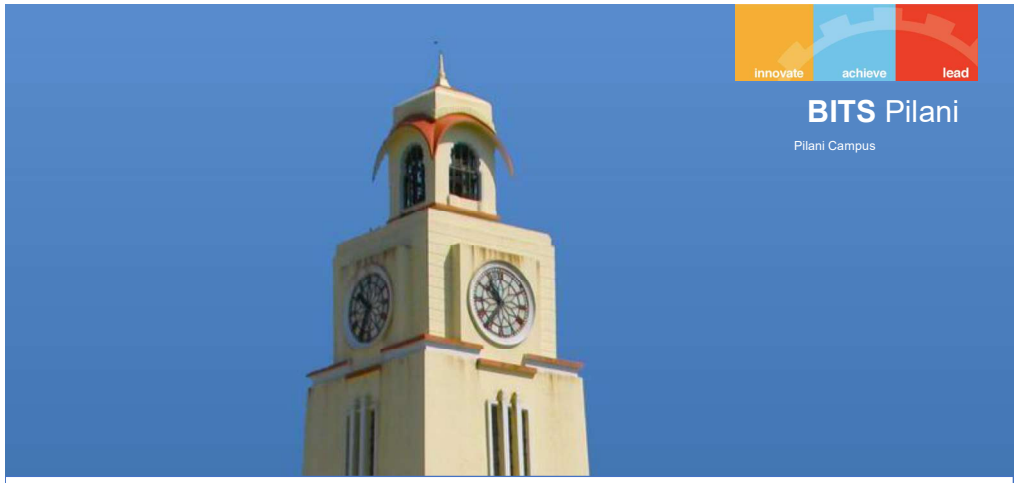
In 1993, Word 6.0 was the biggest release, feature-wise, Microsoft had ever produced.

In addition to all the new features, the team had another very large objective. Their code base had diverged, and it was extremely slow and costly for Microsoft to implement Word separately for each platform: Windows, DOS, and Mac. This code convergence effort was supposed to save Microsoft substantial development time, and—they tried to convince themselves—improve the offering since Word would have the same features on every platform.

It also meant that there was great pressure to get the release out so they could start to gain the efficiencies of a single code base.

At the time, Word for Mac was a relatively small market. It was only $60 million, versus Windows, which at that point was more than a $1 billion market. If you remember, back then Windows machines absolutely dominated, and the future of Apple was not a sure thing. However, the Mac community was also very vocal—with passionate

Chapter 40 of Inspired: How to create products customers love?
By Marty Kagan, Wiley Publication, 2017



Discovery Ideation Techniques

Customer Interviews

- There are many forms of customer interviews, so this is not really a single technique.
- Some are informal and some are more formal.
- In every user or customer interaction, we always have the opportunity to learn some valuable insights.
- Need to understand
  - Are your customers who you think they are?
  - Do they really have the problems you think they have?
  - How does the customer solve this problem today?
  - What would be required for them to switch?

Making the Most of Customer Interviews

Frequency. Establish a regular cadence of customer interviews. This should not be a once-in-a-while thing. A bare minimum would be two to three hours of customer interviews per week, every week.

Purpose. You are not trying to prove anything during these interviews, one way or the other. You're just trying to understand and learn quickly. This mindset is critical and needs to be sincere.

Recruiting users and customers. I talk much more about this when we discuss the usability testing technique, but for now, be sure to talk primarily to people in your intended target market. You're looking for about an hour of their time.

Location. It's always amazing to see customers in their native habitat. There's so much to learn just by observing their environment. But it's also fine to meet them somewhere convenient or have them come to your office. If you need to do this over a video call, that's not as good, but much better than not doing at all.

Preparation. Be clear beforehand what problem it is you think they have, and think about how you'll either confirm or contradict that.

Making the Most of Customer Interviews contd.

Who should attend. My favorite is to bring three people to these interviews: the product manager, the product designer, and one of the engineers from the team (we normally rotate among those that want to attend). Usually, the designer drives (because they've usually been trained how to do this well), the product manager takes notes, and the developer observes.

Interview. Work to keep things natural and informal, ask open-ended questions, and try to learn what they're doing today (not so much what they *wish* they were doing, although that's also interesting).

Afterward. Debrief with your colleagues to see if you've all heard the same things and had the same learnings. If you made any promises to the customer during that session, be sure you keep them.

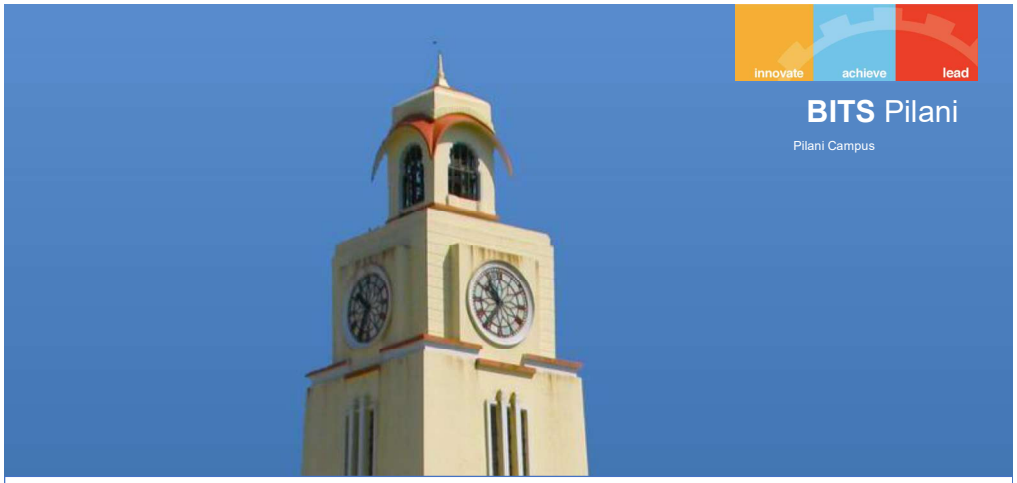
Summary

- For many product discovery efforts framing and true problem solving becomes critically important.
  - Discovery framing techniques facilitates these activities
- For complicated product efforts, it often helps to have some way to scope out and plan your discovery efforts.
  - Discovery planning techniques are focused on these aspects
- It is important to address the relevant question: How do we generate the types of ideas that are likely to truly help us solve the hard business problems that our leaders have asked us to focus on right now?
  - Discovery ideation techniques are tuned towards investigating this question

Thank you

Contents

- Discovery Ideation Techniques contd.
  - Concierge Test Technique
  - The Power of Customer Misbehavior
  - Hack Days
- Discovery Prototyping Techniques
  - Principles of Prototypes
  - Feasibility Prototype Technique
- Summary



Discovery Ideation Techniques contd.

Concierge Test Technique

- A *concierge test* is a relatively new name to describe an old but effective technique.
- The idea is that we do the customer's job for them—manually and personally.
- Just as if you went to a hotel concierge and asked if he could find you some theater tickets to a popular show.
- You don't really know the details of what that concierge is doing for you to get those tickets, but you do know that he is doing something.
- With this technique, *you* become the concierge. You do what the user or customer needs done for them.
  - You may have to ask them to train you first, but you are in their shoes doing the tasks they would do.

Concierge Test Technique contd.

A concierge test requires going out to the actual users and customers and asking them to show you how they work so that you can learn how to do their job, and so that you can work on providing them a much better solution.

If you are building a *customer-enabling* product, the users may be employees of your company, but the technique is the same—you go to these colleagues and ask them to teach you how they do their job.

Like the principle of shared learning, it is most valuable if the product manager, the product designer, and one of the engineers does the concierge test.

The Power of Customer Misbehavior

Historically, the two main approaches used by good teams to come up with product opportunities have been:

1. Try to assess the market opportunities and pick potentially lucrative areas where significant pain exists.
2. Look at what the technology or data enables—what's just now possible—and match that up with the significant pain.

You can think of the first as following the market, and the second as following the technology. Either way can get you to a winning product.

However, some of the most successful companies today have taken a third approach, and while it's not appropriate for every company, I would like to suggest that this is an extremely powerful technique that's largely underutilized and underappreciated in our industry.

This technique is to allow, and even encourage, our customers to use our products to solve problems other than what we planned for and officially support.

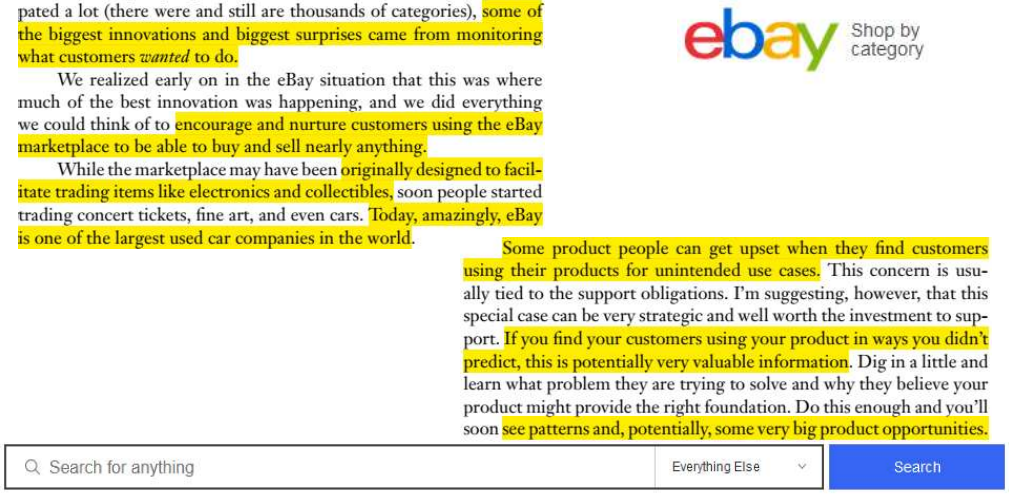
Case Study: Customer Misbehavior

From its earliest days, eBay has always had an "Everything Else" category. This is where people could buy and sell things that we at eBay couldn't anticipate people might want to trade. And while we anticipated a lot (there were and still are thousands of categories), some of the biggest innovations and biggest surprises came from monitoring what customers *wanted* to do.

We realized early on in the eBay situation that this was where much of the best innovation was happening, and we did everything we could think of to encourage and nurture customers using the eBay marketplace to be able to buy and sell nearly anything.

While the marketplace may have been originally designed to facilitate trading items like electronics and collectibles, soon people started trading concert tickets, fine art, and even cars. Today, amazingly, eBay is one of the largest used car companies in the world.

Some product people can get upset when they find customers using their products for unintended use cases. This concern is usually tied to the support obligations. I'm suggesting, however, that this special case can be very strategic and well worth the investment to support. If you find your customers using your product in ways you didn't predict, this is potentially very valuable information. Dig in a little and learn what problem they are trying to solve and why they believe your product might provide the right foundation. Do this enough and you'll soon see patterns and, potentially, some very big product opportunities.



The Power of Developer Misbehavior

While the eBay example was intended to be used by end users (buyers and sellers), this same concept is what's behind the trend toward exposing some or all of a product's services via programmatic interfaces (public APIs).

I consider developers to be one of the consistently best sources of truly innovative product ideas.

With a public API, you are essentially saying to the developer community, "These are the things we can do—perhaps you can leverage these services to do something amazing that we couldn't anticipate ourselves."

Facebook's platform strategy is a good example of this. They opened up access to their social graph to discover the types of things that developers might be able to do once they could leverage this asset.

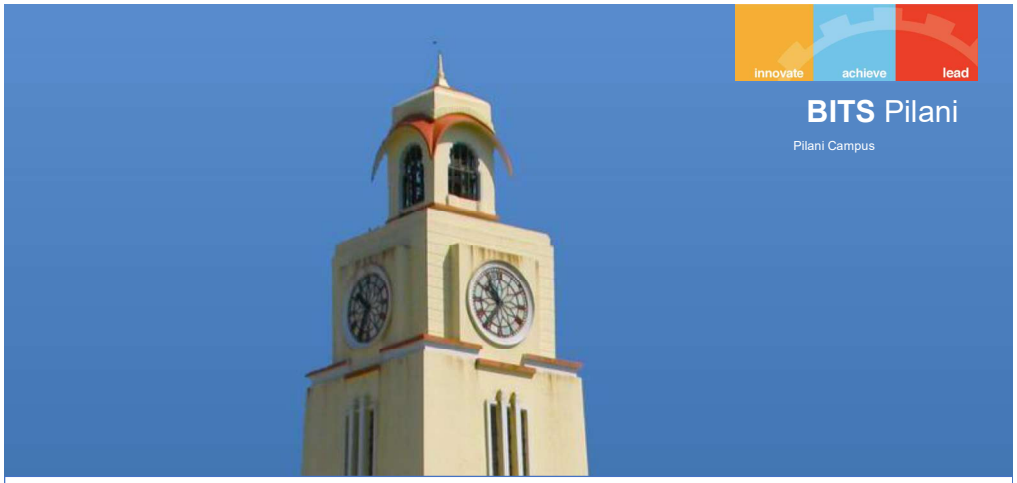
I have been a long-time fan of public APIs as a part of a company's product strategy. I consider developers to be one of the consistently best sources of truly innovative product ideas. Developers are in the best position to see what's just now possible, and so many innovations are powered by these insights.

Hack Days

- The two main types of hack days are directed and undirected.
- *Undirected* hack day
  - People can explore whatever product ideas they like, so long as it's at least loosely related to the mission of the company.
- *Directed* hack day
  - There's a customer problem (for example, something is really difficult to learn and use, or it takes too long to do) or business objective we've been assigned (for example, "Reduce the customer churn rate" or "Increase customer lifetime value")
  - People from the product teams to self-organize and work on any ideas they like that might address this objective.
- The goal is for the self-organizing groups to explore their ideas and create some form of prototype that can be evaluated, and if appropriate, tested on actual users.

Directed Hack Days: Benefits

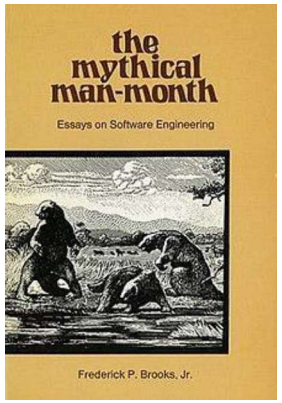
- Two major benefits to these directed hack days.
- The first is practical, as the technique facilitates the inclusion of engineers at ideation.
  - Best ideas come from the engineers on the team, and we need to ensure this is happening.
  - It should be happening on an ongoing basis, but this technique will ensure it happens.
- The second benefit is cultural.
  - This is one of the effective techniques for building a team of missionaries rather than mercenaries.
  - The engineers, if they haven't already, are now diving much deeper into the business context and playing a much larger role in terms of innovation.



Discovery Prototyping Techniques

Principles of Prototypes

"Plan to throw one away; you will, anyhow."



Frederick Phillips Brooks Jr. (1931 – 2022)

What is a Prototype?

- Prototypes of various forms have been around for as long as we've been applying technology to solve problems.
- Brook's quote (from the previous slide) is as relevant today as when it was first published (in 1975!)
  - Many things have changed, not the least of which is that the tools and techniques we have for developing prototypes and testing them have progressed dramatically
- However, those involved in product development, even thought leaders, still have a very narrow interpretation of what is meant by the term *prototype*.
- Typically they associate the term prototype with the type that they were first exposed to, for example:
  - Prototype to test for feasibility
  - Prototype for usability testing
- But there are in fact many very different forms of prototypes
- Each with different characteristics and each suited to testing different things
  - Feasibility Prototypes
  - User Prototypes
  - Live-Data Prototypes
  - Hybrid Prototypes

Principles of Prototypes

- The overarching purpose of any form of prototype is to learn something at a much lower cost in terms of time and effort than building out a product.
  - All forms of prototype should require *at least* an order of magnitude less time and effort as the eventual product.
- Realize that one of the key benefits of any form of prototype is to force you to think through a problem at a substantially deeper level than if we just talk about it or write something down.
  - This is why the very act of creating a prototype so often exposes major issues otherwise left uncovered until much later.
- Similarly, a prototype is also a powerful tool for team collaboration.
  - Members of the product team and business partners can all experience the prototype to develop shared understanding.

Principles of Prototypes contd.

- There are many different possible levels of *fidelity* for a prototype. The fidelity primarily refers to how realistic the prototype looks. There is no such thing as one appropriate level of fidelity.
  - Sometimes we don't need the prototype to look realistic at all, and other times it needs to be very realistic.
  - The principle is that we create the *right* level of fidelity for its intended purpose, and we acknowledge that lower fidelity is faster and cheaper than higher fidelity, so we only do higher fidelity when we need to.
- The primary purpose of a prototype is to tackle one or more product risks (value, usability, feasibility, or viability) in discovery.
  - In many cases, the prototype goes on to provide a second benefit, which is to communicate to the engineers and the broader organization what needs to be built.
  - This is often referred to as *prototype as spec*.
  - In some cases, the prototype will need to be supplemented with additional details (usually, use cases, business rules, and acceptance criteria).

Feasibility Prototype Technique

- Common examples of situations wherein engineers may identify a significant feasibility risk involved in solving a particular problem
  - Algorithm concerns
  - Performance concerns
  - Scalability concerns
  - Fault tolerance concerns
  - Use of a technology the team has not used before
  - Use of a third-party component or service the team has not used before
  - Use of a legacy system the team has not used before
  - Dependency on new or related changes by other teams
- The main technique used for tackling these types of risks is for one or more of the engineers to build a *feasibility prototype*.
- An engineer will create the feasibility prototype because it is typically code (as opposed to most prototypes created by special-purpose tools intended to be used by product designers).

Feasibility Prototype

- A feasibility prototype is a long way from a commercially shippable product—the idea is to write just enough code to mitigate the feasibility risk.
  - This typically represents just a small percentage of the work for the eventual shippable product.
- Most of the time the feasibility prototype is intended to be throwaway code
  - It is intended to be just enough to collect the data, for example, to show that performance would likely be acceptable or not.
  - There is usually no user interface, error handling, or any of the typical work involved in productization.
- Building a feasibility prototype requires usually just a day or two of time.
  - Exploring a major new technology could very well take significantly longer

Feasibility Prototype contd.

- The amount of time the feasibility prototype is estimated to take comes from the engineers, but whether or not the team takes that time depends on the product manager's judgment call as to whether it's worth pursuing this idea.
- While it's the engineers who do this feasibility prototyping work, it is considered discovery work and not delivery work.
  - It's done as part of deciding whether to even pursue this particular approach or idea.
- Whenever product teams are found to have grossly underestimated the amount of work required to build and deliver something, it is usually because they have proceed to delivery without adequately considering the feasibility risk. It may be that:
  - The engineers were simply too inexperienced with their estimates
  - The engineers and product manager had an insufficient understanding of what was going to be needed
  - The product manager did not give the engineers sufficient time to truly investigate

Summary

- Discovery ideation techniques such as concierge test technique, the power of customer misbehavior, and hack days offer facilities to teams for idea generation
- Prototyping is a critical activity in the product development journey

Thank you

Contents

- Discovery Prototyping Techniques contd.
  - User Prototype Technique
  - Live-Data Prototype Technique
  - Hybrid Prototype Technique
- Assignment 1: Any clarifications?
- Summary

User Prototype Technique

- A user prototype one of the most powerful tools in product discovery—is a simulation.
- There is a wide range of user prototypes.
- At one end of the spectrum are low-fidelity user prototypes.
  - A low-fidelity user prototype doesn't look real—it is essentially an interactive wireframe. Many teams use these as a way to think through the product among themselves.
  - Low-fidelity user prototypes, however, represent only one dimension of your product—the information and the workflow—there's nothing there about the impact of visual design or the differences caused by the actual data.
- At the other end of the spectrum are high-fidelity user prototypes.
  - A high-fidelity user prototype is still a simulation; however, now it looks and feels very real.
  - The data you see is very realistic, but it's not real either – mostly meaning its not live.

The Value of User Prototypes

A user prototype is key to several types of validation and is also one of our most important communication tools.

There are many tools for creating user prototypes—for every type of device, and for every level of fidelity. The tools are mainly developed for product designers. In fact, your product designer almost certainly already has one or more favorite user prototyping tools.

It's also the case that some designers prefer to hand-code their high-fidelity user prototypes, which is fine so long as they are fast, and they are willing to treat the prototype as disposable.

What is the Biggest Limitation of a User Prototype?

- It's not good for *proving* anything—like whether or not your product will sell.
  - Where a lot of novice product people go sideways is when they create a high-fidelity user prototype and they put it in front of 10 or 15 people who all say how much they love it. They think they've validated their product, but unfortunately, that's not how it works.
- We have much better techniques for validating value, so it's important that you understand what a user prototype is *not* appropriate for.
- User prototyping is one of the most important techniques for product teams, so it is well worth developing your team's skills and experience in creating user prototypes at all levels of fidelity.

A Quick Case Study

- In an e-commerce user prototype example, when a search is done for a particular type of mountain bike, it always comes back with the same set of mountain bikes. But if looked at closely, they're not the actual bikes asked for. And it is noticed that every time it is searched, it's always the same set of bikes no matter what price or style is specified.
- Is user prototyping relevant here?
  - If yes, why?
  - If no, why not?
- Depends ☺
  - If you are trying to test the relevance of the search results, this would not be the right tool for the job.
  - But if you are trying to come up with a good overall shopping experience or figure out how people want to search for mountain bikes, this is probably more than adequate, and it's very quick and easy to create

Live-Data Prototype Technique

- Sometimes, in order to address a major risk identified in discovery, we need to be able to collect some actual usage data. But we need to collect this evidence while in discovery, well before taking the time and expense of building an actual scalable and shippable product.
  - For example, when applying game dynamics, search result relevance, many social features, and product funnel work.
- This is the purpose of a live-data prototype.
- A *live-data prototype* is a very limited implementation. It typically has none of the productization that's normally required, such as the full set of use cases, automated tests, full analytics instrumentation, internationalization and localization, performance and scalability, etc.
- The live-data prototype is substantially smaller than the eventual product, and the bar is dramatically lower in terms of quality, performance, and functionality. It needs to run well enough to collect data for some very specific use cases, and that's about it.

The Objective of a Live-Data Prototype

The key is to be able to send some limited amount of traffic, and to collect analytics on how this live-data prototype is being used.

Creating a Live-Data Prototype

- When creating a live-data prototype, our engineers don't handle all the use cases. They don't address internationalization and localization work, they don't tackle performance or scalability, they don't create the automated tests, and they only include instrumentation for the specific use cases we're testing.
- A live-data prototype is just a small fraction of the productization effort (in my experience, somewhere between 5 and 10 percent of the eventual delivery productization work), but you get big value from it.

Live-Data Prototype: Limitations

- First, this is code, so engineers must create the live-data prototype, not your designers.
- Second, this is not a commercially shippable product, it's not ready for primetime, and you can't run a business on it.
  - So, if the live-data tests go well, and you decide to move forward and productize, you will need to allow your engineers to take the time required to do the necessary delivery work.
  - It is definitely *not* okay for the product manager to tell the engineers that this is "good enough." That judgment is not up to the product manager. And the product manager does need to make sure key executives and stakeholders understand the limitations as well.
- The technology for creating live-data prototypes is so good that we can often get what we need in just a couple days to a week. And once we have it we can iterate very quickly.

Live-Data Prototype: Importance

- What's important is that actual users will use the live-data prototype for real work, and this will generate real data (analytics) that we can compare to our current product—or to our expectations—to see if this new approach performs better.

Hybrid Prototype Technique

- The techniques discussed so far address different needs:
  - User prototypes are pure simulations
  - Feasibility prototypes address technical risks
  - Live-data prototypes are designed to be able to collect evidence, or even statistically significant proof, as to the effectiveness of a product or an idea
- These three categories of prototypes handle most situations well
- However, a wide variety of hybrid prototypes also combine different aspects of each of these in different ways
- An exceptionally powerful tool for learning quickly in product discovery—is today often referred to as a *Wizard of Oz* prototype

A Wizard of Oz Prototype

- A Wizard of Oz prototype combines the front-end user experience of a high-fidelity user prototype but with an actual person behind the scenes performing manually what would ultimately be handled by automation.
- A Wizard of Oz prototype is *absolutely* not scalable, and we would never send any significant amount of traffic to this.
- But the benefit from our perspective is that we can create this very quickly and easily, and from the user's perspective, it looks and behaves like a real product.
- Can you think of a scenario where this kind of a prototype can be helpful?

An Example of a Wizard of Oz Prototype

- For example, imagine that today you have some sort of live chat–based help for your customers, but it's only available during the hours when your customer service staff is in the office.
- You know that your customers use your product from all around the world at all hours, so you would like to develop an automated chat-based system that provides helpful answers anytime.
- You could (and should) talk to your customer service staff about the types of inquiries they routinely get and how they respond (a *concierge test* could help you learn that quickly). Soon you will want to tackle the challenges of this sort of automation.
- One way to learn very quickly and test out several different approaches is to create a Wizard of Oz prototype that provides a simple, chat-based interface.
  - However, behind the scenes it is literally you as product manager, or another member of your team, who is receiving the requests and composing responses.

Characteristics of a Wizard of Oz Prototype

- These types of hybrids are great examples of the *build things that don't scale* philosophy of product discovery.
- By being a little clever, we can quickly and easily create tools that let us learn very quickly.
- Admittedly, it's mainly qualitative learning, but that's often where our biggest insights come from anyway.

Assignment 1: Any clarifications?

Summary

- Feasibility prototypes address technical risks
- Live-data prototypes are designed to be able to collect evidence, or even statistically significant proof, as to the effectiveness of a product or an idea
- A wide variety of hybrid prototypes also combine different aspects of each of these in different ways

Thank you

Contents

- Product Requirements
  - Product requirements specification
  - Requirements attributes
  - Product requirements lifecycle
  - Gathering requirements
  - Requirements analysis
  - Requirement verification and validation
  - Requirement types
  - Documentation

Do You Need Product Requirements Specification?

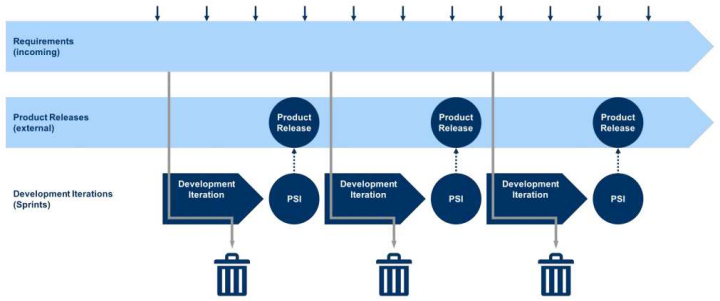
- Attention to requirements specification increased during the 1980s and 1990s in order to master the growing complexity of software and systems development.
- However, starting in the second half of the 1990s, the movement of agile development widely challenged the use of requirements specification documents.
- Now, do we need product requirements specifications today?
- The answer is:
  - It depends
- While a lean and agile approach to requirements and design is indisputably state-of-the-art, it may actually include the use of requirements specification.
  - Because, sometimes, attempting to be too lean and entirely omit requirements specification does more harm than good.

When can you do without product requirements specification?

- When is “pure” agile development without traditional requirements specification your best choice?
  - These are situations where agile requirements practices mostly based on user stories will do a good job: One or few closely collaborating teams develop a system that exists in one or few instances only. This is, for instance, the case with low to medium size web services or mobile apps.
- In addition to numbers of teams and product instances, you may wish a few more prerequisites to be in place for this approach to work out, for instance:
  - Your development teams should have the opportunity of close customer and user involvement.
  - You may want your teams to be relatively stable (i.e., low turnover rate) over the system’s entire lifetime.

When can you do without product requirements specification? contd.

- In this approach your requirements will hardly ever manifest explicitly in written form.
- User stories rather represent requirements but do not actually define them.
- Requirements mostly manifest in the implemented product release (or PSI, potentially shippable increment), and in your automated tests.
- But no kind of requirements statement will survive after the end of its development iteration or agile Sprint.



When do you need a product requirements specification?

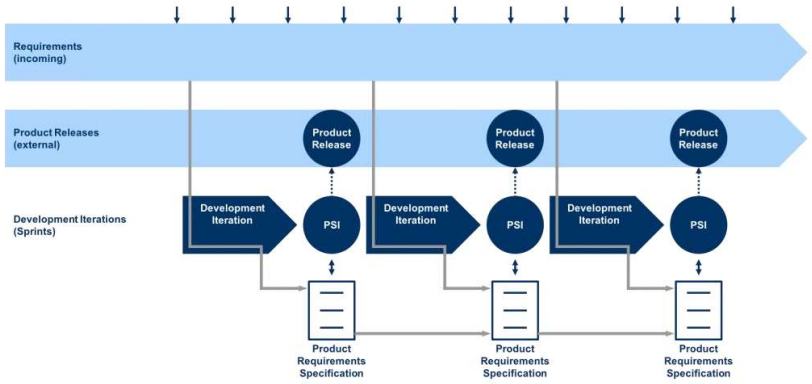
- There are quite some situations where you need a product requirements specification. (Of course, you should create and use it in a lean and agile manner.) These situations include:
- High support and maintenance demands
  - Your customers or your business environment often raise questions of the types: Can the system do X or Y? What would be needed to accomplish Z? Having an up-to-date product specification will then tremendously ease your life.
- Development of product variants
  - You develop not only one product but a wide spectrum of similar yet different variants. This is often the case with technical hardware/software devices like sensors or control units. Then you want to reuse existing assets instead of developing similar functionality again and again. Reuse of product requirements along with their implementation gives you the longest lever.

When do you need a product requirements specification? contd.

- Subcontracted development
  - Here the degree of needed specification detail can vary, depending on your subcontractor relationship. However, nearly always some kind of requirements specification will form the core of the development contract. You better want to have this contract be clear and precise.
- Regulatory compliance
  - You have no choice but to document your requirements explicitly. Requirements define essential product characteristics and design decisions that must be traced down further to implementation and test. The requirements specification also is the indispensable basis for controlled and auditable product change.
- Wherever such situations exist, and they exist in tens of thousands throughout the industries, organizations will need product requirements specifications.

When do you need a product requirements specification? contd.

- These specifications complement product increments and preserve all relevant requirements information



Requirements Attributes (features)

- To be considered as good - each product requirement must possess certain attributes.
- For example, in the software engineering there is the INVEST principle, so the requirements should be:
  - Independent, Negotiable, Valuable, Estimable, Small, Testable.
- For the general product requirements - the desired attributes include the following:
  - **Clear** - the requirement is unambiguously defined.
  - **Complete** - all that is needed - is consisted in the requirement.
  - **Credible** - the requirement must be technically possible to be achieved.
  - **Consistent** - the requirement is not in collision with other requirements.
  - **Verifiable** - the requirement must be verified and validated (so it should be expressed in physical terms and measurable attributes).
  - **Necessary** - if some requirement brings clear business value - it can be treated as more necessary than other requirements that are “nice to have” (e.g. product color).

Product Requirements Lifecycle

- The product requirements lifecycle includes the following phases:
  - Requirements gathering
  - Requirements analysis
  - Requirements verification/validation
  - Requirements management and tracking
  - Requirements specification (documentation)

Gathering of Product Requirements

- In this phase, we have to determine the sources of the product requirements and the approach for their handling and management. It is important to identify all of the sources:
  - **Objectives** - also known as 'Business concerns' or 'Critical Success Factors', represent the comprehensive goal of the product. Special attention should be paid to the cost of reaching the target (through a feasibility study).
  - **Domain knowledge** - a production engineer needs to have knowledge of the product's application domain and to be ahead of others; to resolve conflicts in the requirements and to be a mediator between all participants.
  - **Stakeholders** - some products might prove to be unsatisfactory, because of the different perspectives of the stakeholders. The product engineer needs to identify a correct representation and to manage these situations (different culture and politics).
  - **Operating environment** - product requirements come from the environment in which the product is used on daily basis.
  - **The organizational environment** - the product will have to support the business environment and the processes (conditions) that it will be used it. New product should not bring unplanned changes, so it requires careful analysis by the product creators, before making a final decision.

Gathering of Product Requirements (contd.)

- When all sources are identified - the product engineer can start the requirements elicitation (gathering). This area deals with the techniques for selection and extraction, which can be particularly sensitive. For example, users may have difficulty in defining actions and sometimes they omit important information. These techniques are not passive and require hard work for the articulation of all requirements. The *interview* represents a traditional way of collecting client's requests. This technique has certain advantages and limitations.
- It is necessary to prepare scenarios for the requirements elicitation:
  - Providing a framework for defining the questions: What if...? How is this done..? etc.
  - Using a technique known as Conceptual modeling - i.e. Case Modeling and Diagrams.
- Also, holding meetings (brainstorming) and group work can be more efficient than individual techniques of observation (observing how users will use the product and interact with it).

Requirements Analysis

- After the process of requirements gathering - a process for their analysis is necessary. Requirements analysis refers to the processes required in order to:
  - Identify and resolve the conflicts between the product requirements;
  - Detection of product features, and how the product will behave in real environments;
  - Elaboration of product requirements in the process of product development.
- The traditional view of the analysis requires the use of conceptual modeling and other methods for analysis, such as the Structured Analysis and Design Technique (SADT).
- The most important thing is to accurately describe the clients' requirements, in order to facilitate their validation, verification of their implementation, and the cost estimates.

Requirements Analysis contd.

- Requirements analysis includes:
  - Requirements Classification
  - Conceptual Modelling
  - Architectural Design and Requirements Allocation
  - Conflict resolution (Requirements Negotiation)

Requirement Verification and Validation

- Requirement validation checks the specification to ensure that
  - All product requirements are unambiguously defined
  - All inconsistencies and errors are discovered and corrected
  - The product will meet all standards defined for the: usability, safety, quality etc.
- The primary mechanism for requirement validation is a formal technical review.
- The review team includes engineers, customers, and other stakeholders
- They check product specification for content errors, areas that need clarification, insufficient information, inconsistencies (a major problem when making large products or systems), conflicting demands, or unrealistic requirements (which cannot be achieved).
- Although requirement validation can be done in order to uncover potential errors, it is useful to check each requirement according to a predefined list of questions.

Requirement Verification and Validation contd.

- The following questions are only a small part of what can be asked:
  - Are the requirements clearly stated? Can they be misinterpreted?
  - Are all the aspects of the product operation (functioning) covered?
  - Is the source of the request identified (e.g. person, regulation, document)? Has the final form of the requirement been verified from the original source?
  - Is the requirement quantified?
  - What other requirements are associated with this requirement? Are they clearly stated using a reference matrix or other mechanism?
  - Does the requirement affect domain restrictions?
  - Can the requirement be tested? If it can, can we determine tests (so-called validation criteria) for checking the requirement?
  - Can the requirement be tracked in any product that will be created?

Product Requirements Management and Tracking

- Requirement management is a set of activities that helps the project team identify, control, and track requirements and their changes at any point in the product's life.
  - In this step, requirements prioritization and re-ordering can be done.
- The requirements management begins with identification. Each requirement is assigned a unique identifier of the following form:
  - <type requirement> <requirement Id> where < type requirement> can receive these possible values: F = functional requirement, B = behavioral requirement, I = input requirement, O = output requirement etc. Thus, F09 denotes a function requirement number 9.

Product Requirements Management and Tracking contd.

- Once the requirements have been identified, monitoring (tracking) tables are developed. Each tracking table links specific requirements to one or more aspects of the product or the environment. Some of the many tracking charts are the following:
  - **Properties Tracking Table** - displays how the requirements relate to the visible features of the product.
  - **Source (reason) Tracking Table** - identifies the source of each requirement.
  - **Dependency Tracking Table** - indicates how the requirements are related to each other.
  - **Subsystem Tracking Table** - categorizes requirements according to the subsystems they belong to.

Product Requirements Documentation

- After the gathering and the analysis of the product requirements - the product analyst and the project team should specify all product requirements. Under the requirements specification - there should be a composition (drafting) of a Product Requirements Document (PRD).
- The product requirements document is comprised of: full product's overview, the product features, working environment etc. It is not solely dedicated to the product's functions and features, but it should list all functional and non-functional requirements (such as usability, safety, reliability, etc.).

Product Requirements Documentation contd.

- While writing the PRD, we must consider the following aspects of the product:
  - Define the main product principles (e.g. it should be reliable and have ease of use).
  - Define the product's purposes, i.e. the business values that it will bring for your customers (ease of use, cheap price, new features).
  - Define your clients / customers, your competition, and your product development team.
  - Define the goals of the product, and tasks that can be fulfilled by using the product.
  - Define the different user profiles, e.g. young adults, elderly people, wide audience etc.
- The PRD should be written in clear and concise style, so the product developer will have a clear understanding about the function of the device, and can jump straight into the design.

Product Requirement Types

- There are particular types of requirements that each product should fulfill and here we list the most relevant ones.
- **Functionality**
  - Each product is developed with its main purposes (goals). For example - the cars are designed to take us to the destination by driving, with certain technical characteristics, exterior and interior features etc. The product's functional requirements should include all technical details in order to provide its uninterrupted functionality.
- **Quality**
  - Quality is any element (feature), measurable or not, that gives things values beyond their functionality and features. Typical quality requirements include: reliability of the product, consistency, durability, availability, customer experience, look and feel, performance, maintainability, materials / ingredients etc.
- **Usability**
  - Usability requirements are created to ensure that the product will be easy to use. Usability is a broad concept, but it can be split into more elementary ones - like intuitive and easy to learn, so the users are able to flow through the tasks without being interrupted. Also - they should increase the productivity and performance of the user while using the product.

Product Requirement Types contd.

- **Reliability**
  - Reliability is one of the most important non-functional (quality) requirements. Reliability is measured as time to failure, probability of failure and failure-free operation. It also defines the mean time to repair and between repairs, coefficient of availability and unavailability, failure rate etc.
- **Safety**
  - Safety is often introduced as requirement in order to avoid, or reduce potential risks. In physical products (items) - safety relates to well-being, health and life protection of the users, while in the IT systems - it refers to protection of user's personal data (e.g. credit card number etc.).
- **Packaging**
  - Packaging requirement refers to the product packaging and its purpose is to protect the product for transport damages, and also for marketing (design) purposes. Packaging requirements are subject to regulations in all countries worldwide. For example, European Union has introduced Packaging Regulations Directive in 1998, and it must be obeyed in all EU countries. It starts with massive containers, dangerous materials, chemicals and also includes food products and small bags.

Summary

- It is critical to understand whether there is a need for product requirement specifications
- There are various stages of the product requirements lifecycle
- Gathering, analysing, verifying, and validating are critical activities in the process
- There are several different types of product requirements

Thank you

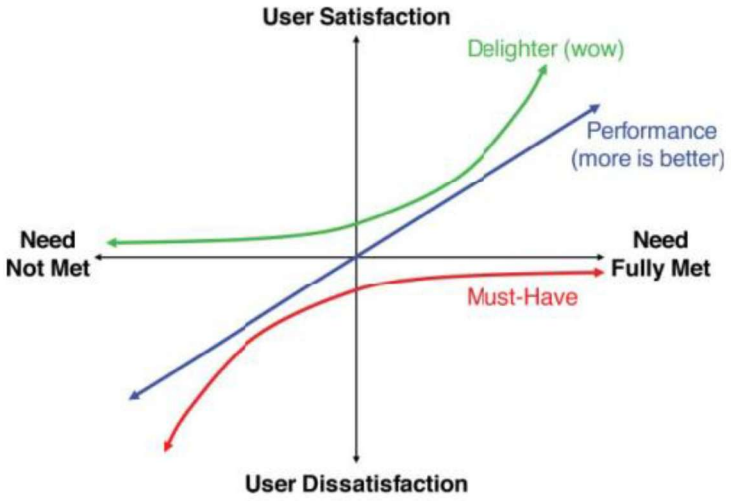
Contents

- Specify Minimum Viable Product (MVP) feature set
  - Kano model
  - What is MVP?
  - Prioritizing features

Kano Model

- A framework for understanding customer needs and satisfaction developed by quality management expert Noriaki Kano
- Kano model also plots a set of two parameters on horizontal and vertical axes:
  - (1) how fully a given customer need is met (horizontal axis)
  - (2) the resulting level of customer satisfaction (vertical axis)
- The horizontal axis ranges from the need not being met at all on the left to the need being fully met on the right
- The vertical axis ranges from complete customer dissatisfaction at the bottom to complete satisfaction at the top
- The utility of the model is that it breaks customer needs into three relevant categories that you can use:
  - Performance needs
  - Must-have needs
  - Delighters

Kano Model: The Plot



Understanding Different Needs

- With performance needs, more is better.
  - As the need is more fully met, the resulting customer satisfaction increases. Say you were shopping for a car and considering two different models. If they were identical in all aspects but Car A had twice the fuel efficiency (e.g., miles per gallon) of Car B, you would have a preference for Car A. Fuel efficiency is a performance benefit for cars.
- Must-have needs don't create satisfaction by being met. Must-have features are "table stakes" or "cost of entry"—boxes that must be checked for customers to be satisfied with your product.
  - Sticking with the car example, seat belts would be a must-have feature. If you were interested in a car but realized it had no seat belts, you wouldn't buy it for fear of getting hurt if you were in an accident. Your must-have need for a reasonable level of safety is not being met. That being said, if Car A had five seat belts and Car B had 100 seat belts, you wouldn't say that Car B is 20 times better than Car A. Once you have one seat belt per passenger, your must-have need has been met.

Understanding Different Needs contd.

- Delighters provide unexpected benefits that exceed customer expectations, resulting in very high customer satisfaction.
- The absence of a delighter doesn't cause any dissatisfaction because customers aren't expecting it.
  - Returning again to cars, GPS navigation systems were a delighter when the first car models came out with that new technology in the mid-1990s. They meant no longer having to print out directions from your computer and no more getting lost. This feature fundamentally changed how people drove from point A to point B, resulting in customer delight.

Different Features in Cars

- Cars did not always have built-in cup holders. Chrysler changed that when it introduced the minivan in the early 1980s, which had two functional cup holders sunk into the plastic of the dashboard.
  - They were delighters because drivers no longer had to worry about spilling their beverages as they drove.
- Cup holders are now ubiquitous in cars—and an increasing percentage of cars come with GPS navigation as a standard feature.
- That illustrates an important aspect of the Kano model: Needs migrate over time.
  - Yesterday's delighters become today's performance features and tomorrow's must-haves.
  - Growing customer expectations and competition continuously raise the bar over time

Kano Model Exhibits Hierarchy

- The Kano model also exhibits hierarchy
- The fact that your product has a delighter doesn't matter if it's missing a must-have.
  - A navigation system would be pointless in a car with no seat belts.
- You have to meet basic needs before you can get credit for performance features
- And your product must be competitive on performance features before delighters matter
- You can think of this as a three-tier pyramid with must-haves on the bottom, performance features just above that, and delighters at the top.

Applying the Kano Model

- You can apply the Kano model to gain clarity about the problem space.
- Think about the customer benefits that are relevant in your product category and classify them into the three categories of must-haves, performance benefits, and delighters.
- Evaluating competitive products and reading product reviews can help inform you as you create this framework.

Minimum Viable Product (MVP)

- You are not going to start off by designing a new product that delivers on your full value proposition
  - That would take too long and be too risky
- For your MVP, you want to identify the minimum functionality required to validate that you are heading in the right direction
- The goal is to identify the top three to five features for each benefit.
- There is not much value in looking beyond those top features right now because things will change—a lot—after you show your prototype to customers.

User Stories: Features with Benefits

User stories (used in Agile development) are a great way to write your feature ideas to make sure that the corresponding customer benefit remains clear. A user story is a brief description of the benefit that the particular functionality should provide, including whom the benefit is for (the target customer), and why the customer wants the benefit. Well-written user stories usually follow the template:

As a [type of user],
I want to [do something],
so that I can [desired benefit].

Here's an example of a user story that follows this template:

As a professional photographer,
I want to easily upload pictures from my camera to my website,
so that I can quickly show my clients their pictures.

Guidelines for Writing "Good" User Stories

- Independent
  - A good story should be independent of other stories. Stories shouldn't overlap in concept and should be implementable in any order.
- Negotiable
  - A good story isn't an explicit contract for features. The details for how a story's benefit will be delivered should be open to discussion.
- Valuable
  - A good story needs to be valuable to the customer.
- Estimable
  - A good story is one whose scope can be reasonably estimated.
- Small
  - Good stories tend to be small in scope. Larger stories will have greater uncertainty, so you should break them down.
- Testable
  - A good story provides enough information to make it clear how to test that the story is "done" (called acceptance criteria).

Breaking Features Down

- Once you have written high-level user stories for your top features, the next step is to identify ways to break each of them down into smaller pieces of functionality; a process called "chunking"
- The goal is to find ways to reduce scope and build only the most valuable pieces of each feature.
- Say you are working on a photo sharing application and start out with the user story:
  - "As a user, I want to be able to easily share photos with my friends so that they can enjoy them."
  - What can be a way to break this story down?
- One way to break this story down is by the various channels a customer can use to share photos: Facebook, Twitter, Pinterest, email, text message, and so forth.
- Each of those would be a distinct feature chunk or smaller scope user story.
- You may not need to build out all of these sharing channels for your MVP.

Breaking Features Down contd.

- Even if you decided that you did, it helps to break the story down to be more specific in your product definition
  - To enable more accurate scoping from development
  - To allow you to explicitly prioritize the order in which you build the chunks
- You might also limit scope by enabling the user to share only the photo and nothing else for your MVP.
- You may have ideas for additional functionality down the road such as adding an optional message to each photo or the ability to tag users in photos.
- Each of those would be a distinct feature chunk.

Return on Investment (ROI)

- After you have finished chunking your feature ideas, you should perform a second-pass prioritization that accounts for both the value and the effort.
- A simple way to illustrate ROI is to imagine that $100 is invested in a stock.
- Several months later, it is worth $200 and it is sold,
- There is a return, or a net profit of $200 - $100 = $100
- Since the initial investment was $100, the ROI is $100 / $100 = 1, or 100%. The formula for ROI is

$$\text{ROI} = \frac{\text{Final Value} - \text{Investment}}{\text{Investment}} = \frac{\text{Return}}{\text{Investment}}$$

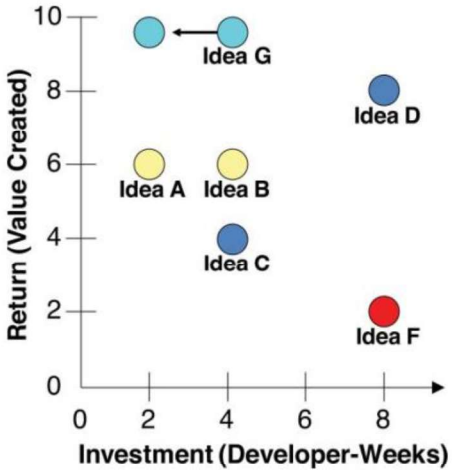
- What is the difficulty with directly using this formula in the product development context?

Using ROI in Product Development

In the context of investing, both of the numbers you plug into the formula are monetary amounts (e.g., dollars). However, that's usually not the case for ROI in the context of product development. When you are building a product or feature, the investment is usually the time that your development resources spend working on it, which you generally measure in units such as developer-weeks (one developer working for one week). It's true that you could probably calculate an equivalent dollar amount, but people use units like developer-weeks because they are simpler and clearer.

Similarly, in the context of developing a new product, "return" is often not a dollar amount. Instead, it is usually some relative measure of the amount of customer value you expect a certain feature to create. As long as you use an appropriate number scale to estimate customer value, the ROI calculations will work out fine. You need to use a "ratio scale," which just means that the scores you use are in proportion to their value. For example, say you use a 0 to 10 scale for customer value and estimate scores for all your feature chunks. Using a ratio scale, if one feature chunk has a score of 10 and a second feature chunk has a score of 5, that should mean that the first feature would create double the amount of customer value as the second.

Visualizing ROI

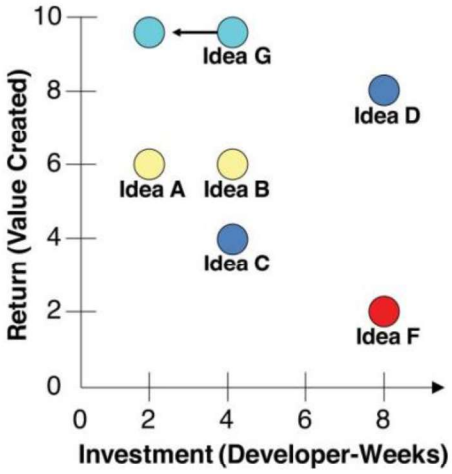


feature ideas A and B, both of which are estimated to create 6 units of customer value. However, idea B requires 4 developer-weeks to implement while idea A requires only 2 developer-weeks. The ROI for idea A is 6 ÷ 2 = 3, while the ROI for idea B is 6 ÷ 4 = 1.5. You should prioritize feature A above feature B.

Sometimes two features offer about the same ROI. Look at feature ideas C and D. Idea C offers 4 units of customer value for 4 developer-weeks, for an ROI of 4 ÷ 4 = 1. Idea D offers 8 units of customer value for 8 developer-weeks, for an ROI of 8 ÷ 8 = 1. When you have two feature ideas with the same ROI, it's best to prioritize the smaller scope idea higher because it takes less time to implement. You will deliver the value to customers more quickly—and by having the feature live sooner, you will get valuable customer feedback on it sooner, too.

There are bad ideas out there too—such as idea F, which offers 2 units of customer value for 8 developer weeks, an ROI of 2 ÷ 8 = 0.25. The large effort of a low-ROI idea is often recognized early as the team works on implementing it; however, they usually don't realize the low customer value until after launch. Google Buzz and Google Wave are examples of low-ROI projects that each took a larger number of developer-hours to build but were shut down shortly after launching when customer reaction indicated that they had not created enough value.

Visualizing ROI contd.

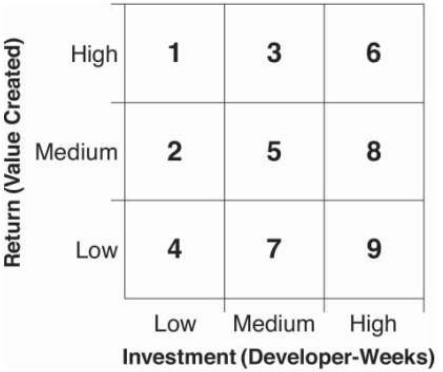


Good product teams strive to come up with ideas like idea G -the ones that create high customer value for low effort. Great product teams are able to take ideas like that, break them down into chunks, trim off less valuable pieces, and identify creative ways to deliver the customer value with less effort than initially scoped—indicated in the figure by moving idea G to the left.

Some people struggle to create numerical estimates of customer value they feel are accurate. However, that isn't something to worry about too much, since this isn't about achieving decimal point precision. Even the effort estimates aren't likely to be very precise, because you haven't fully designed the features yet. You can't expect developers to give you accurate estimates based on just a high-level description of a feature. The accuracy of the estimates should be proportional to the fidelity of the product definition. The main point of these calculations is less about figuring out actual ROI values and more about how they *compare* to each other. You want to focus on the highest ROI features first and avoid the lower ROI features.

You can sort your list of feature chunks by estimated ROI to create a rank-ordered list—which is a good starting point to help decide which feature chunks should be part of the MVP candidate. However, sometimes you can't just follow the strict rank order to create a "complete" MVP; you might need to skip down to include important features.

Approximating ROI

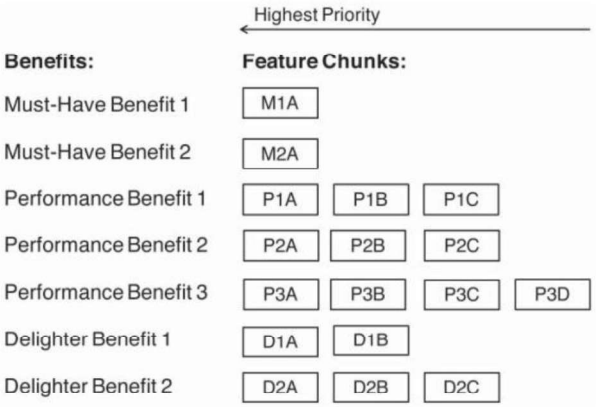


If you are struggling with creating numerical estimates of customer value or development effort, you can score each feature idea high, medium, or low on customer value and on effort.

All of your feature ideas will fall into one of the nine buckets. Even though you won't be calculating ROIs for each feature numerically, you can rank order the nine buckets based on ROI, as shown in the figure. So all the features in square number 1, which has the highest value and the lowest effort, would be higher priority than the features in square number 2, which would be higher priority than the features in square number 3, and so on.

If you find yourself stuck because you're not sure about the estimates for customer value and effort, just use your best guess to place each feature into one of the nine cells. These a re just your starting hypotheses; you can—and likely will—change them as you learn and iterate.

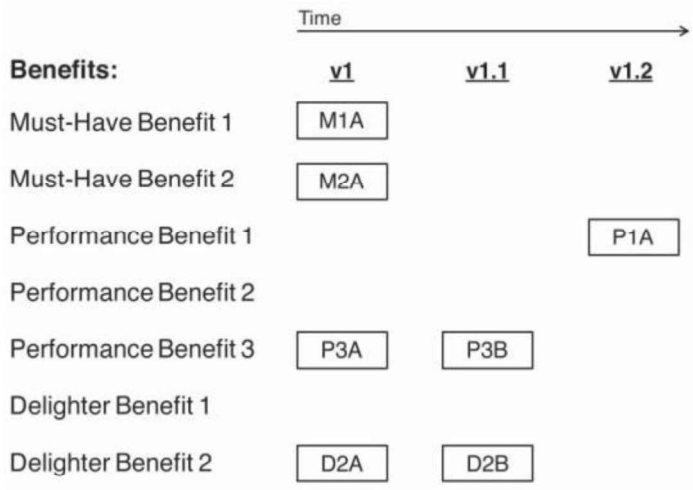
Feature Chunks vs. Benefits



Deciding on the MVP

- Once you have organized your list of feature chunks by benefit and prioritized them, it's time to start making some tough decisions.
  - You must decide on the minimum set of functionality that will resonate with your target customers.
  - You are going to look down the leftmost column of feature chunks and determine which ones you think need to be in your MVP candidate. To start with, your MVP candidate needs to have all the must-haves you've identified.
  - After that, you should focus on the main performance benefit you're planning to use to beat the competition.
  - Delighters are part of your differentiation, too. You should include your top delighter in your MVP candidate.
- The goal is to make sure that your MVP includes *something* that customers find superior to others products and, ideally, unique.

MVP Features by Versions



General Guidelines for MVP

- It is not recommend that you plan more than one or two minor versions ahead at the outset—since a lot of things are apt to change when you show your MVP candidate to customers for the first time.
- You'll learn that some of your hypotheses weren't quite right and will come up with new ones.
- You may end up changing your mind on which benefit is most important or come up with ideas for new features to address the same benefits.
- Your MVP is still just a candidate, a bundle of interrelated hypotheses.
- You need to get customer feedback on your MVP candidate to test those hypotheses.
- So if you've made tentative plans beyond your MVP, you must be prepared to throw them out the window and come up with new plans based on what you learn from customers ☺

Summary

- The Kano Model provides a framework for understanding customer needs and satisfaction developed by quality management expert
- For your minimum viable product (MVP), you want to identify the minimum functionality required to validate that you are heading in the right direction
- User stories offer a helpful way to write down feature ideas.
- ROI helps account for both the effort invested in and value derived from offering different features.

Thank you

Contents

- Validating the Minimum Viable Product (MVP)
  - Will the customer buy this, or choose to use it? (*Value risk*)
  - Can the user figure out how to use it? (*Usability risk*)
  - Can we build it? (*Feasibility risk*)
  - Does this solution work for our business? (*Business viability risk*)

Testing Value

- Customers don't have to buy our products, and users don't have to choose to use a feature. They will only do so if they perceive real *value.*
- So many companies and product teams think all they need to do is match the features (referred to as *feature parity*), and then they don't understand why their product doesn't sell, even at a lower price.
- The customer must perceive your product to be *substantially better* to motivate them to buy your product and then wade through the pain and obstacles of migrating from their old solution.

Just because someone can use our product doesn't mean they will choose to use our product.

Elements of Value

- There are several elements of value, and there are techniques for testing all of them.
  - Testing Demand
  - Testing Value Qualitatively
  - Testing Value Quantitatively

Testing for Different Elements of Value

Testing Demand

Sometimes it's unclear if there's *demand* for what we want to build. In other words, if we could come up with an amazing solution to this problem, do customers even care about this problem? Enough to buy a new product and switch to it? This concept of demand testing applies to entire products, down to a specific feature on an existing product. We can't just assume there's demand, although often the demand is well established because most of the time our products are entering an existing market with demonstrated and measurable demand. The real challenge in that situation is whether we can come up with a demonstrably better solution in terms of value than the alternatives.

Testing Value Qualitatively

The most common type of qualitative value testing is focused on the *response*, or reaction. Do customers love this? Will they pay for it? Will users choose to use this? And most important, if not, why not?

Testing Value Quantitatively

For many products, we need to test *efficacy*, which refers to how well this solution solves the underlying problem. In some types of products, this is very objective and quantitative. For example, in advertising technology, we can measure the revenue generated and easily compare that to other advertising technology alternatives. In other types of products, such as games, it's much less objective.

Qualitative Value Testing Techniques

- If users and customers are not responding to a product the way we had hoped, we need to figure out why that's the case
  - That's why we do qualitative testing.
- Qualitative testing is not about proving anything
- Qualitative testing is about rapid learning and big insights
- When you do this type of qualitative user testing, you don't get your answer from any one user but every user you test with is like another piece of the puzzle.
  - Eventually, you see enough of the puzzle that you can understand where you've gone wrong.
- Qualitative testing of your product ideas with real users and customers is *probably the single most important discovery activity for you and your product team.*
- Teams are advised to do at least two or three qualitative value tests every single week.

Different Qualitative Value Testing Techniques

- Interview First
  - We generally begin the user test with a short user interview where we try to make sure our user has the problems, we think she has, how she solves these problems today, and what it would take for her to switch.
- Usability Test
  - During the usability test, we test to see whether the user can figure out how to operate our product.
  - After a usability test the user knows what your product is all about and how it's meant to be used.
  - This testing involves at least you as product manager and your product designer.
  - But often the *magic* happens when one of your engineers is right there watching the qualitative testing with you.
  - High-fidelity, live-data prototype or a hybrid prototypes can be used for usability testing

Different Qualitative Value Testing Techniques contd.

- Specific Value Tests
  - When you're sitting face to face with actual users and customers is that people are generally nice—and not willing to tell you what they *really* think.
  - So tests for value are designed to make sure the person is *not just being nice to you.*

Specific Value Tests

Using Money to Demonstrate Value

One technique I like for gauging value is to see if the user would be willing to pay for it, even if you have no intention of charging them for it. We're looking for the user to pull out his or her credit card right then and there and ask to buy the product (but we don't really want the card information).

Using Reputation to Demonstrate Value

But there are other ways a user can "pay" for a product. You can see if they would be willing to pay with their reputation. You can ask them how likely they'd be to recommend the product to their friends or co-workers or boss (typically on a scale of 0–10). You can ask them to share on social media. You can ask them to enter the e-mail of their boss or their friends for a recommendation (even though we don't save the e-mails, it's very meaningful if people are willing to provide them).

Using Time to Demonstrate Value

Especially with businesses, you can also ask the person if they'd be willing to schedule some significant time with you to work on this (even if we don't need it). This is another way people pay for value.

Using Access to Demonstrate Value

You can also ask people to provide the login credentials for whatever product they would be switching from (because you tell them there's a migration utility or something). Again, we don't really want their login and password—we just want to know if they value our product highly enough that they're truly willing to switch right then and there.

Iterating the Prototype

- Remember, this is not about proving anything. It's about rapid learning. As soon as you believe you have a problem, or you want to try a different approach, you try it.
- If you show your prototype to two different people and the response you get is substantially different
  - Your job is to try to figure out why
- You might determine that you just aren't able to get people interested in this problem, or you can't figure out a way to make this usable enough that your target users can realize this value.
  - In that case, you may decide to stop right there and put the idea on the shelf
  - Some product managers consider this a big failure. It can instead be viewed it as saving the company wasted cost and effort

Iterating the Prototype contd.

- The remarkable thing about this kind of qualitative testing is just how easy and effective it is.
- The best way to prove this to yourself is to take your laptop or mobile device with your product or prototype on it to someone who hasn't seen it yet, and just give it a try.

As product manager, you need to make sure you are at every single qualitative value test. Do not delegate this.

Quantitative Value Testing Techniques

- While qualitative testing is all about fast learning and big insights, quantitative techniques are all about collecting evidence.
- We will sometimes collect enough data that we have *statistically significant results* (especially with consumer services with a lot of daily traffic)
- At other times we'll set the bar lower and just collect actual usage data that we consider useful *evidence*—along with other factors—to make an informed decision.

While qualitative testing is all about fast learning and big insights, quantitative techniques are all about collecting evidence.

Quantitative Value Testing Techniques contd.

- A/B Testing
  - The gold standard for this type of testing is an A/B test.
  - The reason we love A/B tests is because the user doesn't know which version of the product she is seeing.
  - This yields data that is very predictive, which is what we ideally want.
- Invite-Only Testing
  - Used when your company is more risk averse, or if you just don't have enough traffic to be able to show to 1 percent—or even 10 percent—and get useful results anytime soon
  - This is where you identify a set of users or customers that you contact and invite to try the new version.
  - You tell them that it is an experimental version, so they are effectively opting in if they choose to run it.
  - The data that this group generates is not as predictive as from a true, blind, A/B test.

Quantitative Value Testing Techniques contd.

- Customer Discovery Program
  - A variation of the invite-only test is to use the members of the customer discovery program we discussed in the section on ideation techniques.
  - These companies have already opted in to testing new versions, and you already have a close relationship with them so you can follow up with them easily.

Testing Usability

- It is important to do usability testing in discovery—using prototypes, before we build the product—and not at the end, where it's really too late to correct the issues without significant waste or worse. The key points in the process are:
  - Recruiting Users to Test
  - Preparing the Test
  - Testing Your Prototype
  - Summarizing the Learning

Recruiting Users to Test

- You'll need to round up some test subjects. If you're using a user research group, they'll likely recruit and schedule the users for you, which is a huge help, but if you're on your own, you've got several options:
  - If you've established the customer-discovery program, you can leverage that group; If you're working on a consumer product, you'll want to supplement that group.
  - You can advertise for test subjects in the mainstream or digital media
  - If you have a list of e-mail addresses of your users, you can do a selection from there.
  - You can solicit volunteers on your company website; that you'll still need to call and screen the volunteers to make sure the people you select are in your target market
  - You can always go to where your users congregate, such as trade shows.
  - If you're asking users to come to your location, you will likely need to compensate them for their time. We often will arrange to meet the test subject at a mutually convenient location, such as a Starbucks; its commonly called *Starbucks testing*

Preparing the Test

- We usually do usability testing with a *high-fidelity user prototype*
- Most of the time, when we do a usability and/or value test, it's with the product manager, the product designer, and one of the engineers from the team
- You will need to define in advance the set of tasks that you want to test.
  - Usually, these are fairly obvious. If, for example, you're building an alarm clock app for a mobile device, your users will need to do things like set an alarm, find and hit the snooze button, and so on. There may also be more obscure tasks, but concentrate on the primary tasks—the ones that users will do most of the time.
  - Some people still believe that the product manager and the product designer are too close to the product to do this type of testing objectively, and they may either get their feelings hurt or only hear what they want to hear. So, first, we train the product managers and designers on how to conduct themselves, and second, we make sure the test happens quickly—before they fall in love with their own ideas.

Preparing the Test contd.

- You should have one person administer the usability test and another person taking notes. It's helpful to have at least one other person to debrief with afterward to make sure you both saw the same things and came to the same conclusions.
- Formal testing labs will typically have setups with two-way mirrors or closed-circuit video monitors with cameras that capture both the screen and the user from the front. Informal testing at a Starbucks table is also fine.
- The other environment that works really well is your customer's office. It can be time consuming to do, but even 30 minutes in their office can tell you a lot.
- There are tools for doing this type of testing remotely, and I encourage that, but they are primarily designed for usability testing and not for the value testing that will usually follow.
  - Thus remote usability testing as a supplement rather than a replacement.

Testing Your Prototype

We want to learn whether the user or customer really has the problems we think they have, and how they solve those problems today, and what it would take for them to switch.

Tips for Administering the Actual Test

- When you first start the actual usability test, make sure to tell your subject that this is just a prototype, it's a very early product idea, and it's not real.
- See if they can tell from the landing page of your prototype what it is that you do, and especially what might be valuable or appealing to them
- When testing, you'll want to do everything you can to keep your users in *use mode* and out of *critique mode*.
- During the testing, the main skill you have to learn is to keep quiet.
- There are three important cases you're looking for: (1) the user got through the task with no problem at all and no help; (2) the user struggled and moaned a bit, but he eventually got through it; or (3) he got so frustrated he gave up.
- In general, you'll want to avoid giving any help or *leading the witness* in any way.
- Act like a parrot. This helps in many ways. First, it helps avoid leading. If they're quiet and you really can't stand it because you're uncomfortable, tell them what they're doing: "I see that you're looking at the list on the right."
- Fundamentally, you're trying to get an understanding of how your target users think about this problem and to identify places in your prototype where the model the software presents is inconsistent or incompatible with how the user is thinking about the problem.
- You will find that you can tell a great deal from body language and tone. It's painfully obvious when they don't like your ideas, and it's also clear when they genuinely do.

Tips for Administering the Actual Test

- When you first start the actual usability test, make sure to tell your subject that this is just a prototype, it's a very early product idea, and it's not real.
- See if they can tell fr… …t it is that you do, and especially what migh…
- When testing, you'll v… …users in *use mode* and out of *critique mode*.
- During the testing, th… …iet.
- There are three impo… …got through the task with no problem at all and … …d a bit, but he eventually got through it; or (3) h…
- In general, you'll wan… …*tness* in any way.
- Act like a parrot. This… …ading. If they're quiet and you really can't stand … …what they're doing: "I see that you're looking at …
- Fundamentally, you'r… …ur target users think about this problem and to identify places in your prototype where the model the software presents is inconsistent or incompatible with how the user is thinking about the problem.
- You will find that you can tell a great deal from body language and tone. It's painfully obvious when they don't like your ideas, and it's also clear when they genuinely do.

The point is to gain a deeper understanding of your users and customers and, of course, to identify the friction points in the prototype so you can fix them.

Testing Feasibility

- When we talk about validating feasibility, the engineers are really trying to answer several related questions:
  - Do we know how to build this?
  - Do we have the skills on the team to build this?
  - Do we have enough time to build this?
  - Do we need any architectural changes to build this?
  - Do we have on hand all the components we need to build this?
  - Do we understand the dependencies involved in building this?
  - Will the performance be acceptable?
  - Will it scale to the levels we need?
  - Do we have the infrastructure necessary to test and run this?
  - Can we afford the cost to provision this?

Testing Feasibility contd.

- With most product ideas that your engineers review in discovery, they will quickly consider these points and simply say “No problem.”
- That’s because most of our work is not all that new, and engineers have usually built similar things many times before.
- However, there are definitely ideas where this is not the case, and some or many of these questions can be very difficult for the engineers to answer.
  - One very common example right now is that many teams are evaluating machine-learning technology, considering build/buy decisions, and assessing whether the technology is suitable for the job at hand—and, more generally, trying to understand its potential.
- The question to engineers isn’t, “Can you do this?” Rather, you are asking them to look into it and answer the question, “What’s the best way to do this and how long would it take?”
- The engineers will sometimes come back and say they need to create a *feasibility prototype* to answer one or more of these questions.
  - First consider whether the idea is potentially worth investing the necessary time in discovery
  - If so, then encourage the engineers to proceed.

Testing Feasibility contd.

Many of our best product ideas are based on approaches to solving the problem that are only now possible, *which means new technology and time to investigate and learn that technology.*

Testing Business Viability

- It’s hard enough just trying to come up with a product that your customers love and your engineers can build and deliver.
  - Many products never get to this point.
- However, the truth is this is not enough.
  - The solution must also *work for your business.*
  - This is often the least favorite part of a product manager’s job.
  - But this is often what separates the good product managers from the great ones
  - And this is really what is meant by being the CEO of the product
- Building a business is always hard.
  - You must have a business model that’s viable. The costs to produce, market and sell your product must be sufficiently less than the revenue your product generates. You must operate within the laws of the countries you sell in. You must hold up your end of business agreements and partnerships. Your product must fit within the brand promise of your company’s other offerings.

Testing Business Viability contd.

- Here are the main stakeholders in a tech-powered product company
  - Marketing
  - Sales
  - Customer Service
  - Finance
  - Legal
  - Business Development
  - Security
  - CEO/COO/GM

Summary

- The MVP has to be tested along multiple dimensions addressing the following:
  - Value risk
  - Usability risk
  - Feasibility risk
  - Business viability risk

Thank you

Contents

- Building your product using Agile
  - Agile methods: Principles, user stories, epics, product backlog

Motivation for Agile

- Traditional heavyweight methods can be overly regulated, planned, and micromanaged, i.e. bureaucratic
- In reaction, a number of lightweight methods started evolving that collectively became known as “agile”, emphasizing:

Individuals and interactions	**over**	processes and tools
Working software		comprehensive documentation
Customer collaboration		contract negotiation
Responding to change		following a plan

"while there is value in the items on the right, we value the items on the left more"

Agile: Key Characteristics

Highly iterative and flexible

Working software delivered frequently

Working software (not “WIP”) is the measure of progress

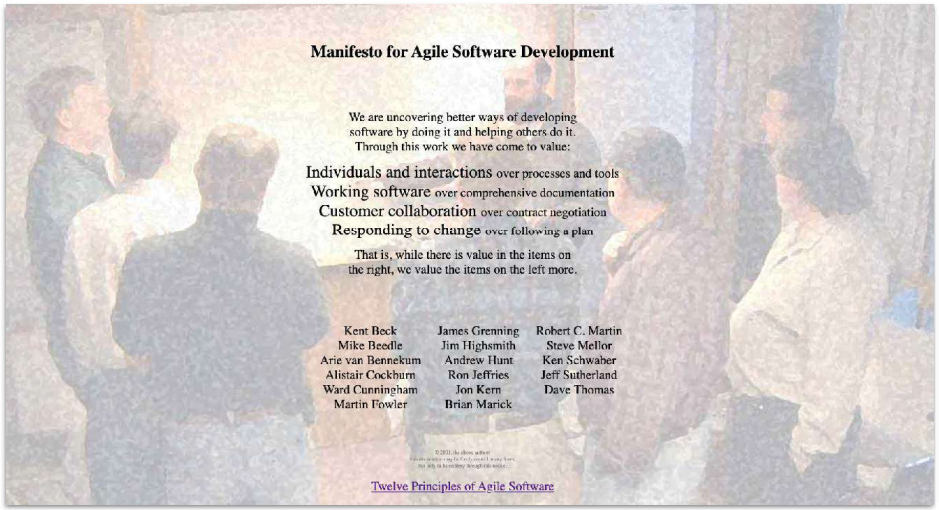
Agile – Key Characteristics

Late changes in requirements – no problem

Close, daily co-operation

Visualize activities and “WIP” to manage team’s capacity

The Agile Manifesto (2001)



https://agilemanifesto.org/

Agile Principles (*reorganized*; Meyer 2014)

Agile principles

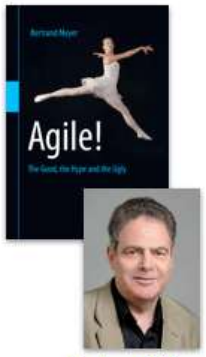
Organizational

1 Put the customer at the center.
2 Let the team self-organize.
3 Work at a sustainable pace.
4 Develop minimal software:
  4.1 Produce minimal functionality.
  4.2 Produce only the product requested.
  4.3 Develop only code and tests.
5 Accept change.

Technical

key insight

6 Develop iteratively:
  6.1 Produce frequent working iterations.
  6.2 Freeze requirements during iterations.
7 Treat tests as a key resource:
  7.1 Do not start any new development until all tests pass.
  7.2 Test first.
8 Express requirements through scenarios.



Bertrand Meyer

What is a User Story?

- A User Story is a short, simple description of a feature told from the viewpoint of a person who desires the new capability, usually a customer or user.
- It typically follows the form:
- As a <user role>, I want <some goal> for <some reason>

Origins

1998 Alistair Cockburn
A user story is a promise for a conversation

1999 Kent Beck, *Extreme Programming Explained*
Story – one thing the customer wants the system to do

2004 Mike Cohn, *User Stories Applied**
A User Story describes functionality that will be valuable to either a user or purchaser of a system or software

2014 Jeff Patton, *User Story Mapping*
Story Mapping is a technique that provides the big picture that a pile of stories so often misses

User Story "Card"

Title View My Bills	**Priority**	**Estimate**
User Story As a patient, I want to see all my bills sorted on date and institution, so I can check I paid correctly		
Acceptance Criteria		

Card
Conversation
Confirmation

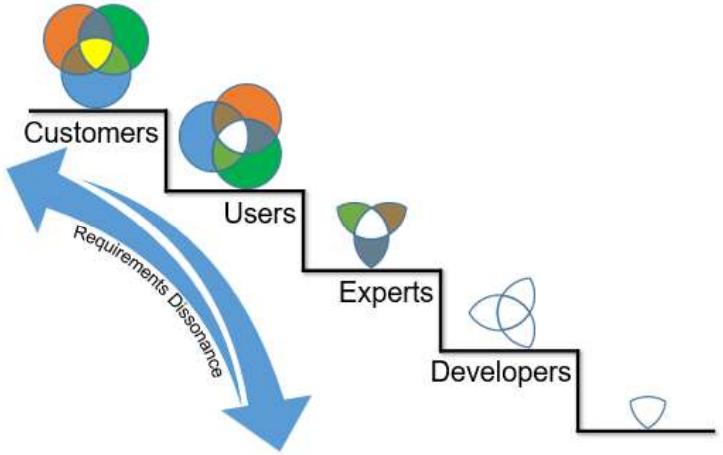
Details as Sub-Stories

Title View My Bills		
User Story As a patient, I want to see all my bills sorted on date and institution, so I can check I paid correctly		
Acceptance Criteria • I see only my bills • I see all my bills • I see bills chronologically • *I see bills from selected institutions*		

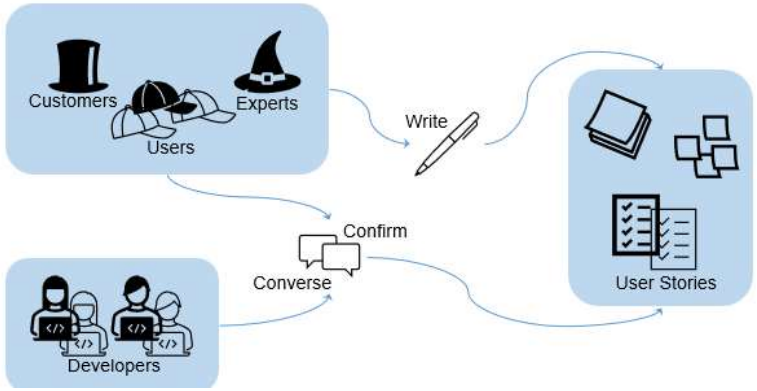
Title View My Unpaid Bills	**Priority**	**Estimate**
User Story As a patient, I want to see all my unpaid bills, so I can check w		
Accepta		

Title Alert on Unpaid Bills	**Priority**	**Estimate**
User Story As a patient, I want to be alerted to all my unpaid bills, so I can pay punctually		
Acceptance Criteria		

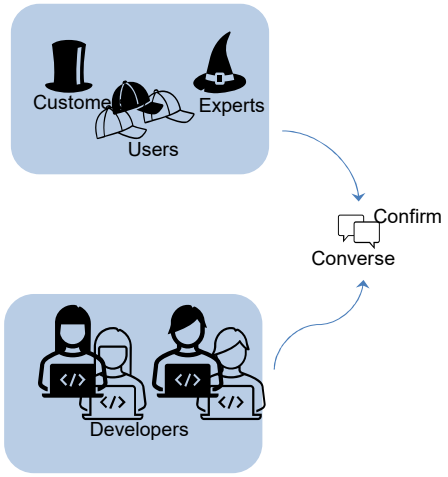
Requirements: Different Voices



Who Does What



Why Converse?



Mike Cohn:

'Software requirements is a communication problem…'

'When the business side dominates, it mandates functionality and dates with little concern…'

'When the developers dominate… technical jargon replaces the language of the business… developers lose the opportunity to learn what is needed by listening…'

'If either side dominates these communications, the project loses…'

'User stories provide…just enough written down that we don't forget and that we can estimate and plan…'

How do User Stories Induce Conversations?

Roles

- There is no one 'User'
- There are many 'Roles' each with their own needs
  - What are these roles?
  - Name the roles
  - Group them based on similar pattern of use
  - Rename intuitively as needed
  - Use these roles to enumerate user stories

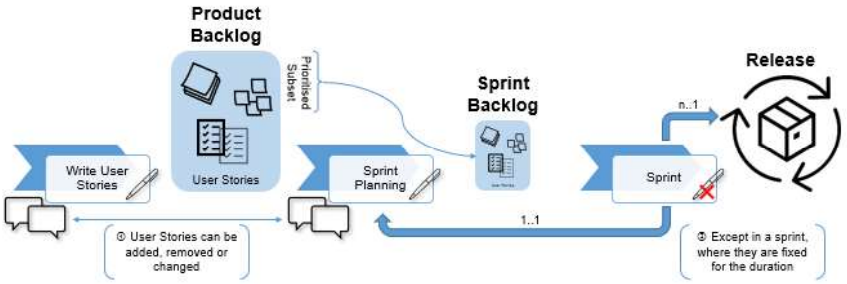
User Stories

- User Stories are not all obvious or of the same size
- They can be apparent at different moments, be of different size and will change with time
- The aim is to capture as many as apparent with the thought that there are more
- As with traditional business analysis, various techniques are useful to trawl for user stories
  - **Questionnaire –** for a start with large user base
  - **Observation –** insights from visible/invisible observation and/or user demonstration
  - **User Interview –** structured, focused, non-leading interview of curated users
  - **Story-writing Workshop –** whole team lo-fidelity brainstorming, prototyping, role-playing, etc.

User Stories, Epics, and Themes

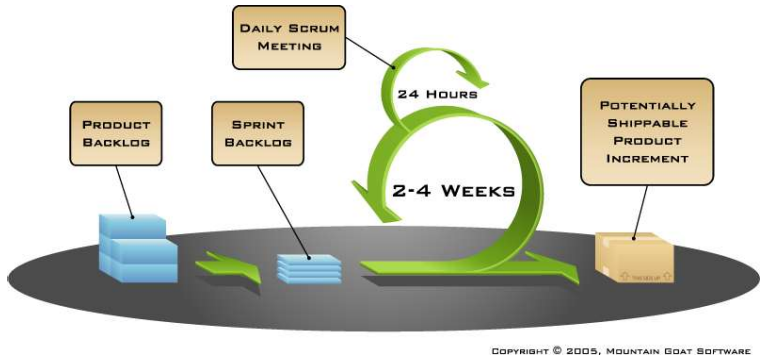
- Although in general, we want to estimate user stories whose sizes are within one order of magnitude, this cannot always be the case.
- If we are to estimate everything within one order of magnitude, it would mean writing all stories at a fairly fine-grained level.
- For features that we're not sure we want (a preliminary cost estimate is desired before too much investment is put into them) or for features that may not happen in the near future, it is often desirable to write one much larger user story.
  - A large user story is sometimes called an **epic**
- Additionally, a set of related user stories may be combined (usually by a paper clip if working with note cards) and treated as a single entity for either estimating or release planning.
  - Such a set of user stories is referred to as a **theme**
- An epic, by its very size alone, is often a theme on its own.

Where do User Stories End Up?



The Idea of a "Sprint"

- Sprints are fixed-length (e.g. 2-4 weeks) iterations that aim to achieve some agreed-upon outcome
  - e.g. some software functionality – tested, integrated, documented
- Requirements are frozen for the length of a sprint



Product Backlog

- Before the first Sprint can begin, there needs to be a Product Backlog
- At any point, the Product Backlog is the single, definitive view of 'everything that could be done by the Team—ever—in order of priority'
- Only a single Product Backlog exists for a product
- The Product Backlog exists and evolves over the lifetime of a product

Characteristics of a Good Product Backlog

- A good Product Backlog is **DEEP**
  - **Detailed appropriately –** the top priority items are more fine-grained and detailed than the lower priority items, since the former will be worked on sooner than the latter. For example, the top 10% of the backlog may be composed of very small, well-analysed items, and the other 90% much less so
  - **Estimated –** the items for the current release need to have estimates, and furthermore, should be considered for re-estimation each Sprint as everyone learns and new information arises. The Team provides the Product Owner with effort estimates for each item on the Product Backlog, and perhaps also technical risk estimates. The Product Owner and other business stakeholders provide information on the value of the product requests, which may include revenue gained, costs reduced, business risks, importance to various stakeholders, and more
  - **Emergent –** in response to learning and variability, the Product Backlog is regularly refined. Each Sprint, items may be added, removed, modified, split, and changed in priority. Thus, the Product Backlog is continuously updated by the Product Owner to reflect changes in the needs of the customer, new ideas or insights, moves by the competition, technical hurdles that appear, and so forth
  - **Prioritized –** the items at the top of the Product Backlog are prioritized or ordered in a 1-N order. In general, the highest-priority items should deliver the most bang for your buck: lots of bang (business value) for low buck (cost). Another motivation to increase the priority of an item is to tackle high risks early, before the risks attack you

Characteristics of Good User Stories

Independent

Negotiable

Valuable

Estimable

Small

Testable

A good story has the six attributes.

Bill Wake, *Extreme Programming Explored*, coined the acronym for ease of recall.

Independent

Independent	Ensure User Stories are as independent as possible. Each User Story can then be estimated and selected for inclusion in a Sprint without regard for any other User Stories.
Negotiable	Not all User Stories can easily be made independent, e.g., • *As a patient, I want to see all my bills sorted on date and institution...* • *As a billing executive, I want to create bills showing different dates for admitting, discharging, billing, invoicing, reminding and paying...*
Valuable	Are they related? Which comes first? What is the date of relevance to the patient?
Estimable	
Small	
Testable	More inter-dependent User Stories makes estimation and planning more complicated.

Negotiable

Independent
Negotiable
Valuable
Estimable
Small
Testable

User stories are not cast in stone until a Sprint begins. At that point, all the User Stories selected for the Sprint are fixed until completion of the Sprint.

All other User Stories are negotiable, e.g.,

- Does the User Story fulfil the reason for its existence?
- Is it too small and needs to be combined with another User Story?
- Is it too big and needs to be broken further?
- Are the acceptance criteria sufficient?

Valuable

Independent
Negotiable
Valuable
Estimable
Small
Testable

'Valuable' here is from the viewpoint of the customers and users.

A User Story takes the form:

As a <user role>, I want <some goal> for <some reason>

The 'goal' is the capability/functionality to be created. The 'reason' is the value that will be created using the capability/functionality, e.g.,

As a patient, I want to see all my bills sorted on date and institution, so I can check I paid correctly

'I can check I paid correctly' is of value to the patient.

Estima(ta)ble

Independent
Negotiable
Valuable
Estimable
Small
Testable

User Stories should be clear enough so that developers can reasonably state the effort required to fulfil them. When might User Stories be unclear:

- Developers lack domain knowledge
  - ➢ Converse with the User to the level needed to make an estimate; stop there, this is not the moment to converse on how to create the capability/functionality
- Developers lack technical knowledge, e.g.,

  As a security officer, I want communications to be protected by TLS, so that sensitive information cannot be intercepted
  - ➢ Take a time-out to investigate TLS, but strictly within a timebound
- The User Story is too big, e.g.,

  As a student, I want to communicate with the instructor to schedule a consult, to clarify doubts
  - ➢ Break it up to bite-size

  As a student, I want to email the instructor to schedule a consult...

  As a student, I want to Telegram the instructor to schedule a consult...

  As a student, I want to Teams Chat the instructor to schedule a consult...

Small

Independent
Negotiable
Valuable
Estimable
Small
Testable

Think Goldilocks, not too small, not too big, just the right size for planning.

- If it is too big, it cannot fit within a Sprint, e.g.,

  As a patient, I want so see all information the hospital has of me...
  - ➢ Break it up to bite-size

  As a patient, I want to see all my bills...

  As a patient, I want to see all my medical records...
- If it is too small, chances are that an original User Story has been broken down to such granular User Stories that inter-dependency makes planning complicated, e.g.,

  As a patient, I want to see all government subsidies...

  As a patient, I want to see all ward charges line items...
  - ➢ Club them up to modular-size

Testable

Independent	'If the story cannot be tested, how can the developers know when they have finished coding?' – Mike Cohn, *User Stories Applied*
Negotiable	There must be a clear set of acceptance criteria for what success looks like and the means to test the criteria: • The screen must not take too long to appear – it is untestable; how long is long, one second, two seconds, etc. • The screen must appear within 2 seconds – this is testable
Valuable	Every user story is negotiable; things change, possibly constantly and quickly. Therefore, it is important that testing be as automated as possible to catch knock-on effects of changes as User Stories are coded:
Estimable	• The app must be intuitive to use – maybe a test to observe a user click once through the app suggesting intuitive and clicking multiple times suggesting non-intuitive is possible, but it can hardly be automated
Small	
Testable	

Summary

- Agile methods are particularly suited for software product development
- Agile methods involve detailed interactions between different stakeholders
- User stories are a key driver of agile methods

Thank you

Contents

- Building your product using Agile contd.
  - Story points
  - Velocity
  - Planning poker
  - Release planning
  - Continuous integration and delivery
  - Version control

Software Estimation

- Software estimates involve …
  - calculations
  - judgement
  - appraisal
- … and may result in promises based on factors such as
  - size
  - effort
  - time
- Estimation is the process by which estimates are arrived at

Agile Estimation

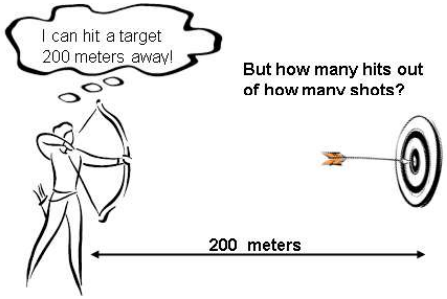
“ … estimating and planning processes must themselves be agile. Without agile estimating and planning, we cannot have agile projects.” – Mike Cohn

Understanding Estimates

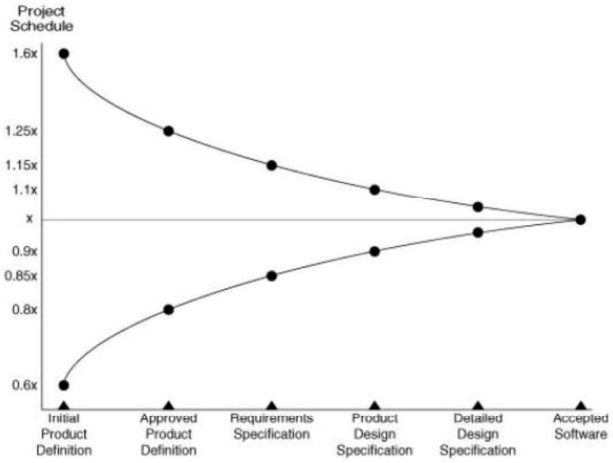
- "The project will require 70 person-months of effort"
- Does this statement mean it can be said with 100% surety that the project will require 70 person-months of effort?
- Given the uncertainties inherent in every real-life scenario, this seems too audacious a claim!
- So if the level of surety is not 100%, what is it?

Single point estimates

- In all 'single-point' estimates, the level of certainty of the statement being true is often not 1 (absolute certainty)
- What that level of certainty is needs to be clearly understood and expressed in either of the following ways:
  - The estimator is 80% confident that the project will need 70 person-months
  - The project will at the most need 90 person-months and at the least need 60 person-months

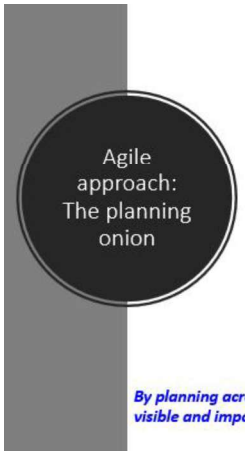


The Cone of Uncertainty



Agile Estimating and Planning by Mike Cohn, 2006. Pearson Education. Kindle Edition.

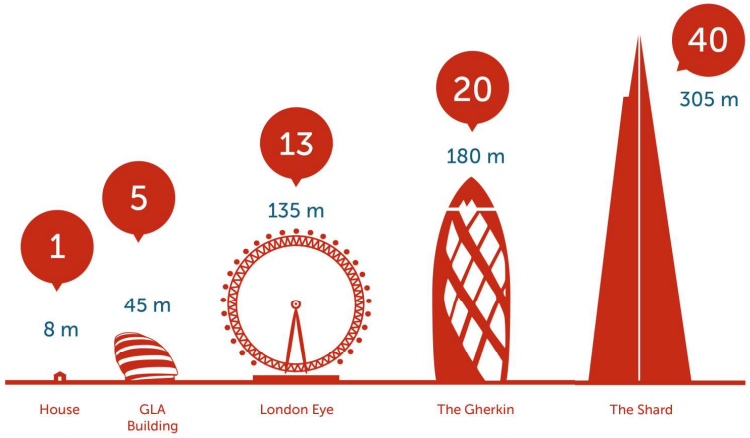
Planning



By planning across at least three time-horizons—release, iteration, and day—agile teams focus on what is visible and important to the plan they are creating.

Agile Estimating and Planning by Mike Cohn, 2006. Pearson Education. Kindle Edition.

Estimating with Story Points



https://www.zentao.pm/agile-knowledge-share/story-point-track-the-progress-of-story-completion-306.mhtml

Estimating size with story points: Key themes

- Story points are a unit of measure for expressing the overall size of a user story, feature, or other piece of work
- When we estimate with story points, we assign a point value to each item
  - The raw values we assign are unimportant; what matters are the relative values
  - A story that is assigned a two (2) point value should be twice as much as a story that is assigned a one (1) point value; it should also be two-thirds of a story that is estimated as three (3) story points

Number of story points

- The number of story points associated with a story represents the overall size of the story
- There is no set formula for defining the size of a story
- A story-point estimate is an amalgamation of the amount of effort involved in developing the feature, the complexity of developing it, the risk inherent in it etc.

Estimating size with story points

- Either
  - Select a story that you expect to be one of the smallest stories will be worked with and say that story is estimated at one story point, or
  - Select a story that seems somewhat medium and give it a number somewhere in the middle of the range you expect
- Once a story-point value is arbitrarily assigned to the first story
  - Each additional story is estimated by comparing it with the first story or with any others that have been estimated
- On an agile project it is not uncommon to begin an iteration with incompletely specified requirements, the details of which will be discovered during the iteration
  - However, we need to associate an estimate with each story, even those that are incompletely defined
- When a loosely defined user story is encountered, need to make some assumptions, take a guess, and move on

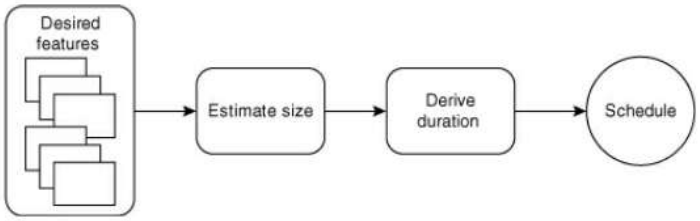
Velocity

- The concept of *velocity* allows us to understand how estimating in unitless story points can possibly work
- Velocity is a measure of a team's rate of progress
- Velocity is calculated by summing the number of story points assigned to each user story that the team completed during the iteration

Using velocity in estimation

- If the team completes three stories each estimated at five story points, their velocity is 15
- If the team completes two five-point stories, their velocity is 10
- If a team completed 10 story points of work last iteration, our best guess is that they will complete 10 story points this iteration
  - Since story points are estimates of relative size, this will be true whether they work on two five-point stories or five two-point stories!

Estimating duration begins with estimating size



If we sum the story-point estimates for all desired features we come up with a total size estimate for the project. If we know the team's velocity we can divide size by velocity to arrive at an estimated number of iterations. We can turn this duration into a schedule by mapping it onto a calendar.

A key tenet of agile estimating and planning is that we estimate size but derive duration.

Agile Estimating and Planning by Mike Cohn, 2006. Pearson Education. Kindle Edition.

Velocity corrects estimation errors

- In points-based approach to estimating planning errors are self-correcting because of the application of velocity
- An example
  - Suppose a team estimates a project to include 200 points of work
  - They initially believe they will be able to complete 25 points per iteration, which means they will finish in eight iterations
  - However, once the project begins, their observed velocity is only 20
  - Without re-estimating any work they will have correctly identified that the project will take 10 iterations rather than eight

Separation of effort and duration

- Estimating in story points completely separates the estimation of effort from the estimation of duration
- Effort and schedule are obviously related, but separating them allows each to be considered independently
- Using story points and velocity, duration is not even being estimated, it is being derived or computed
  - The distinction is subtle, but significant

Estimating in ideal days

- *Ideal time* is the amount of time that something takes when stripped of all peripheral activities
- *Elapsed time,* on the other hand, is the amount of time that passes on a clock (or perhaps a calendar)
- It is almost always far easier and more accurate to predict the duration of an event in ideal time than in elapsed time
- In a software project, ideal time differs from elapsed time because of the natural overhead we experience every day
  - Answering email, making a support call to a vendor, interviewing a candidate, and attending meetings …
  - Multitasking on project activities

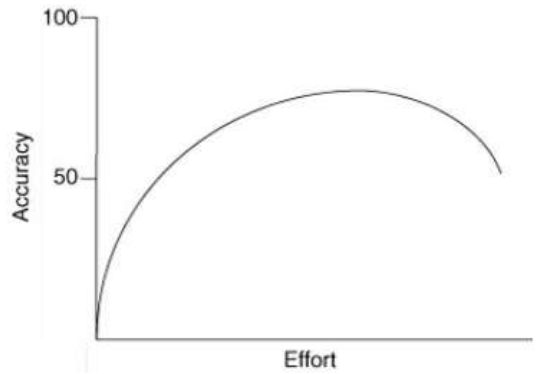
Ideal days as a measure of size

- User stories can be estimated in *ideal days* under the assumptions:
  - The story being estimated is the only thing that will be worked on
  - Everything needed will be on hand when you work is started
  - There will be no interruptions
- When we estimate the number of ideal days that a user story will take to develop, test, and accept, it is not necessary to consider the impact of the overhead of the environment in which the team works
  - If developing a particular screen will take a developer one ideal day, it will take him/her one ideal day regardless of whether the developer is employed by a start-up with no overhead or other demands on his/her time or by a huge bureaucracy

Estimating size in ideal days

- Ideal days can be thought of as another estimate of size, just as story points are only when considerations of organizational overhead are ignored
- Then an estimate of size expressed as a number of ideal days can be converted into an estimate of duration using velocity in exactly the same way as with story points

Estimation: Effort vs accuracy trade-off



Beyond a certain point, more effort invested in estimation will not lead to more accurate estimates!

Agile Estimating and Planning by Mike Cohn, 2006. Pearson Education. Kindle Edition

Planning poker: Overview

- Planning poker combines expert opinion, analogy, and disaggregation into an enjoyable approach to estimating, that results in quick but reliable estimates
- Participants in planning poker include all of the developers on the team
- "Developers" refers to all programmers, testers, database engineers, analysts, user interaction designers, and so on.
- On an agile project, this will typically not exceed 10 people
  - If it does, it is usually best to split into two teams; each team can then estimate independently, which will keep the size down
- The product owner participates in planning poker but does not estimate

Planning Poker: Steps

1. Each estimator is given a deck of cards that reads 0, 1, 2, 3, 5, 8, 13, 20, 40, and 100.
2. For each user story to be estimated, a moderator reads the description and answers any questions that the estimators have.
3. Each estimator privately selects a card representing his or her estimate.
   - Cards are not shown until each estimator has made a selection.
   - At that time, all cards are simultaneously turned over and shown so that all participants can see each estimate.
4. If estimates differ, the high and low estimators explain their estimates and discuss them among the team.
5. After the discussion, each estimator re-estimates by selecting a card.
6. Steps 3 - 5 are repeated until the estimates converge.

Planning poker: Detailed process

"At the start of planning poker, each estimator is given a deck of cards. Each card has written on it one of the valid estimates. Each estimator may, for example, be given a deck of cards that reads 0, 1, 2, 3, 5, 8, 13, 20, 40, and 100. The cards should be prepared prior to the planning poker meeting, and the numbers should be large enough to see across a table. Cards can be saved and used for the next planning poker session. For each user story or theme to be estimated, a moderator reads the description. The moderator is usually the product owner or an analyst. However, the moderator can be anyone, as there is no special privilege associated with the role. The product owner answers any questions that the estimators have. However, everyone is asked to remain aware of the effort/accuracy curve. The goal in planning poker is not to derive an estimate that will withstand all future scrutiny. Rather, the goal is to be somewhere well on the left of the effort line, where a valuable estimate can be arrived at cheaply. After all questions are answered, each estimator privately selects a card representing his or her estimate."

Agile Estimating and Planning by Mike Cohn, 2006. Pearson Education. Kindle Edition.

Planning poker: Detailed process (contd.)

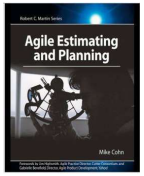
"Cards are not shown until each estimator has made a selection. At that time, all cards are simultaneously turned over and shown so that all participants can see each estimate. It is very likely at this point that the estimates will differ significantly. This is actually good news. If estimates differ, the high and low estimators explain their estimates. It's important that this does not come across as attacking those estimators. Instead, you want to learn what they were thinking about.

As an example, the high estimator may say, "Well, to test this story, we need to create a mock database object. That might take us a day. Also, I'm not sure if our standard compression algorithm will work, and we may need to write one that is more memory efficient." The low estimator may respond, "I was thinking we'd store that information in an XML file—that would be easier than a database for us. Also, I didn't think about having more data—maybe that will be a problem.""

Agile Estimating and Planning by Mike Cohn, 2006. Pearson Education. Kindle Edition.

Planning poker: Detailed process (contd.)

"The group can discuss the story and their estimates for a few more minutes. The moderator can take any notes she thinks will be helpful when this story is being programmed and tested. After the discussion, each estimator re-estimates by selecting a card. Cards are once again kept private until everyone has estimated, at which point they are turned over at the same time. In many cases, the estimates will already converge by the second round. But if they have not, continue to repeat the process. The goal is for the estimators to converge on a single estimate that can be used for the story. It rarely takes more than three rounds, but continue the process as long as estimates are moving closer together."



Agile Estimating and Planning by Mike Cohn, 2006. Pearson Education. Kindle Edition.

The right amount of discussion!

- Some amount of preliminary design discussion is necessary and appropriate when estimating
- However, spending too much time on design discussions sends a team too far up the effort/accuracy curve
- An effective way to encourage some amount of discussion but make sure that it does not go on too long is to:
  - Place a two-minute sand timer in the middle of the table where planning poker is being played.
  - Anyone in the meeting can turn the timer over at any time
  - When the sand runs out (in two minutes), the next round of cards is played
  - If agreement isn't reached, the discussion can continue
  - But someone can immediately turn the timer over, again limiting the discussion to two minutes. The timer rarely needs to be turned over more than twice
  - Over time this helps teams learn to estimate more rapidly

Smaller sessions

- It is possible to play planning poker with a subset of the team, rather than involving everyone
  - This is not ideal but may be a reasonable option, especially if there are many, many items to be estimated, as can happen at the start of a new project
- The best way to do this is to split the larger team into two or three smaller teams, each of which must have at least three estimators
  - It is important that each of the teams estimates consistently
  - What your team calls three story points or ideal days had better be consistent with what my team calls the same
- To achieve this, start all teams together in a joint planning poker session for an hour or so
  - Have them estimate ten to twenty stories
  - Then make sure each team has a copy of these stories and their estimates and that they use them as baselines for estimating the stories they are given to estimate

When to play planning poker

- Teams will need to play planning poker at two different times
- First, there will usually be an effort to estimate a large number of items before the project officially begins or during its first iterations
  - Estimating an initial set of user stories may take a team two or three meetings of from one to three hours each
  - Naturally, this will depend on how many items there are to estimate, the size of the team, and the product owner's ability to clarify the requirements succinctly.
- Second, teams will need to put forth some ongoing effort to estimate any new stories that are identified during an iteration
  - One way to do this is to plan to hold a very short estimation meeting near the end of each iteration
  - Normally, this is quite sufficient for estimating any work that came in during the iteration, and it allows new work to be considered in the prioritization of the coming iteration

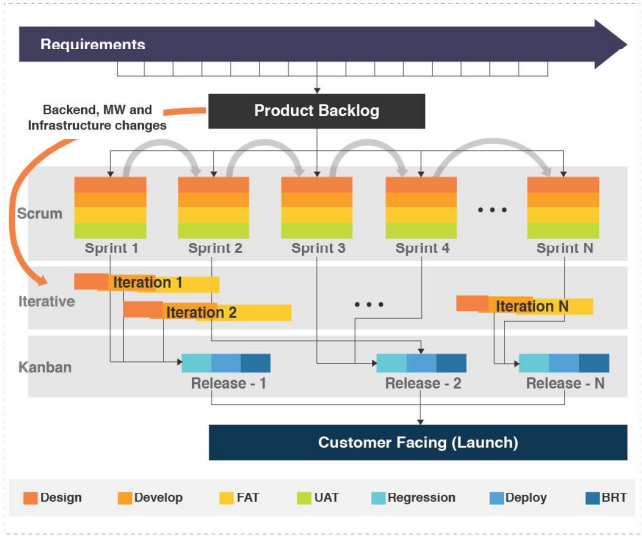
Why planning poker works

- First, planning poker brings together multiple expert opinions to do the estimating
  - These experts form a cross-functional team from all disciplines on a software project
  - They are better suited to the estimation task than anyone else
  - "The people most competent in solving the task should estimate it." - Jørgensen
- Second, a lively dialogue ensues during planning poker, and estimators are called upon by their peers to justify their estimates
- Finally, planning poker works because it's fun ☺

Release Planning

- Release planning is the creation process of a very high-level plan for multiple Sprints (e.g. three to twelve iteration).
  - It is a guideline that reflects expectations about which features will be implemented and when they are completed.
  - It also serves as a base to monitor progress within the project.
  - Releases can be intermediate deliveries done during the project or the final delivery at the end.
- To create a Release Plan the following things have to be available:
  - A prioritized and estimated Product Backlog
  - The (estimated) velocity of the team
  - Conditions of satisfaction

Release Planning contd.



https://www.scrum-institute.org/Release_Planning.php

Continuous Integration and Delivery, Version Control

- Many product teams use *continuous integration* to iterate their product development more quickly.
- Development teams use a *version control system* to keep track of every single revision made to the code; this makes it easy to see and manage changes.
- Version control also simplifies the process of restoring the code base to any prior state, so unwanted changes can be reverted.
  - Git is arguably the most popular version control system for Agile development.
- When developers make changes or additions, they start with the current, stable version of the code base, called the *mainline* or *trunk*.
  - Version control lets developers start with separate copies of the trunk (called branches), that they can modify without affecting the trunk.
  - When developers are done building new functionality, they commit their changes to the version control system.
  - Before doing so, each developer should perform *unit testing* of his or her code by writing the relevant test cases and ensuring they all pass.

Continuous Integration and Delivery, Version Control contd.

- A team of developers all work in parallel, each committing their changes.
- Before merging the new code with the trunk and releasing it, all the changes are combined or “integrated” to build the new version of the whole product.
- *Integration testing* is performed at this point to ensure that the new product works as intended.
- Historically, integration has typically been a manual process.
- Continuous integration uses an automated build process to create a new version of the product based on the latest code commits.
- The new build is automatically tested, and the team is notified about which tests passed or failed. They fix any issues and once the new code passes all the tests, clear it for deployment.
- Different teams conduct continuous integration with different frequencies: some daily, some multiple times a day, and some after each individual code commit.
- Continuous integration helps teams identify and resolve product development issues sooner than they otherwise would, which improves the speed with which the team can iterate.

Summary

- Effective estimation is an essential element in the success of software projects
- Story points are a unit of measure for expressing the overall size of a user story, feature, or other piece of work
- Velocity is a measure of a team’s rate of progress
- Expert opinion, analogy, and disaggregation are the commonly used estimation techniques
- Planning poker combines expert opinion, analogy, and disaggregation into an enjoyable approach to estimating that results in quick but reliable estimates

Thank you

BITS Pilani

SEZG507: Product Discovery & Requirements Engineering

Session 16: Managing Use Case Evolution

Contents

- Managing Use Case Evolution
  - Introduction to Use Cases
  - Use Case as a Contract for Behaviour
  - Scope
  - Stakeholders and Actors
  - The Levels of Use Cases
  - Scenarios and Steps
  - Use Case Formats

What is a Use Case?

- A use case captures a contract between the stakeholders of a system about its behavior.
- The use case describes the system's behavior under various conditions as the system responds to a request from one of the stakeholders, called the *primary actor*.
  - The primary actor initiates an interaction with the system to accomplish some goal. The system responds, protecting the interests of all the stakeholders.
- Different sequences of behavior, or scenarios, can unfold, depending on the particular requests made and the conditions surrounding the requests.
- The use case gathers those different scenarios together.
- Use cases are fundamentally a text form, although they can be written using flow charts, sequence charts, Petri nets, or programming languages.
- Under normal circumstances, they serve as a means of communication from one person to another, often among people with no special training.
  - Simple text is, therefore, usually the best choice.

Three Concepts

- A well-written use case is easy to read. It consists of sentences written in only one grammatical form—a simple action step—in which an actor achieves a result or passes information to another actor.
- The writer has to master three concepts that apply to every sentence in the use case and to the use case as a whole.
  - *Scope:* What is really the system under discussion?
  - *Primary actor:* Who has the goal?
  - *Level:* How high- or low-level is that goal?

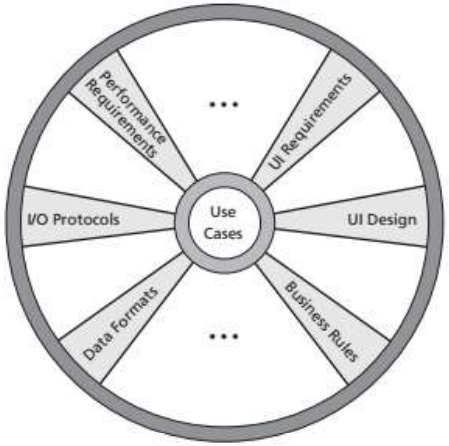
Key Components of a Use Case

- *Actor:* anyone or anything with behavior
- *Stakeholder:* someone or something with a vested interest in the behavior of the system under discussion (SuD)
- *Primary actor:* the stakeholder who or which initiates an interaction with the SuD to achieve a goal
- *Use case:* a contract for the behavior of the SuD
- *Scope:* identifies the system that we are discussing
- *Preconditions and guarantees:* what must be true before and after the use case runs
- *Main success scenario:* a case in which nothing goes wrong
- *Extensions:* what can happen differently during that scenario
  - Numbers in the extensions refer to the step numbers in the main success scenario at which each different situation is detected
- When a use case references another use case, the referenced use case is underlined

Requirements and Use Cases

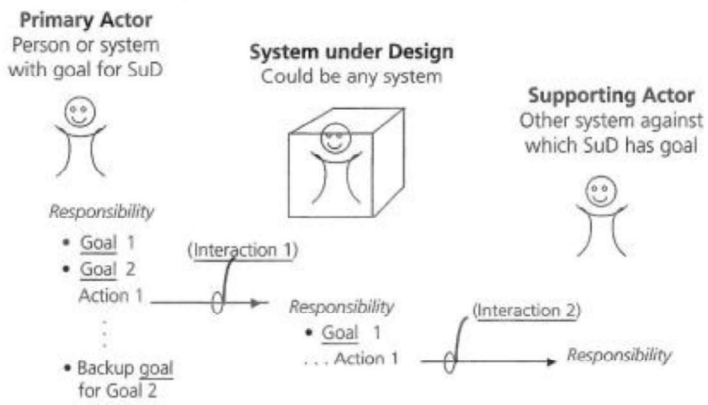
- They really are requirements
  - You shouldn't have to convert them into some other form of behavioral requirements. Properly written, they accurately detail what the system must do.
- They are not all of the requirements
  - They don't detail external interfaces, data formats, business rules, and complex formulae. They constitute only a fraction (perhaps a third) of all the requirements you need to collect—a very important fraction but a fraction nonetheless.

Use Cases as Project-Linking Structure



Use cases provide a scaffolding that connects information in different parts of the requirements and they help crosslink user profile information, business rules, and data format requirements

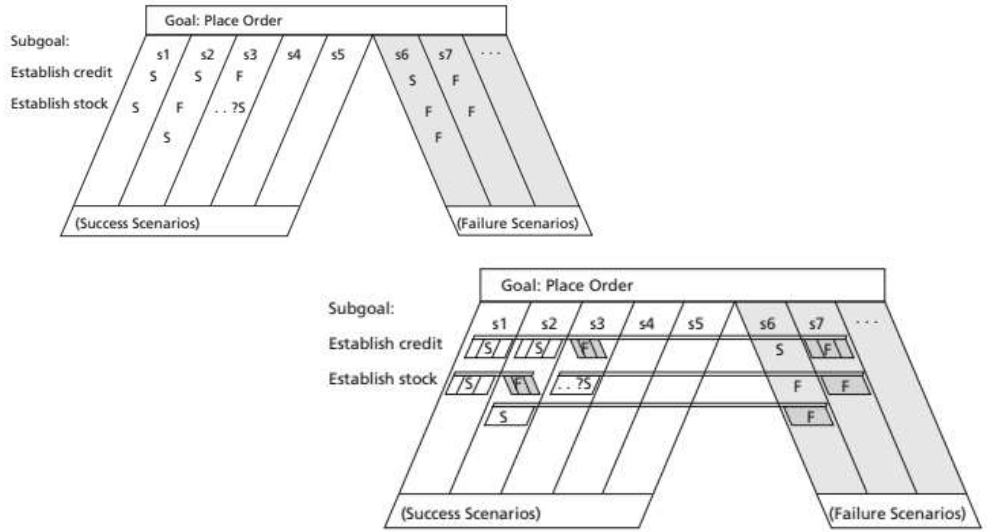
Actors Have Goals



A Use Case Collects Scenarios

- The primary actor has a goal; the system should help the primary actor reach that goal.
- Some scenarios show the goal being achieved; some end with it being abandoned.
- Each scenario contains a sequence of steps showing how the actions and interactions unfold. A use case collects all the scenarios together, showing all the ways that the goal can succeed or fail.

Scenarios Succeed or Fail: A Metaphor



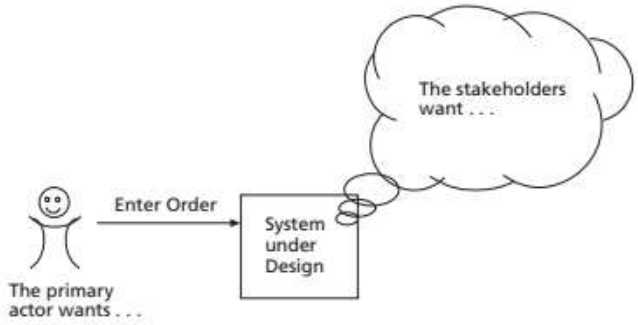
Principles Observed from the Metaphor

- Some scenarios end with success; some end with failure.
- A use case collects together all the scenarios, success and failure.
- Each scenario is a straight description for one set of circumstances with one outcome.
- Use cases contain scenarios (stripes on the trousers), and a scenario contains sub use cases as its steps.
- A step in a scenario does not care which stripe in the sub use case was used but only whether it ended with success or failure.

Contract between Stakeholders with Interests

- The Actors and Goals model explains how to write sentences in the use case, but it does not cover the need to describe the internal behavior of the system under discussion.
- For that reason the Actors and Goals model needs to be extended with the idea of a use case as a contract between stakeholders with interests.
- The SuD operates a contract between stakeholders, with the use cases detailing the behavioral part of that contract.
- Not all of the stakeholders are present while the system is running.
  - The primary actor is usually present, but not always.
  - The other stakeholders are not present, and so we might call them *offstage actors*.
  - The system acts to satisfy the interests of these offstage actors, including gathering information, running validation checks, and updating logs
- To satisfy the interests of the stakeholders, we need to describe three sorts of actions:
  - An interaction between two actors (to further a goal)
  - A validation (to protect a stakeholder)
  - An internal state change (on behalf of a stakeholder)

Primary Actor vs Offstage Stakeholders



Scope

- *Scope* is the word we use for the extent of what we design as opposed to someone else's design job or an already existing design.
- Keeping track of the scope of a project, or even just the scope of a discussion, can be difficult.
- The *in/out list* can be used to control scope discussions for ordinary meetings as well as project requirements
  - Simply construct a table with three columns.
  - The left column contains any topic; the next two columns are labeled "In" and "Out."
  - Whenever there might confusion as to whether a topic is within the scope of the discussion, add it to the table and ask people whether it is in or out.

A Sample In/Out List

Topic	In	Out
Invoicing in any form		Out
Producing reports about requests (e.g., by vendor, by part, by person)	In	
Merging requests into one PO	In	
Partial deliveries, late deliveries, wrong deliveries	In	
All new system services, software	In	
Any nonsoftware parts of the system		Out
Identification of any preexisting software that can be used	In	
Requisitions	In	

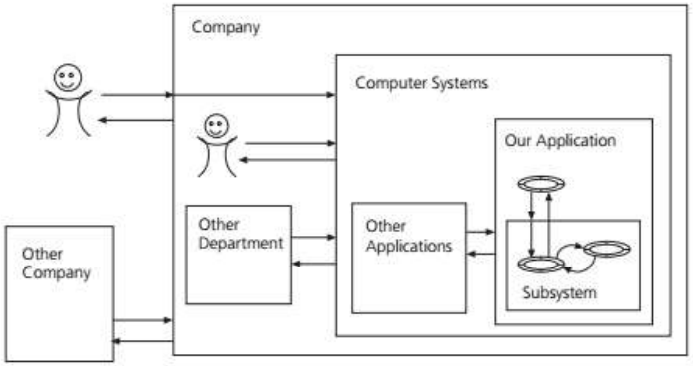
The Actor-Goal List

- The actor-goal list names all the user goals that the system supports, showing the system's functional content.
- Unlike the in/out list, which shows items that are both in and out of scope, the actor-goal list includes only the services that will actually be supported by the system.
- To make this list, construct a table of three columns.
  - Put the names of the primary actors—the actors having the goals—in the left column
  - Put each actor's goals with respect to the system in the middle column
  - Put the priority, or an initial guess as to the release in which the system will support that goal, in the third column
  - Update this list continually over the course of the project so that it always reflects the status of the system's functional boundary

A Sample Actor-Goal List

Actor	Task-level Goal	Priority
Any	Check on requests	1
Authorizor	Change authorizations	2
Buyer	Change vendor contacts	3
Requestor	Initiate a request	1
	Change a request	1
	Cancel a request	4
	Mark request delivered	4
	Refuse delivered goods	4
Approver	Complete request for submission	2
Buyer	Complete request for ordering	1
	Initiate PO with vendor	1
	Alert of nondelivery	4
Authorizer	Validate Approver's signature	3
Receiver	Register delivery	1

Design Scope



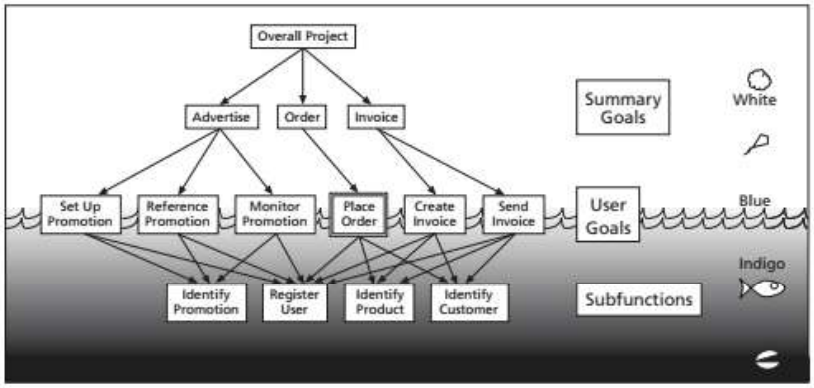
Stakeholders and Actors

- A stakeholder is someone who participates in the contract.
  - A stakeholder is someone or something that has a vested interest in the behavior of the use case.
- An actor is anything having behavior.
- An actor might be a person, a company or organization, a computer program, or a computer system—hardware, software, or both.
- Actors can be found in:
  - The *stakeholders* of the system
  - The *primary actor* of a use case
  - The *system under design* (*SuD*) itself
  - The *supporting actors* of a use case
  - The *internal actors*—the components within the SuD

The Primary Actor

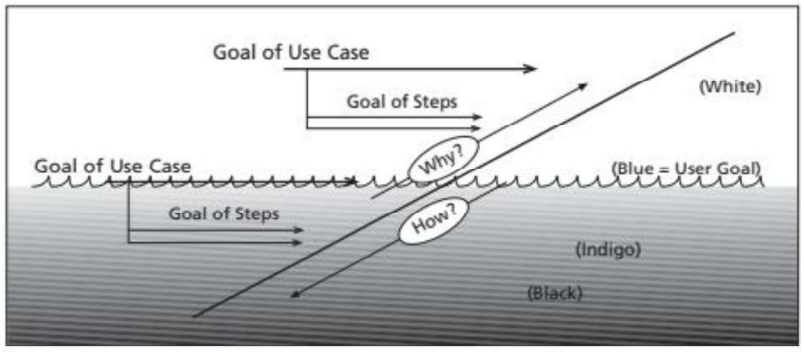
- The primary actor of a use case is the stakeholder that calls on the system to deliver one of its services.
- It has a goal with respect to the system—one that can be satisfied by its operation.
- The primary actor is often, but not always, the actor who triggers the use case.
- Primary actors are important at the beginning of requirements gathering and just before system delivery.
- A rich list of primary actors confers three other advantages:
  - It focuses our minds on the people who will use the system.
  - It sets up the structure for the actor-goal list, which will be used to prioritize and partition the development work.
  - It will be used to partition a large set of use cases into packages that can be given to different design teams.

Use Case Levels



The idea with the sea-level metaphor is this: The sky goes upwards for a long distance above sea level, and the water goes down for a long distance below sea level, but there is only one level where sky and sea meet: at sea level. The same holds for goals. There are many goal levels above the user goal and many below, but the user goals are the important ones to write. Therefore, sea level (waves) corresponds to them. A cloud or a kite indicates higher than sea level; a fish or a clam indicates lower than sea level.

Ask "Why" to Shift Levels



Scenarios and Steps

- A set of use cases is an ever-unfolding story of primary actors pursuing goals.
- Each use case has a crisscrossing story line that shows the system delivering the goal or abandoning it.
- This story line is presented as a main scenario and a set of scenario fragments as extensions to it.
- Each scenario or fragment starts from a triggering condition that indicates when it runs and goes until it shows completion or abandonment of its goal.
- Goals come in all different sizes, as we have seen, so we use the same writing form to describe the pursuit of any size goal at any scenario level.

Success Scenario and Surrounding Structure

- The main success scenario and all scenario extensions sit within a structure that consists of the following:
  - *A condition under which the scenario runs.*
  - *A goal to achieve.*
  - *A set of action steps.*
  - *An end condition*
  - *A possible set of extensions written as scenario fragments.*

Use Case Format: Fully Dressed

<the name should be the goal as a short active verb phrase>
Context of Use: <a longer statement of the goal, if needed, its normal occurrence conditions>
Scope: <design scope, what system is being considered black-box under design>
Level: <one of: summary, user-goal, subfunction>

Primary Actor: <a role name for the primary actor or description>
Stakeholders and Interests: <list of stakeholders and key interests in the use case>
Precondition: <what we expect is already the state of the world>
Minimal Guarantees: <how the interests are protected under all exits>
Success Guarantees: <the state of the world if goal succeeds>
Trigger: <what starts the use case, may be time event>
Main Success Scenario:
<put here the steps of the scenario from trigger to goal delivery and any cleanup after>
<step #> <action description>
Extensions:
<put here there extensions, one at a time, each referring to the step of the main scenario>
<step altered> <condition>: <action or sub use case>
<step altered> <condition>: <action or sub use case>
Technology & Data Variations List:
<put here the variations that will cause eventual bifurcation in the scenario>
<step or variation # > <list of variations>
<step or variation # > <list of variations>
Related Information:
<whatever your project needs for additional information>

Use Case Format: Casual

Primary Actor: User
Scope: Application
Level: Subfunction
Upon presenting themselves, the user is asked to enter a username and password. The system verifies that a submitter exists for that username and that the password corresponds to that submitter. The user is then given access to all the other submitter commands.
If the username corresponds to a submitter that is marked as an administrator, then the user is given access to all the submitter and administrator commands. If the username does not exist, or if the password does not match the username, then the user is rejected.

Use Case Format: One-Column Table

USE CASE #	<the name is the goal as a short active verb phrase>	
Context of Use	<a longer statement of the context of use if needed>	
Scope	<what system is being considered black box under design>	
Level	<one of summary, primary task, subfunction>	
Primary Actor	<a role name for the primary actor, or a description>	
Stakeholder and Interests	Stakeholder	Interest
	<stakeholder name>	<put here the interest of the stakeholder>
	<stakeholder name>	<put here the interest of the stakeholder>
Preconditions	<what we expect is already the state of the world>	
Minimal Guarantees	<the interests as protected on any exit>	
Success Guarantees	<the interests as satisfied on a successful ending>	
Trigger	<the action upon the system that starts the use case>	
Description	Step	Action
	1	<put here the steps of the scenario from trigger to goal delivery and any cleanup after>
	2	<...>
	3	
Extensions	Step	Branching Action
	1a	<condition causing branching> : <action or name of sub use case>
Technology and Data Variations		
	1	<list of variations>

Use Case Format: Two-Column Table

Customer	System
Enters order number	
	Detects that the order number matches the winning number of the month.
	Registers the user and order number as this month's winner.
	Sends an e-mail to the sales manager.
	Congratulates the customer and gives her instructions on how to collect the prize.
Exits the system	

Use Case: An Example

1. **Use Case Name:** Register for Courses
   - 1.1. **Brief Description**
     This use case allows a Student to register for course offerings in the current semester. The Student can also modify or delete course selections if changes are made within the add/drop period at the beginning of the semester. The Course Catalog System provides a list of all the course offerings for the current semester.
     The main actor of this use case is the Student. The Course Catalog System is an actor within the use case.
2. **Flow of Events:**
   The use case begins when the Student selects the "maintain schedule" activity from the Main Form. *[Refer to user-interface prototype for screen layout and fields.]*
   - 2.1. **Basic Flow**
     - 2.1.1. Create a Schedule
       - 2.1.1.1. The Student selects "create schedule."
       - 2.1.1.2. The system displays a blank schedule form. *[Refer to user-interface prototype for screen layout and to the domain model for required fields]*
       - 2.1.1.3. The system retrieves a list of available course offerings from the Course Catalog System. *[How is this selected and displayed? Text? Drop-down lists?]*
       - 2.1.1.4. The Student selects 4 primary course offerings and 2 alternate course offerings from the list of available offerings. Once the selections are complete the Student selects "submit." *[Define "primary course offerings" and "alternative course offerings" in project glossary. Must exactly 4 and 2 selections be made? Or "up to 4. . ." etc.]*
       - 2.1.1.5. The Add Course Offering subflow is performed at this step for each selected course offering.
       - 2.1.1.6. The System saves the schedule. *[When is the master schedule updated? Immediately? Nightly (batch)?]*
   - 2.2. **Alternative Flows**
     - 2.2.1. Modify a Schedule
       - 2.2.1.1. The Student selects "modify schedule."
       - 2.2.1.2. The system retrieves and displays the Student's current schedule (e.g., the schedule for the current semester). *[Is this only available for the current semester?]*
       - 2.2.1.3. The system retrieves a list of all the course offerings available for the current semester from the Course Catalog System. The system displays the list to the Student.
       - 2.2.1.4. The Student can then modify the course selections by deleting and adding new courses. The Student selects the courses to add from the list of available courses. The Student also selects any course offerings to delete from the existing schedule. Once the edits are complete the Student selects "submit."
       - 2.2.1.5. The Add Course Offering subflow is performed at this step for each selected course offering.
       - 2.2.1.6. The system saves the schedule.
     - 2.2.2. Delete a Schedule
       - 2.2.2.1. The Student selects the Delete Schedule activity.
       - 2.2.2.2. The system retrieves and displays the Student's current schedule.
       - 2.2.2.3. The Student selects "delete."
       - 2.2.2.4. The system prompts the Student to verify the deletion.
       - 2.2.2.5. The Student verifies the deletion.
       - 2.2.2.6. The system deletes the schedule. *[At what point are the student slots freed up?]*
     - 2.2.3. Save a Schedule
       At any point, the Student may choose to save a schedule without submitting it by selecting "save." The current schedule is saved, but the student is not added to any of the selected course offerings. The course offerings are marked as "selected" in the schedule.
     - 2.2.4. Add Course Offering
       The system verifies that the Student has the necessary prerequisites and that the course offering is open. The system then adds the Student to the selected course offering. The course offering is marked as "enrolled in" in the schedule.
     - 2.2.5. Unfulfilled Prerequisites or Course Full
       If in the Add Course subflow the system determines that the Student has not satisfied the necessary prerequisites or that the selected course offering is full, an error message is displayed. The Student can either select a different course offering or cancel the operation, at which point the use case is restarted.
     - 2.2.6. No Schedule Found
       If in the Modify a Schedule or Delete a Schedule subflows the system is unable to retrieve the Student's schedule, an error message is displayed. The Student acknowledges the error and the use case is restarted.
     - 2.2.7. Course Catalog System Unavailable
       If the system is unable to communicate with the Course Catalog System after a specified number of tries, the system will display an error message to the Student. The Student acknowledges the error message and the use case terminates.
     - 2.2.8. Course Registration Closed
       If, when the student selects "maintain schedule," registration for the current semester has been closed, a message is displayed to the Student and the use case terminates. Students cannot register for courses after registration for the current semester has been closed.
3. **Special Requirements**
   No special requirements have been specified for this use case at this time.
4. **Preconditions**
   - 4.1. Login
     Before this use case begins the Student has logged onto the system.
5. **Postconditions**
   There are no postconditions associated with this use case.
6. **Extension Points**
   There are no extension points associated with this use case.

Summary

- A use case captures a contract between the stakeholders of a system about its behavior.
- Use cases can be specified at varying levels.
- There are different formats for documenting use cases.

Thank you