Whitepaper

Voice of the customer

How fast teams figure out what to build and when it has to ship, and why the answer keeps moving while they build it.

FTTM methodology series.

A methodology paper from lateralworks. Voice of the customer is the discipline that produces the right-product input to FTTM. This paper sets out the four-step Wants / Translation / Solution / Refresh loop used by fast teams, with two anonymized case studies.

Prepared by	Date	Online
lateralworks Methodology paper	May 2026 Rev 1.0	lateralworks.com FTTM series

Table of contents

Voice of the customer

Core thesis. Right-product is determined first, not last. Voice of the customer is how fast teams figure out what “right” means today, then again next month, and again until the program ships. Programs that freeze the requirements early ship the wrong product into the right window, or the right product into the wrong one.

Overview

Abstract

Most product programs that miss their market window do not miss because the engineering is slow. They miss because the team has been building the wrong thing, or building the right thing for a customer who has moved on. Voice of the customer is the discipline that prevents both failures.

Right-product is the input that lets the rest of the FTTM system work. A team that knows precisely what to build and what not to build can focus its resources, hold its schedule, and converge. A team without that focus drifts. The schedule expands as the team chases late requirement changes, rebuilds features that turn out not to matter, and adds capability nobody asked for. Roughly 80% of future product success is set before the team writes a line of code or runs a first simulation, at the point where the team locks in what customers actually want [25].

Voice of the customer was named and structured by Griffin and Hauser in 1993 as the input layer of Quality Function Deployment [1]. The textbook VOC process gathers customer needs, structures them, and prioritizes them once, then hands the result to engineering as a frozen spec. lateralworks' work with fast teams since 1988 has consistently shown that one-shot VOC is the wrong shape for product programs whose market window is short and whose customer requirements move during development [29]. Fast teams treat VOC as a four-step loop, Wants, Translation, Solution, and Refresh, that runs continuously through the development cycle.

This paper sets out the lateralworks VOC methodology in the form fast teams actually use. It defines the wants/hows distinction that keeps teams from leading with their own solution. It walks through the four-step loop, including the analytic hierarchy work that turns subjective customer language into a defensible priority order. It locates VOC inside the FTTM trio of right-product, right-team, and right-time, and shows how a working VOC discipline lets the schedule converge in the back two-thirds of the program. Two anonymized case studies show the methodology applied to a mobile-phone chip program and to a multi-customer memory product.

The core argument is straightforward. The wants belong to the customer. The hows belong to the team. The team's job is to translate one into the other, validate the result with the customer, and do it again every time anything material changes. Programs that follow this discipline ship the right product into the right window. Programs that do not ship later, with more, that fewer people want [25, 28].

01

Why VOC matters

The right-product problem

A new product program is mostly determined by what it decides to build, not by how fast it builds. lateralworks' field data across 200-plus programs since 1988 [29] points repeatedly to the same finding: roughly four out of five products that miss their financial targets miss because the team built the wrong thing for the chosen window, not because the engineering execution slipped [25]. The schedule then absorbs the consequences. Programs drift while marketing renegotiates with customers, engineering reworks features that turn out not to matter, and resource plans absorb scope nobody is paying for.

This paper is a methodology document. It sets out a single voice-of-the-customer discipline, the four-step Wants/Translation/Solution/Refresh loop, and shows how the discipline lives inside the broader FTTM system [29]. The methodology assumes a competitive technology market, a constrained development window, and a team that already understands its own technology better than the customer does. Those three conditions describe the engagement environment for almost every fast-time-to-market program lateralworks has run.

Scope and audience

What this paper covers

Scope. The paper covers VOC during development. It does not cover post-launch product analytics, customer success measurement, or NPS-style satisfaction tracking. Those are different problems with different tools. VOC, as used here, is the discipline that turns a vague market opportunity into a converged product spec while there is still time to converge.

Audience. The paper is written for the host who owns the program. That is usually a VP of Engineering, a CTO, or a general manager with profit-and-loss responsibility for the new product. The four-step methodology is run by the core team, but the host sets the standard for what counts as complete on each step and protects the refresh cadence.

How the rest of the paper is organized. Section 02 defines the wants/hows distinction and the four-step loop. Section 03 covers how the team chooses which market and which customers to listen to in the first place. Section 04 covers discovery questions and customer visits. Section 05 covers the prioritization machinery, including the AHP models the team uses to rank wants and hows defensibly. Section 06 covers refresh, the step textbook VOC underweights. Section 07 places VOC inside the FTTM system and shows how the four-step loop interacts with the right-team and right-time disciplines. Appendices A and B work two anonymized case studies end to end.

Why right-product comes first

The financial weight of the right-product decision

House and Price's break-even-time work at HP in 1991 [12] made the argument quantitatively: the financial return of a new product depends as much on when it ships as on what it does, and shipping late produces returns substantially below shipping with reduced scope on time. Reinertsen and Smith generalized the point into a cost-of-delay framework [10, 11], which lets a team value a day of schedule slip in the same units it values a unit of margin or a unit of cost reduction. The combined work establishes the arithmetic that motivates fast time to market: if the program ships into the window, the program returns; if it ships outside the window, even a more complete product returns much less.

Right-product is the lever that protects the window. A team that has figured out which customer wants matter, which hows serve those wants, and which features can be deferred without losing the window is a team that can converge. A team still arguing about scope at the back half of the program is a team that ships late. The VOC discipline produces the data that turns the scope argument into a quantitative decision.

02

Methodology

Four steps and a refresh cadence

The single most common failure mode in product VOC is showing the customer a draft of the solution and asking what they think. The team has typically spent months thinking about its own product. It has internal documents describing what the product will do, what features it will include, and what architecture it will use. When the team walks into the customer meeting, the natural opening is to present what is already half-built. The customer responds politely, suggests some refinements, and the team leaves with confirmation of what it already believed [5, 6].

This pattern fails because it never reaches the customer's actual problem. The customer can react to a proposal but cannot reliably articulate, in a single meeting, what is missing from a proposal they have just been shown. Anchoring is too strong. The team's solution becomes the frame, and the customer evaluates that frame against a competitor's frame rather than against the underlying need [5, 6].

Definitions

Wants belong to the customer; hows belong to the team

The wants/hows distinction is the discipline that breaks the anchor. Wants are characteristics of a product that the customer values: low cost, small size, low power, functions while the host device is off, compatible with this specific protocol, available within this market window. Wants are stated in the customer's language and describe a quality or outcome the customer is trying to achieve. Hows are the implementations the team chooses to deliver against those wants: a shared host interface, a particular internal PLL frequency, a single package architecture, a specific power-management scheme. Wants come first; hows come from the team.

The Kano work from 1984 sharpens the structure further [3]. Some wants are must-be qualities, basic features the customer expects without comment, whose absence kills the product but whose presence does not differentiate. Some are performance qualities, where the customer evaluates linearly and more is better. A few are attractive qualities, features the customer did not ask for but recognizes as valuable when shown. The right product mix balances all three. A team that delivers only must-be qualities ships a competent me-too product. A team that delivers only attractive qualities ships something interesting that the market does not buy because the basics are missing.

Ulwick's outcome-driven work [4] argues a stronger version of the same point: the literal voice of the customer should be quieted, and the team should instead measure the customer's desired outcomes against the job the customer is trying to do. lateralworks' field experience supports the underlying logic. Customers really are bad at articulating product features and good at articulating outcomes. But the field experience does not require teams to silence the customer interview entirely. The wants/hows distinction does the same protective work in a less elaborate process.

Figure 1. Voice of the customer surfaces three classes of input: what the customer wants and does not want, when they want it, and what they value about it [29]. Treating those as separate questions keeps the team from collapsing "when" and "what" into a single product spec, which is the failure mode that produces the right product late.

The four steps

Wants, translation, solution, refresh

lateralworks structures VOC as a four-step loop [29]. Each step has a distinct output. The loop runs throughout development, not once at the start.

Wants

Wants are what customers say they want, what they actually mean, and the underlying customer requirements that come out of the translation. The output of this step is a structured list of customer requirements written in the customer's own language. Griffin and Hauser found in 1993 that interviewing roughly 20 to 30 customers in a market segment surfaces close to 90% of the needs that will ever surface from that segment [1]. The diminishing returns are real and useful: they let a team time-box the discovery phase rather than letting it run open-ended.

Translation

Translation converts customer requirements (wants) into product requirements (hows). A want such as "functions while the phone is off" translates into hows including a separate power rail, a low-leakage architecture, and a defined wake protocol. A single want can translate into several hows; a single how can address several wants. The output is a matrix that lets the team see which hows address which wants and how strongly. The matrix structure descends directly from the House of Quality work in QFD [2].

Solution

Solution is the team's proposal back to the customer, presented as a product brief: what the product does, what it does not do, the target price, the target schedule, and the architectural choices the team has made with the reasons. The brief is short. One or two pages, sometimes with a single diagram. It is deliberately not a spec. The point of the brief is to test whether the team has understood the customer correctly, not to commit the team to every detail.

Refresh

Refresh is where the loop becomes a loop. After the solution is presented, the team revisits customers regularly. Typically monthly during the divergence phase, less frequently as the program converges, to test whether wants have changed, whether new customers have entered the segment, and whether the competitive landscape has shifted. Refresh is the step the textbook VOC literature underweights, and it is the step that distinguishes fast teams from slow ones. A team that never refreshes ships against a frozen spec into a market that has moved.

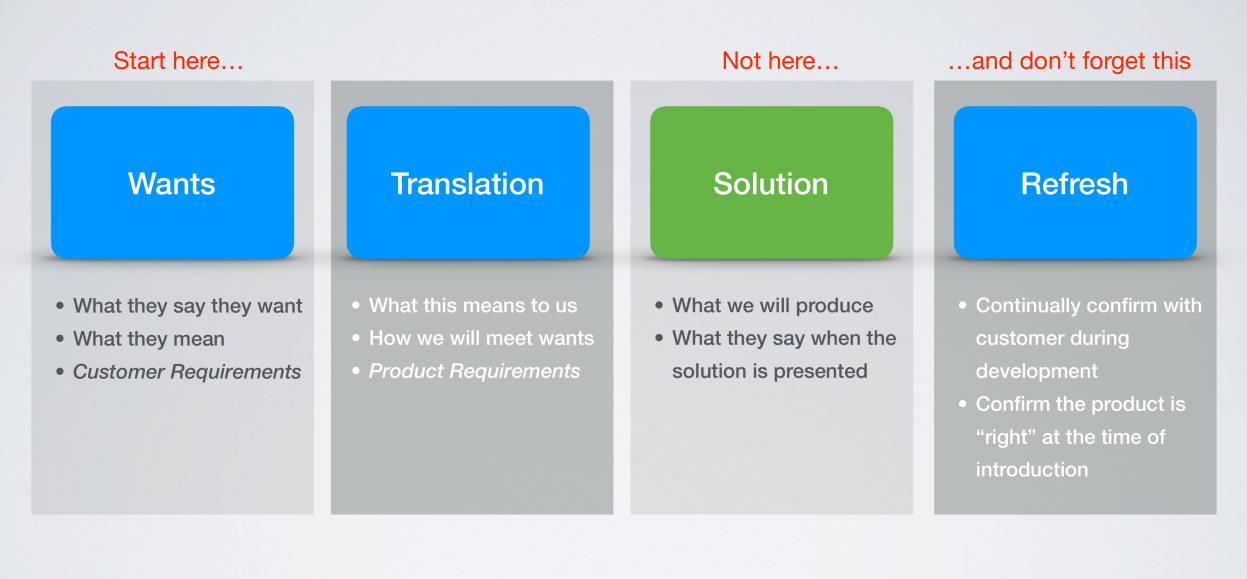

Figure 2. The four-step VOC loop. Fast teams enter at Wants and treat Refresh as a standing cadence. Teams that enter at Solution skip wants and translation and validate their own assumptions back to themselves [29].

Who do we ask

Finding the driving market and customer

VOC done with the wrong customers produces high-confidence answers to the wrong questions. Before the discovery interviews begin, two prior decisions have to be made: which market segment will the product target, and which specific customers inside that segment will drive the requirements.

Filter, select, prioritize markets

A new technology platform usually has more than one plausible market segment. A wireless chip might target mobile phones, laptop dongles, or audio headsets. Each segment has its own volume, margin, time-to-market sensitivity, and competitive structure. The right segment is rarely obvious from the top of the funnel, and reasoning about it qualitatively tends to produce decisions that reflect whoever spoke last.

lateralworks uses an analytic-hierarchy-process model [7] to make the segment decision visible and reversible. The team lists the business objectives it cares about (time-to-market sensitivity, expected unit volume, gross margin, profit-before-tax, ease of entry, leverage of existing technology) and uses pairwise comparison to weight them. The candidate segments are then scored against each weighted objective. The result is a ranked list of segments with an explicit sensitivity profile: how the ranking would change if a different objective were given priority.

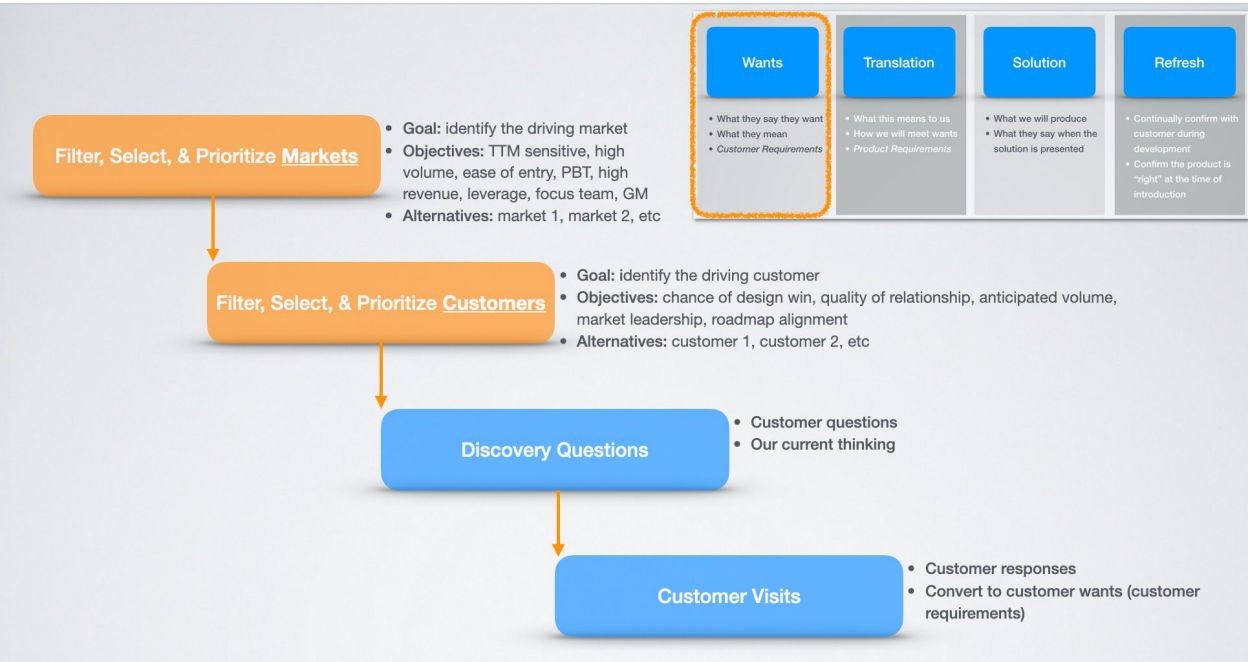

Figure 3. The opening process: filter the markets, pick the driving customer set, draft discovery questions, run customer visits. Each step has its own AHP model and its own deliverable [29].

In one engagement, this analysis showed that the cell-phone segment alone won when gross margin was the dominant objective, while a combined cell-and-PC strategy won when revenue was the dominant

objective. The team could see, before committing engineering resources, that the segment choice was actually a choice about which business objective the program was optimizing. Making that visible let the executive sponsor decide explicitly, rather than implicitly through engineering behavior six months later.

Filter, select, prioritize customers

Inside the chosen segment, the team identifies a small set of candidate customers and runs the same kind of AHP model to identify the driving customer. The criteria are different (chance of design win, quality of relationship, anticipated unit volume, market leadership in the segment, alignment with the team's product roadmap) but the structure is identical. A few specific customers come out of the model with substantially higher weight than the others. Those are the customers whose wants will drive the product definition.

The driving-customer concept matches von Hippel's lead-user research [8], which found that the most innovation-relevant customer input in a market typically comes from a small minority of users who are ahead of the rest of the market on the underlying need. Driving customers are usually market leaders in their segment, and their requirements typically anticipate where the rest of the segment will be by the time the product ships. As one veteran chip designer once put it to us, focus on the market leader and the followers will follow [29].

In a typical engagement the AHP run produces a top-three set of driving customers and a long tail. The team interviews the top three intensively, tracks the next two through normal sales coverage, and ignores the bottom of the list for VOC purposes. Trying to interview every candidate customer is one of the failure modes of overzealous VOC programs. Coverage is not a goal; defensible focus is.

Principle

What customers tell you

Customers tell you what they want. It is up to you to figure out how to deliver that.

lateralworks engagement data

Across 200+ FTTM programs since 1988

Doing the interviews

Discovery questions and customer visits

Discovery questions are the team's working hypothesis about what it wants to learn from the customer, paired with the team's own current thinking on each question. The pairing matters. Without an explicit "our current thinking" column, the team's assumptions stay invisible and the interview leads the witness. With the column, the assumptions are written down and the customer can react to them directly, either confirming them or, much more usefully, disagreeing in ways the team did not anticipate.

A typical discovery sheet runs 15 to 30 questions per customer category, organized by subject area (system architecture, use cases, environmental conditions, target price, market timing) and tagged with priority. Questions are not rhetorical and do not propose features. A good question asks what problem the customer is trying to solve. A weak question asks whether the customer would like a feature the team plans to build.

Visits, not surveys

Customer visits, meaning structured site visits by a small team that includes engineering and not just sales, are the lateralworks standard for VOC data collection. McQuarrie's customer-visits research [9] makes the case in detail: surveys produce data that confirms what is already in the response options, while visits produce data the team did not expect to encounter. The difference matters for fast teams because the productive output of VOC is not a confirmation of the team's existing assumptions but a discovery of the gap between what the team assumes and what the customer actually intends.

The right team for a visit is small. Two or three people, including one engineer who can interpret what the customer says technically, one product or program manager who can hear the business signal, and one note-taker. The visit lasts two to four hours. The agenda is the discovery questions, not a sales pitch.

What they say versus what they mean

The most consistent finding from lateralworks' engagement data is that customers' initial responses rarely express the underlying intent [29]. A customer asked whether near-field functionality should work with the phone off may say "sure, that would be good." Pushed for context, the same customer may explain that public transit ticketing is the actual use case, and that ticketing has to work whether the battery is full, low, or completely flat. The first response is data. The second response is the want.

Capturing both responses is the point of the visit. The team writes down what the customer literally said in one column and what the customer meant in another. Translation happens at the back of the room afterwards, with everyone who attended the visit reviewing the notes together. Translation in the room while the customer is talking is a common mistake. The team starts hearing what it wants to hear and stops hearing what the customer is actually telling them.

Making the calls

Prioritizing wants and hows

A list of customer requirements is not a product definition. The list has to be ranked, and the ranking has to be defensible. Most teams rank requirements implicitly, through arguments about which features get built first and which get postponed. lateralworks runs the ranking explicitly, using AHP pairwise comparison [7], because the engineering trade-offs depend on it and because the customer base needs to be able to influence it.

Wants ranking

The wants model takes the customer requirements as alternatives and the driving customers as criteria. Each customer expresses pairwise importance among the wants. The output is a ranked list of wants, weighted by the prior driving-customer priorities. The model shows not just which wants matter most overall but also which wants matter most to which customer, so the team can see whether the top customer's wants and the second customer's wants are aligned or in tension.

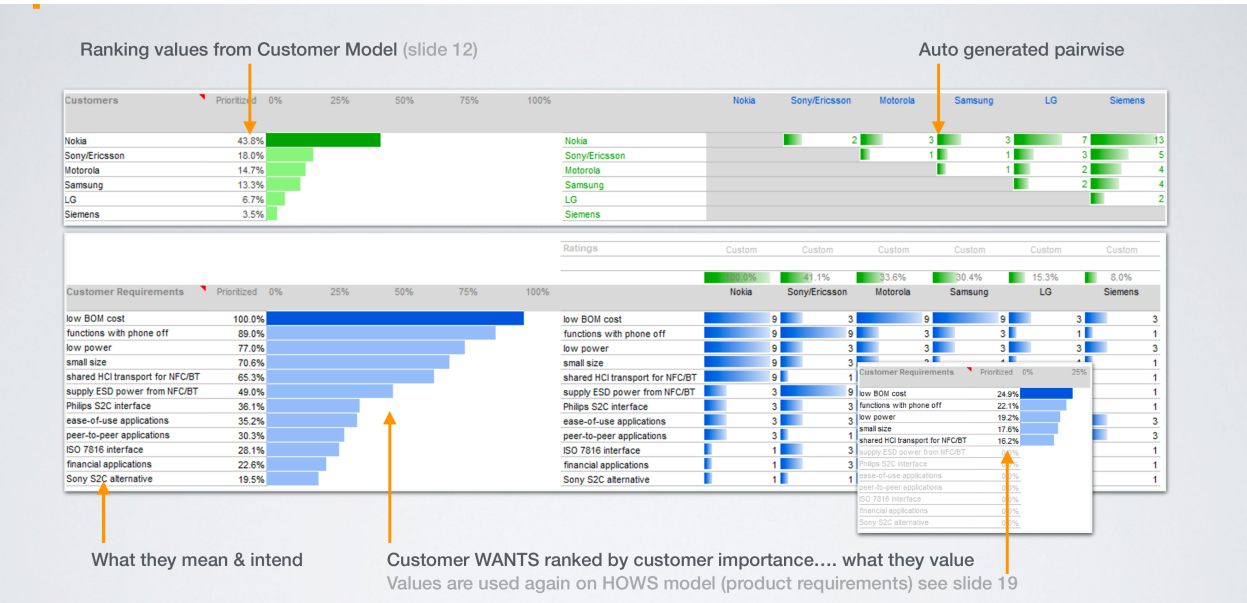

Figure 4. Customer wants ranked by weighted importance. The pairwise model uses the driving-customer weights as input, so a change in the customer ranking propagates automatically through the wants ranking [29].

In a typical engagement the top five wants take roughly 60% of the total weight. Those become the focus of the product definition. The bottom wants typically take less than 20% combined; they get acknowledged but do not drive architecture decisions.

Hows ranking

The hows model takes the product requirements as alternatives and the wants as criteria, with the want weights carried in from the wants model. The output is a ranked list of hows: which implementation choices contribute the most to the customer-weighted wants. The team can see, for example, that the shared host interface contributes to four of the top five wants while a particular package option contributes to only one. That visibility lets the team focus engineering investment on the hows with the highest leverage [29].

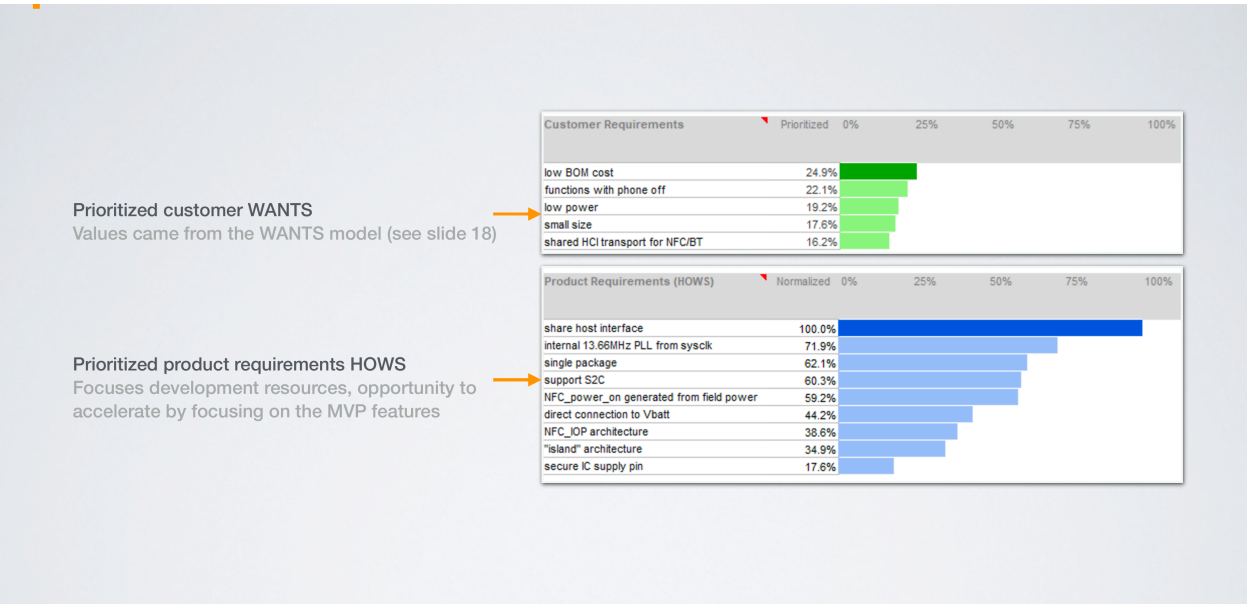

Figure 5. Product requirements (hows) ranked by their contribution to the prioritized customer wants. The shared host interface scores 100% because it serves four of the top five wants [29].

What-if analysis and architectural trade-offs

AHP models are also useful for what-if analysis. If the customer's top want suddenly becomes small size instead of low cost, which hows rise in priority and which fall? In one engagement the model showed that a hard pivot from cost to size drove a single-package architecture to the top of the hows ranking and demoted the share-host-interface how. The team could see that the architectural implications of the want change were substantial, and could go back to the customer to confirm whether the change in priority was real before committing engineering to a different architecture.

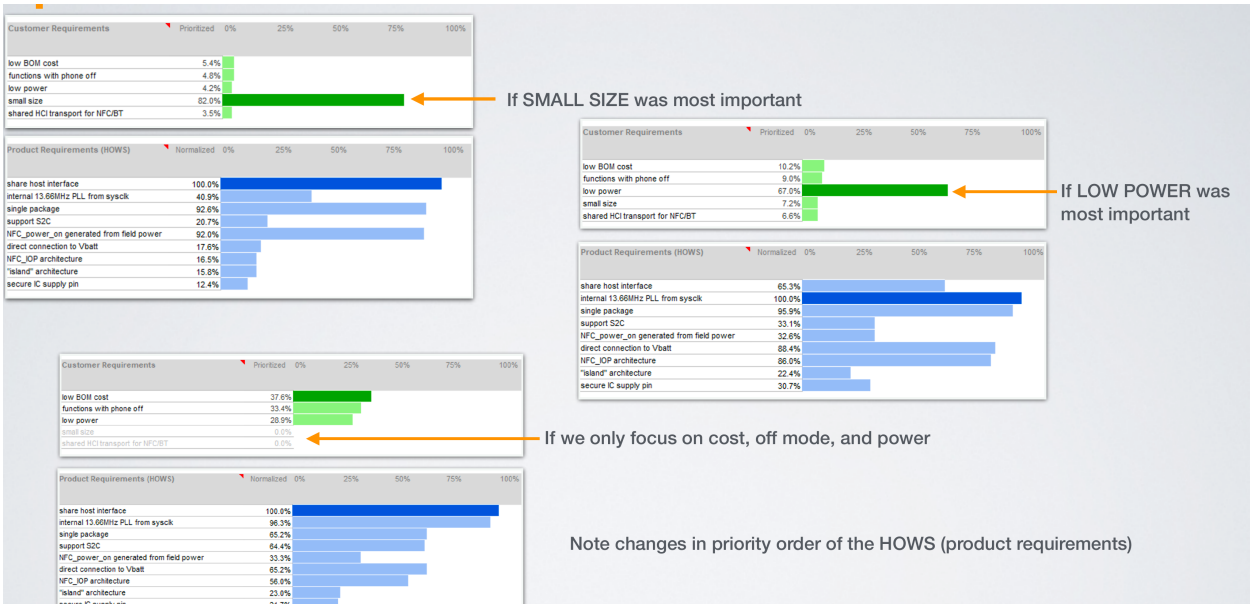

Figure 6. Three what-if scenarios show how the ranked hows shift when different customer wants are weighted most heavily. The model lets the team test the consequences of a priority change before committing engineering to it [29].

Architectural trade-offs sit on top of customer wants but include criteria the customer does not see directly: technical risk, competitive position, time-to-market. The combined model lets the team pick the architecture that maximizes customer-weighted value subject to those engineering constraints. It also produces an artifact that survives the next iteration of the wants, so when the customer comes back with a refined requirement the team can re-run the model rather than re-arguing the architecture from scratch.

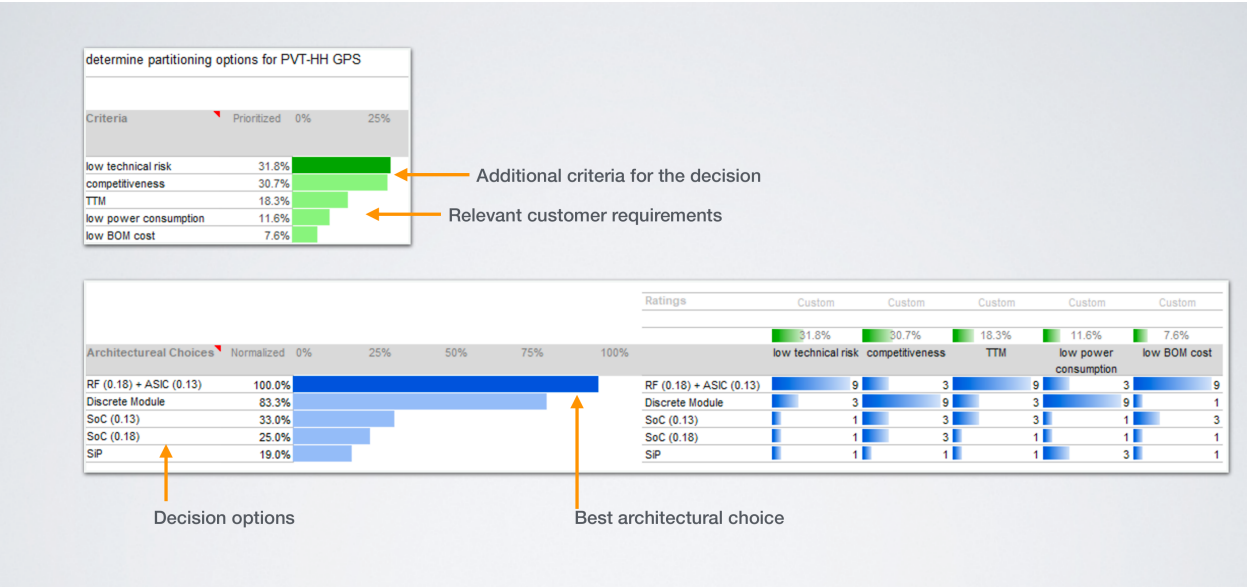

Figure 7. An architectural trade-off model combines customer wants (low power, low cost) with team criteria (technical risk, competitiveness, time-to-market). The combined ranking selects the architecture that best serves the weighted criteria [29].

Benchmarking against competition

The same model structure runs against competitor products. The team scores its own product and each competitor against the prioritized wants. The output is a head-to-head comparison that shows where the

team's product wins, where it loses, and what would have to change to flip the comparison. Sensitivity analysis (what happens if the most important want changes) flags the bets the team is making about which wants will matter at launch.

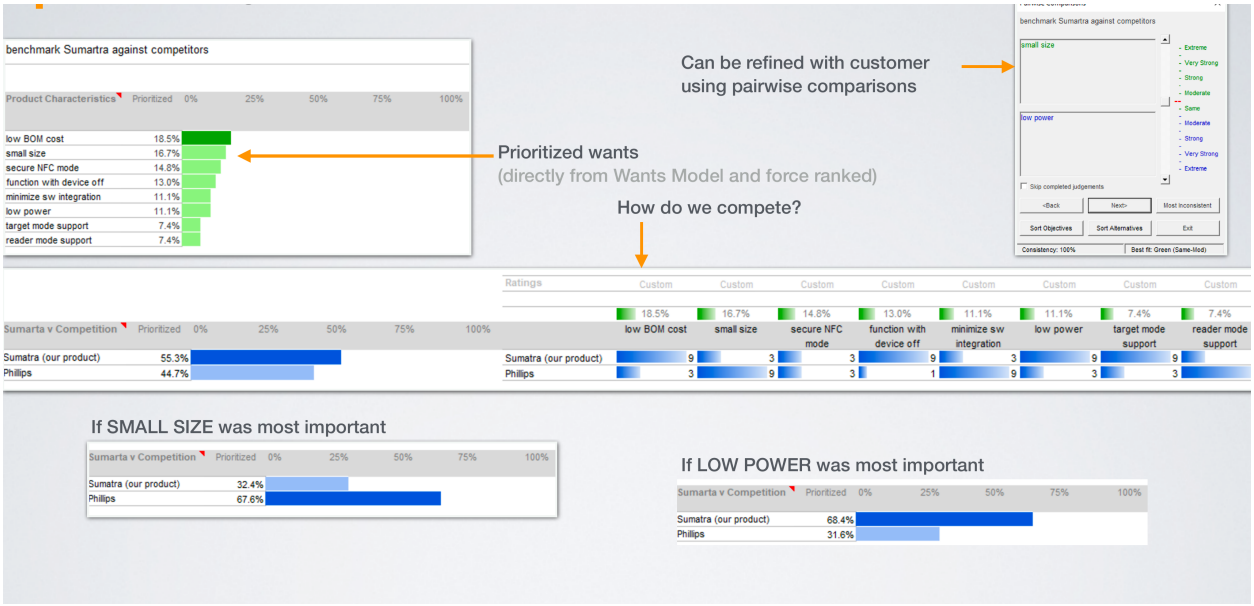

Figure 8. Benchmark of the team's product against a single competitor across the prioritized customer wants. The bottom panels run two sensitivity scenarios: if small size becomes the dominant want, the competitor wins; if low power dominates, the team's product wins [29].

Continuous discovery

Why VOC has to stay open

A single VOC pass at the start of a program is the most common failure mode in industry practice. The team gathers requirements, ranks them, freezes them, and hands the spec to engineering. By the time the product ships 18 to 36 months later, the customer's priorities have moved, the competitive landscape has shifted, and the spec is wrong [25, 29].

Refresh is the discipline that prevents the freeze. Fast teams revisit their VOC models on a fixed cadence (monthly during the divergence phase, quarterly during convergence, and on every major change-of-circumstance event). The cadence is short enough that drift is visible while there is still time to correct, and long enough that the customer base does not feel surveyed to exhaustion.

What gets refreshed

Three things refresh on the standing cadence. First, the prioritized wants model: are the same wants still the top wants? Has a new want emerged? Has an old want fallen off? Second, the prioritized hows model: do the team's implementation choices still serve the wants that are now on top? Third, the competitive benchmark: have any new entrants reset the must-be threshold, and is the differentiated position the team is building toward still differentiated?

Change management

Changes that come out of refresh hit the program as scope changes. The team has to decide whether to accept them (which usually means absorbing schedule impact) or to defer them (which usually means shipping with a known gap). lateralworks runs that decision through a change-management process that uses a cost-of-delay model [10, 11] to value the impact: what would the change cost in delay, what would the change cost not to make, and which is bigger? The answer is the trade-off the executive sponsor approves.

Reinertsen has argued for two decades that cost of delay is the most underused number in product development [10]. lateralworks' field experience confirms this. Teams that quantify the delay cost of a late requirement change make different decisions than teams that argue about it qualitatively. The decisions are usually faster, and they hold against pressure better, because the dollar value of the delay is visible.

Refresh and the schedule

Refresh interacts with the schedule. A change accepted in refresh is a schedule risk the team has accepted explicitly. A change deferred in refresh is a schedule slip the team has avoided. The trade-off is visible to the executive sponsor and to the customer. That visibility is what lets the schedule converge: the team and the host can agree on what is in and what is out, and the engineering work can settle around a stable target.

07

System view

VOC inside the FTTM system

FTTM treats new-product success as a function of three things working together: the right product, the right team, and the right time [29]. VOC is the discipline that produces the right-product input. The right-team and right-time disciplines depend on it.

Without a working VOC discipline the team has no anchor. Engineering chooses what to build based on what is technically interesting; product management chooses based on what is loudest in the last customer meeting; the executive sponsor chooses based on the most recent strategy presentation. The three choices conflict, and the conflict drains schedule. With a working VOC discipline the team has a single ranked list of wants, a single matrix of hows against wants, and a refresh cadence that keeps both current. The conflict resolves before it consumes schedule.

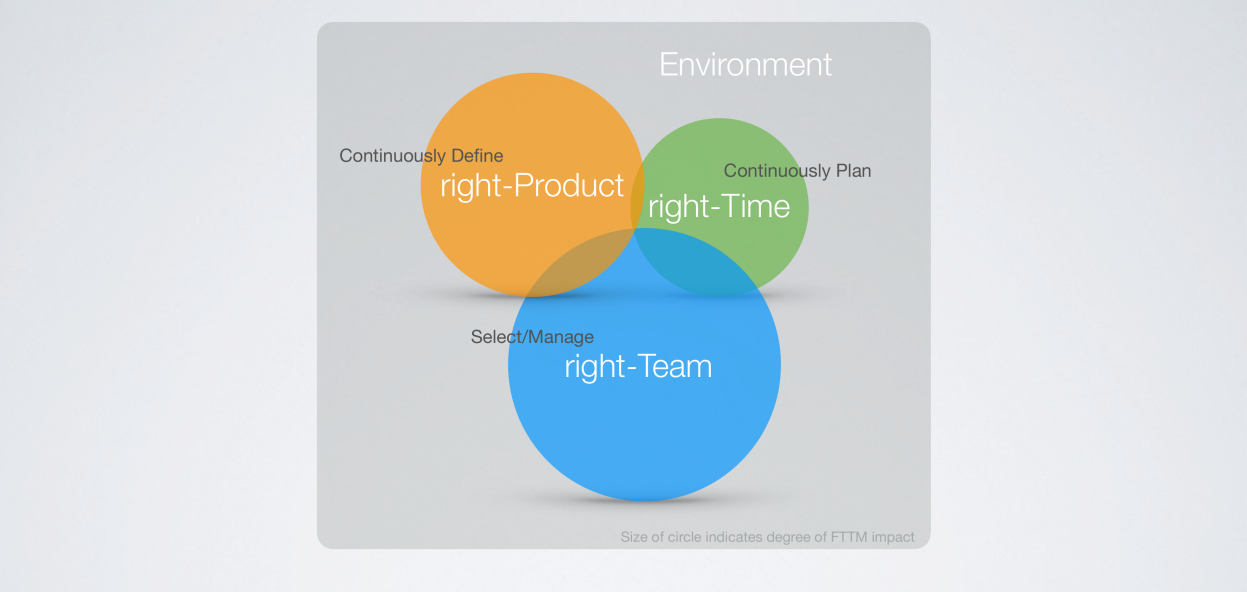

Figure 9. The FTTM system. Right-product is the input the team converges around; right-team is the structure that lets the convergence happen quickly; right-time is the constraint that forces the convergence to finish. The size of each circle reflects the relative impact of each variable on overall FTTM performance [29].

Diverge one third, converge two thirds

Fast teams divide the program time-budget roughly one-third divergence, two-thirds convergence [26, 29]. Divergence is where VOC does its primary work: wants gathering, customer visits, architecture exploration, the first solution proposal. Convergence is where the team locks the wants and the hows and builds the product around them, while running a lighter refresh cadence to catch material changes. Teams that stay in divergence past the one-third point run out of convergence runway and ship late. Teams that enter convergence before they have a stable wants list converge fast around the wrong product.

Right-time forces the convergence

Right-time is the constraint that disciplines the VOC loop. A want that cannot be implemented in the available window is not a want the team can serve; it is a want the team has to negotiate. The customer either accepts a later delivery, accepts a partial implementation, or chooses a competitor. Right-time is therefore a customer input in its own right [29]: when the customer wants the product, in what volumes, with what level of completeness. House and Price's break-even-time framework [12] captures the same point in financial terms.

Right-team executes the trade-offs

The right-team, a cross-functional integrated core team [13, 14], is the structure that lets the trade-offs happen quickly. Marketing, engineering, manufacturing, and quality on the same team mean a customer want raised in refresh can be evaluated against engineering feasibility and manufacturing impact in the same meeting. A want raised across functional silos takes weeks to evaluate; a want raised inside an integrated core team takes hours. The refresh cadence depends on the team structure being able to keep up with it.

Where this leaves the program

A program with a working VOC discipline has three things at every point in its life: a ranked list of customer wants, a ranked list of product hows that serve those wants, and a refresh cadence that keeps both current. A program with all three converges. A program missing any of the three drifts. The argument of this paper is that the discipline of maintaining the three is the most leveraged thing the host can do for fast time to market, because it sets the conditions under which the engineering and the schedule can do their work.

A

Case study A

Mobile NFC chip wants and hows

This appendix walks the four-step methodology through a single anonymized engagement. The program was a wireless chip targeting an emerging mobile-phone protocol. Customer identities, exact specifications, and commercial details have been removed. The structure of the analysis is unchanged.

The work, step by step

Mobile NFC chip: filter, visit, prioritize

Step 1 — Filter markets

The team identified three plausible target segments: a high-volume consumer segment, a moderate-volume professional segment, and a low-volume specialty segment. The AHP model showed the consumer segment ranked highest on revenue and time-to-market sensitivity, the professional segment ranked highest on gross margin, and the specialty segment ranked lowest on all but ease of entry. A combined consumer-plus-professional strategy emerged when revenue was weighted high; the consumer segment alone emerged when gross margin was weighted high. The executive sponsor chose the consumer segment alone, on the strength of the time-to-market sensitivity.

Step 2 — Filter customers

Inside the consumer segment, six candidate customers were identified by sales coverage. The AHP model ranked them on chance of design win, quality of relationship, anticipated volume, market leadership in the segment, and roadmap alignment. Three customers emerged as the drivers: the market leader and two strong followers. The fourth-ranked customer had a different roadmap and was dropped. The bottom two were tracked but not interviewed.

Step 3 — Discovery questions and visits

Roughly 30 discovery questions were developed across system architecture, protocol mode requirements, environmental conditions, target price, and market timing. The team visited each of the three driving customers in person. The visits surfaced one finding the team had not anticipated: a critical use case required functionality with the host device powered off, which conflicted directly with the team’s planned power-management architecture. Anchoring on the original architecture would have produced a product that failed the use case at launch.

Step 4 — Prioritization and architecture

The wants model ranked five customer requirements at the top: low BOM cost, functions with phone off, low power, small size, and shared host interface. The hows model translated those into ranked product requirements: shared host interface, internal PLL, single-package architecture, and a specific power-generation scheme. An architectural trade-off model evaluated five candidate architectures against the prioritized wants plus technical risk, competitiveness, and time-to-market criteria. The chosen architecture combined an RF process node with a separate ASIC node, which scored highest on combined customer-weighted value and engineering feasibility.

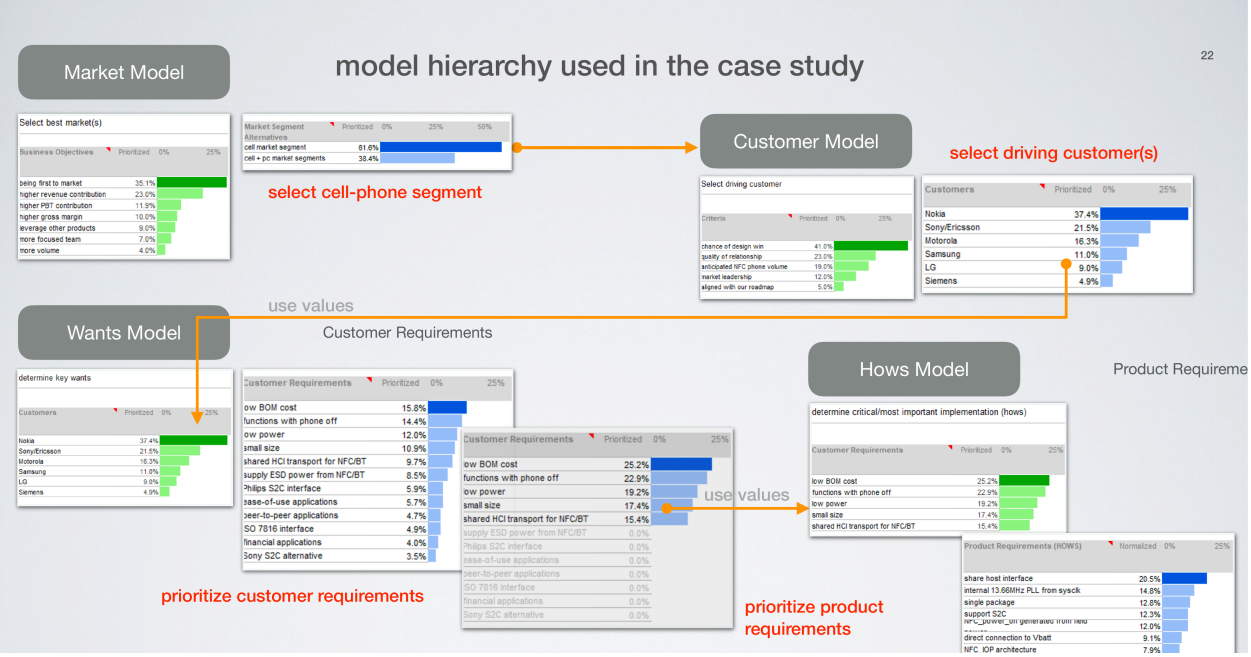

Figure 10. The four-model hierarchy used in the engagement. Output weights from each model feed into the next as input weights, so a change in the driving-customer ranking propagates automatically through the wants and hows models [29].

Refresh during execution

The program ran the wants and hows models on a monthly refresh cadence for the first six months and a quarterly cadence after that. Two material changes came out of refresh. The top customer accelerated its product schedule by one quarter, which the team absorbed by deferring one of the secondary hows. A competitor announced a different package option, which the team absorbed by adjusting the package strategy without re-architecting the core. Neither change drove a major schedule slip, and both decisions were made in a single refresh meeting using the existing AHP models.

B

Case study B

Multi-customer memory product

The second case shows the methodology applied where the team had to serve more than one driving customer with substantially different requirement profiles. The program was a high-density memory product. Three driving customers were ranked highest by the customer model. Each customer’s prioritized wants produced a different ranking of product features. The team could not pick one customer at the expense of the other two, and it could not build three different products.

Multi-customer MVP

Distill, brief, refine, repeat

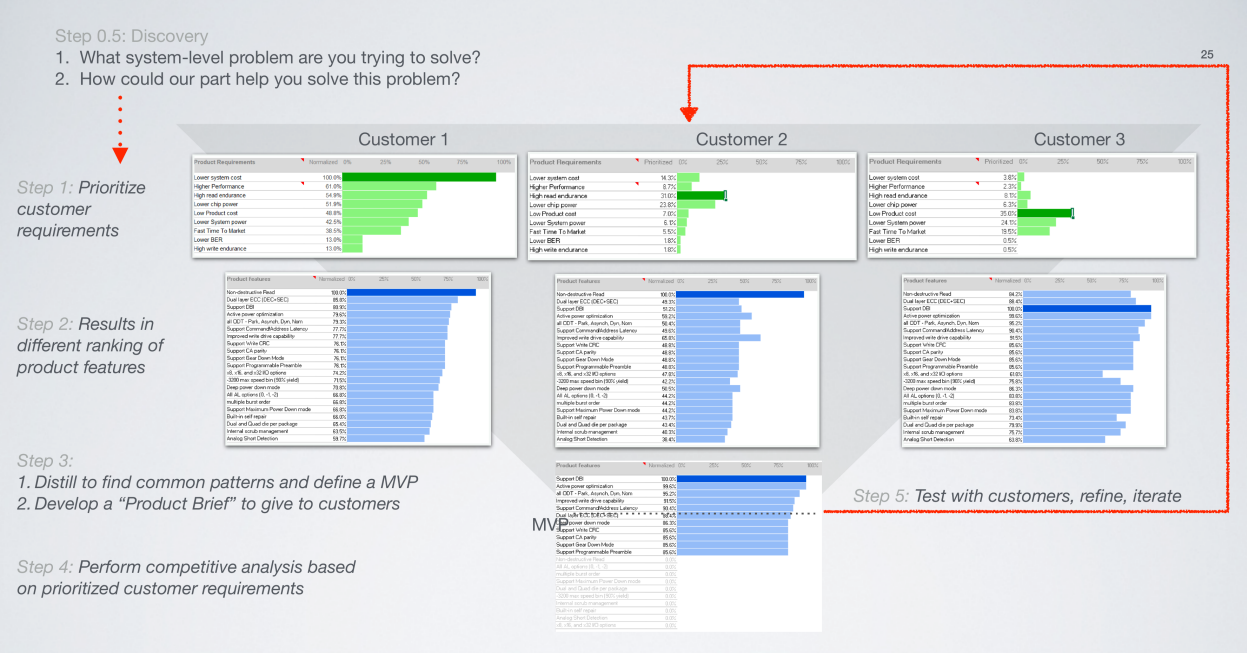

Figure 11. Three driving customers produce three different feature priorities. The minimum viable product is the intersection of the top-weighted features across at least two of the three customers [29].

The exercise the team had to do was distillation, not consensus. Asking the three customers to agree on a single feature list would have produced either the lowest-common-denominator product or an unbuildable superset. Instead, the team identified the features that ranked in the top quartile for at least two of the three customers, packaged those into a minimum viable product, and presented it back to all three customers as a product brief. All three accepted the brief, with refinements. The refinements were absorbed into the wants model and the cycle repeated.

This pattern (distill, brief, refine, repeat) enables a single product to serve a multi-customer market without expanding into a different product per customer. The AHP model is what makes the distillation defensible. Without the model, the decision about which features stay and which go is a political negotiation. With the model, the decision is a weighted output the team can show to each customer.

The case also illustrates a secondary use of refresh: catching the moment when a previously-aligned multi-customer set starts to diverge. Six months into the program, one of the three customers shifted its priorities away from the common set toward features specific to its own system architecture. The refresh model surfaced the divergence within one cycle. The team negotiated with that customer to absorb the change as a future product variant, and shipped the original MVP on schedule into the remaining two customers, which still accounted for the majority of the program's expected revenue. Without the refresh cadence, the team would have either over-extended into the third customer's requirements or shipped

without the third customer at all. The refresh gave the team enough time to negotiate a structured exit.

Sources

References

[1] Griffin, A., and Hauser, J. R. “The Voice of the Customer.” *Marketing Science*, vol. 12, no. 1, 1993, pp. 1–27. https://doi.org/10.1287/mksc.12.1.1

[2] Hauser, J. R., and Clausing, D. “The House of Quality.” *Harvard Business Review*, vol. 66, no. 3, May–June 1988, pp. 63–73.

[3] Kano, N., Seraku, N., Takahashi, F., and Tsuji, S. “Attractive Quality and Must-Be Quality.” *Journal of the Japanese Society for Quality Control*, vol. 14, no. 2, April 1984, pp. 39–48.

[4] Ulwick, A. W. *What Customers Want: Using Outcome-Driven Innovation to Create Breakthrough Products and Services*. McGraw-Hill, 2005.

[5] lateralworks. “VOC: Differentiate ‘wants’ from ‘hows’.” *lateralworks Ideas*, June 2019. https://lateralworks.com/ideas/voc-differentiate-wants-from-hows

[6] lateralworks. “Feature-based VOC vs discovering customer needs.” *lateralworks Ideas*, June 2019. https://lateralworks.com/ideas/feature-based-voc-vs-discovering-customer-needs

[7] Saaty, T. L. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. McGraw-Hill, 1980.

[8] von Hippel, E. “Lead Users: A Source of Novel Product Concepts.” *Management Science*, vol. 32, no. 7, 1986, pp. 791–805.

[9] McQuarrie, E. F. *Customer Visits: Building a Better Market Focus*. 3rd ed., M.E. Sharpe, 2008. First edition Sage Publications, 1993.

[10] Reinertsen, D. G. *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing, 2009.

[11] Smith, P. G., and Reinertsen, D. G. *Developing Products in Half the Time: New Rules, New Tools*. 2nd ed., Van Nostrand Reinhold, 1998.

[12] House, C. H., and Price, R. L. “The Return Map: Tracking Product Teams.” *Harvard Business Review*, vol. 69, no. 1, January–February 1991, pp. 92–101.

[13] Wheelwright, S. C., and Clark, K. B. *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality*. Free Press, 1992.

[14] Clark, K. B., and Fujimoto, T. *Product Development Performance: Strategy, Organization, and Management in the World Auto Industry*. Harvard Business School Press, 1991.

[15] Christensen, C. M. *The Innovator’s Dilemma: When New Technologies Cause Great Firms to Fail*. Harvard Business School Press, 1997.

[16] Christensen, C. M., and Raynor, M. E. *The Innovator’s Solution: Creating and Sustaining Successful Growth*. Harvard Business School Press, 2003.

[17] Cooper, R. G. *Winning at New Products: Accelerating the Process from Idea to Launch*. 5th ed., Basic Books, 2017.

[18] Cooper, R. G., and Edgett, S. J. *Portfolio Management for New Products*. 2nd ed., Perseus Publishing, 2001.

[19] Ohno, T. *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 1988.

[20] Katz, G. “The ‘One Right Way’ to Gather the Voice of the Customer.” *PDMA Visions Magazine*, vol. 25, no. 2, April 2001, pp. 8–12.

[21] Goldratt, E. M. *Critical Chain*. North River Press, 1997.

[22] Blank, S. *The Four Steps to the Epiphany*. 5th ed., K&S; Ranch, 2013. First published 2005.

[23] Ries, E. *The Lean Startup*. Crown Business, 2011.

[24] Anderson, D. J. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

[25] lateralworks. “Right Product... 80% of Future Product Success.” *lateralworks Ideas*, January 2019. https://lateralworks.com/ideas/right-product-80-of-future-product-success

[26] lateralworks. “What, Who, Why, and Value?” *lateralworks Ideas*, July 2019. https://lateralworks.com/ideas/what-who-why-and-value

[27] lateralworks. “From target market to product definition using VOC.” *lateralworks Ideas*, April 2019. https://lateralworks.com/ideas/from-target-market-to-product-definition-using-voc

[28] lateralworks. “Unmanaged fuzzy-front-end.” *lateralworks Ideas*, June 2019. https://lateralworks.com/ideas/unmanaged-fuzzy-front-end

[29] lateralworks. “Internal assessment database — Voice of the Customer engagements.” Engagement data across client programs, 1988–2026.