GQM

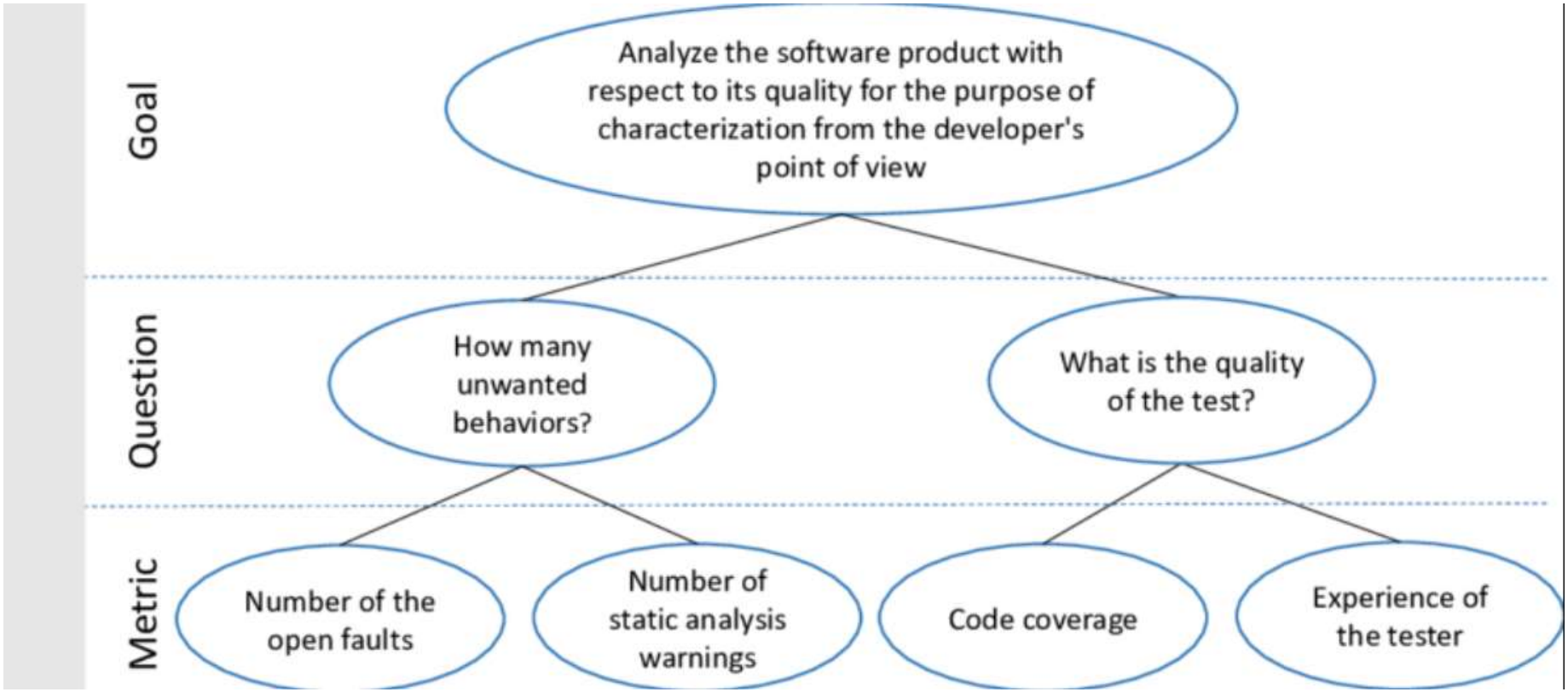

Goal
Analyze the software product with respect to its quality for the purpose of characterization from the developer's point of view
Question
How many unwanted behaviors?
What is the quality of the test?
Metric
Number of the open faults
Number of static analysis warnings
Code coverage
Experience of the tester

Table 1. List of usability smells and associated refactorings

Smell Id	Usability Smell	Abstract Usability Smell(s)	Refactorings
1	Undescriptive Element	User Confusion	Rename Element / Change Widget
2	Misleading Link	User Confusion	Rename Anchor
3	No Processing Page	Premature Abandonment	Add Processing Page
4	Free Input For Limited Values	Risk of Error / Activity too Long / Frequent Empty Results	Add Autocomplete / Change Widget
5	Unformatted Input	-	Change Widget
6	Short Input	User Confusion	Resize Element
7	Unnecessary Bulk Action	Activity Too Long	Distribute Menu
8	Overlooked Content	-	Split Page / Remove Redundant Content
9	Distant Content	User Distractions	Add Link
10	No Client Validation	Subsequent Failed Validations	Anticipate Validation
11	Late Validation	Subsequent Failed Validations	Anticipate Validation
12	Abandoned Form	Premature abandonment / Activity too Long	Split Activity / Postpone activity
13	Scarce Search Results	Frequent empty results.	Add Autocomplete
14	Useless Search Results	-	Add Autocomplete
15	Wrong Default Value	Unnecessary activities in the main process	Set Default Value
16	Unresponsive Element	Difficult Access to Information / Absence of Meaningful Navigation Links	Turn Attribute Into Link / Change Widget

What about Agile?

The "mantra" is to ensure that *users* and *user needs* are an integral part of the project req'ts and the team's development process.

Intent: Shared design responsibility for solving human interface design problems.

- Domain knowledge they have!
- Accomplishing their work with no interaction, so it works for them!
- Solving the design problem is problematic.

The important questions!

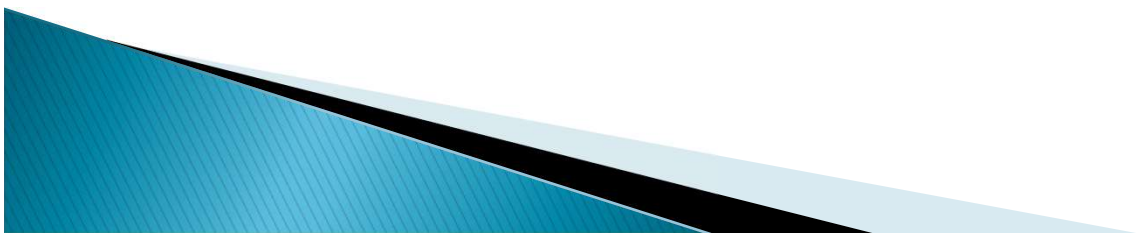
1. Who are the users?
2. What are my users trying to accomplish?
3. How do my users think about what they're trying to accomplish?
4. What kind of experiences do my users find appealing and rewarding?
5. How should my product behave?
6. What form should my product take?
7. How will users interact with my product?
8. How can my product's functions be most effectively organized?
9. How will my product introduce itself to first-time users?

.. more questions

10. How can my product put an understandable, appealing, and controllable face on technology?
11. How can my product deal with problems that users encounter?
12. How will my product help infrequent and inexperienced users understand how to accomplish their goals?
13. How can my product provided sufficient depth and power for expert users?

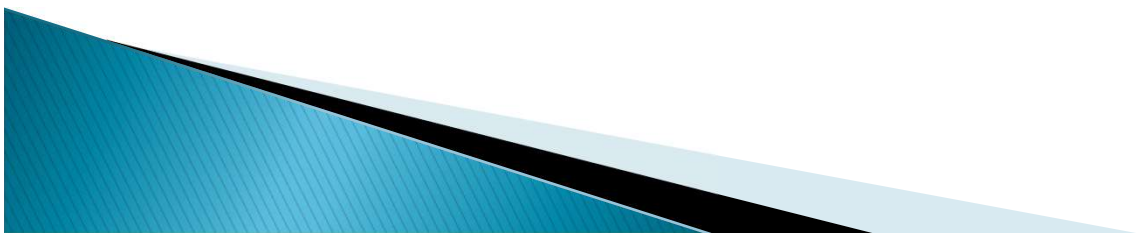
"The remainder of this book is dedicated to answering these questions."

Implementation Models and Mental Models



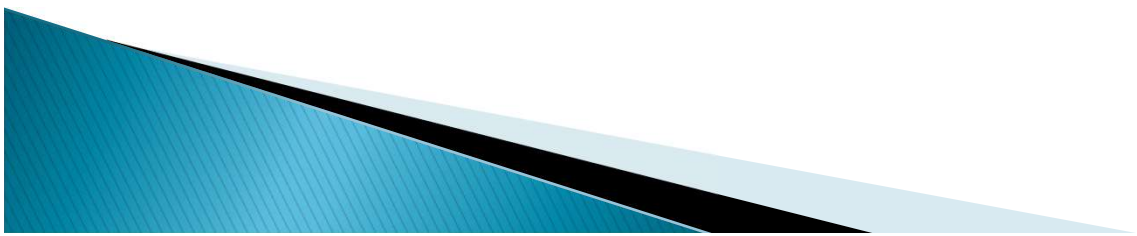
Implementation Models

- Any machine has a mechanism for accomplishing its purpose.
- A motion picture projector, for example, uses a complicated sequence of intricately moving parts to create its illusion. It shines a very bright light through a translucent, miniature image for a fraction of a second. It then blocks out the light for a split second while it moves another miniature image into place. Then it unblocks the light again for another moment. It repeats this process with a new image 24 times per second.



Implementation Models

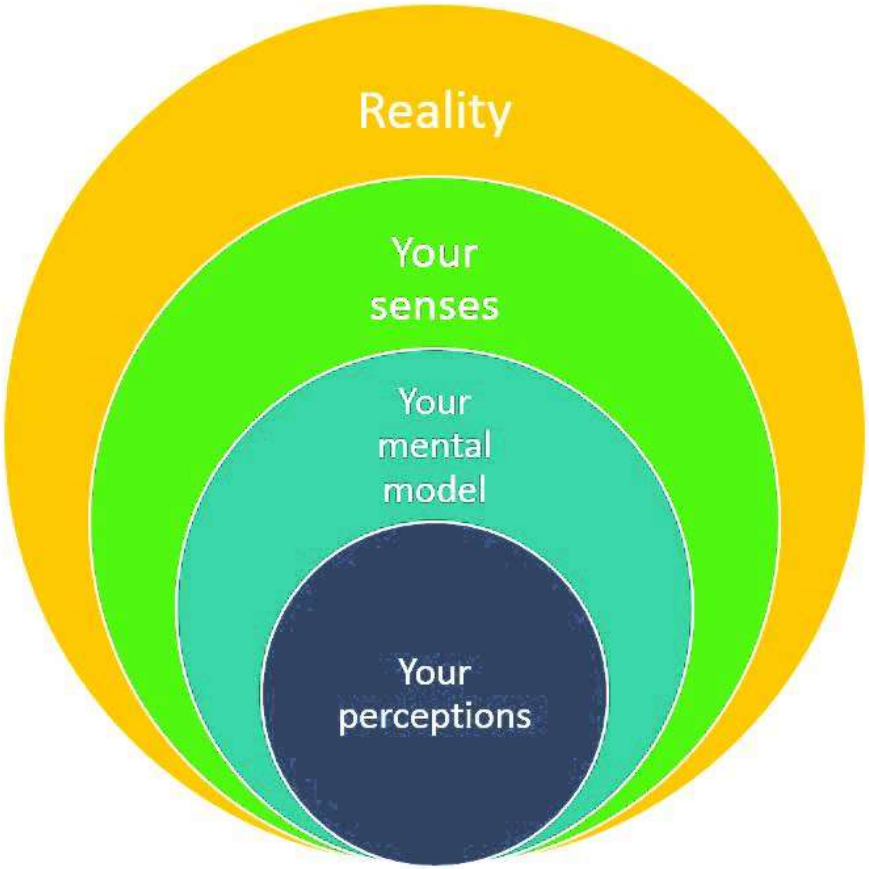
- Software-enabled products don't have mechanisms in the sense of moving parts;
- These are replaced with algorithms and modules of code that communicate with each other.
- The representation of how a machine or a program actually works has been called the **system model** by Donald Norman and others;
- we prefer the term **implementation model** because it describes the details of the way a program is implemented in code.

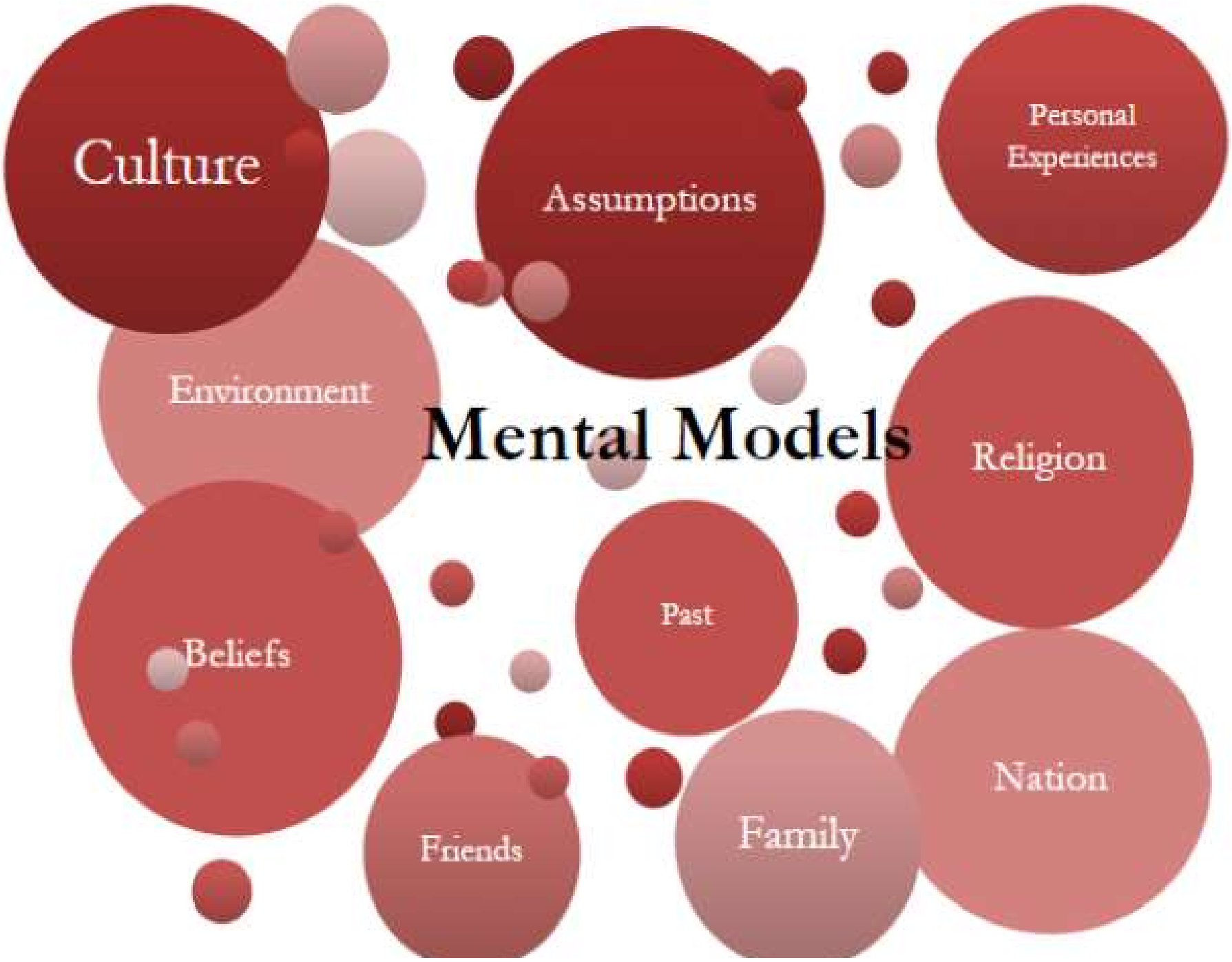


User Mental Models

- The viewer imagines that the projector merely throws a picture that moves onto the big screen. This is called the user's **mental model**, or **conceptual model**.
- People don't need to know all the details of how a complex mechanism actually works in order to use it,
- For example, many people imagine that, when they plug their vacuum cleaners and blenders into outlets in the wall, the electricity flows like water from the wall to the appliances through the little black tube of the electrical cord. This mental model is perfectly adequate for using household appliances. although the power company needs to know the details.

User Mental Models




Culture
Assumptions
Personal Experiences
Environment
Mental Models
Religion
Beliefs
Past
Nation
Friends
Family

User's Mental Model and the Implementation Model

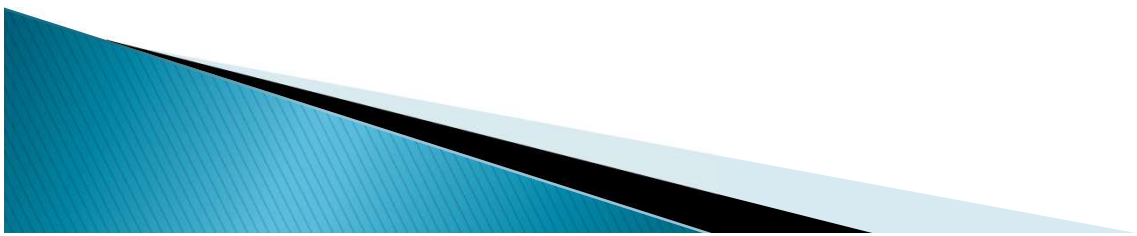
- In the digital world, however, the differences are often quite distinct.
- Our cellular telephone doesn't work like a landline phone; instead, it is actually a radio transceiver that might swap connections between a half-dozen different cellular base antennas in the course of a two-minute call
- When we use a computer to digitally edit sound or to create video special effects like morphing, we are bereft of analogy to the mechanical world,
- So our mental models are necessarily different from the implementation model. Even if the connections were visible, they would remain inscrutable to most people.

Represented Models

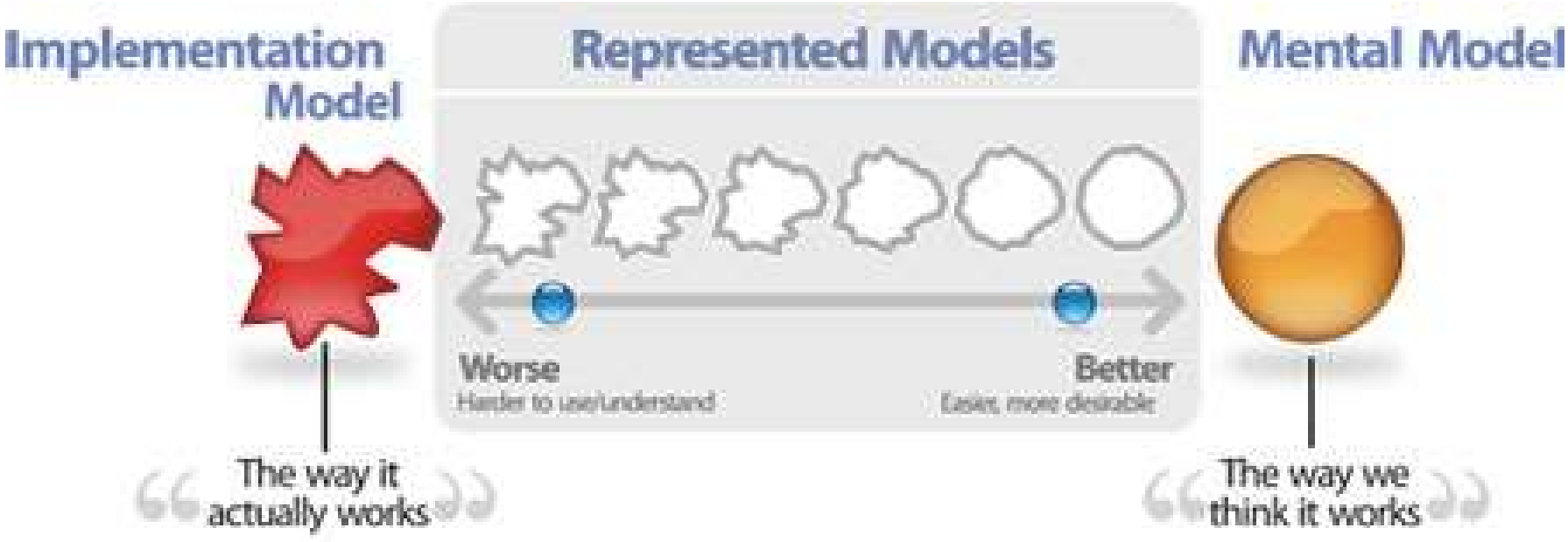
- The software designer has the ability to *represent* the computer's functioning independent of its true actions
- This disconnection between what is implemented and what is offered as explanation gives rise to a *third* model in the digital world, the designer's **represented model**— the way the designer chooses to represent a program's functioning to the user.
- Donald Norman refers to this simply as the **designer's model.**
- In the world of software, a program's represented model can (and often should) be quite different from the actual processing structure of the program.
- For example, an operating system can make a network file server look as though it were a localdisk. The model does not represent the fact that the physical disk drive may be miles away.

Represented Models

- The closer the represented model comes to the user's mental model, the easier he will find the program to use and to understand.
- Generally, offering a represented model that follows the implementation model too closely significantly reduces the user's ability to learn and use the program,

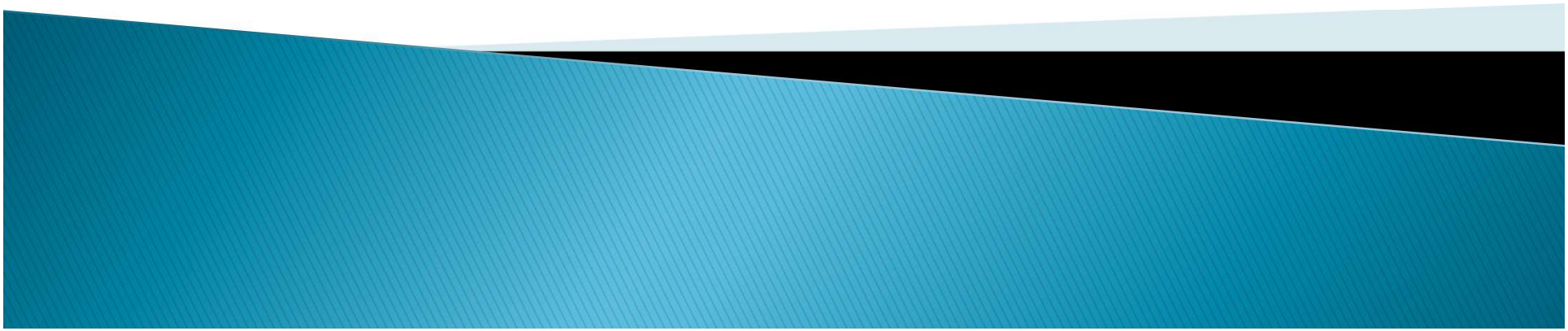


Represented Models

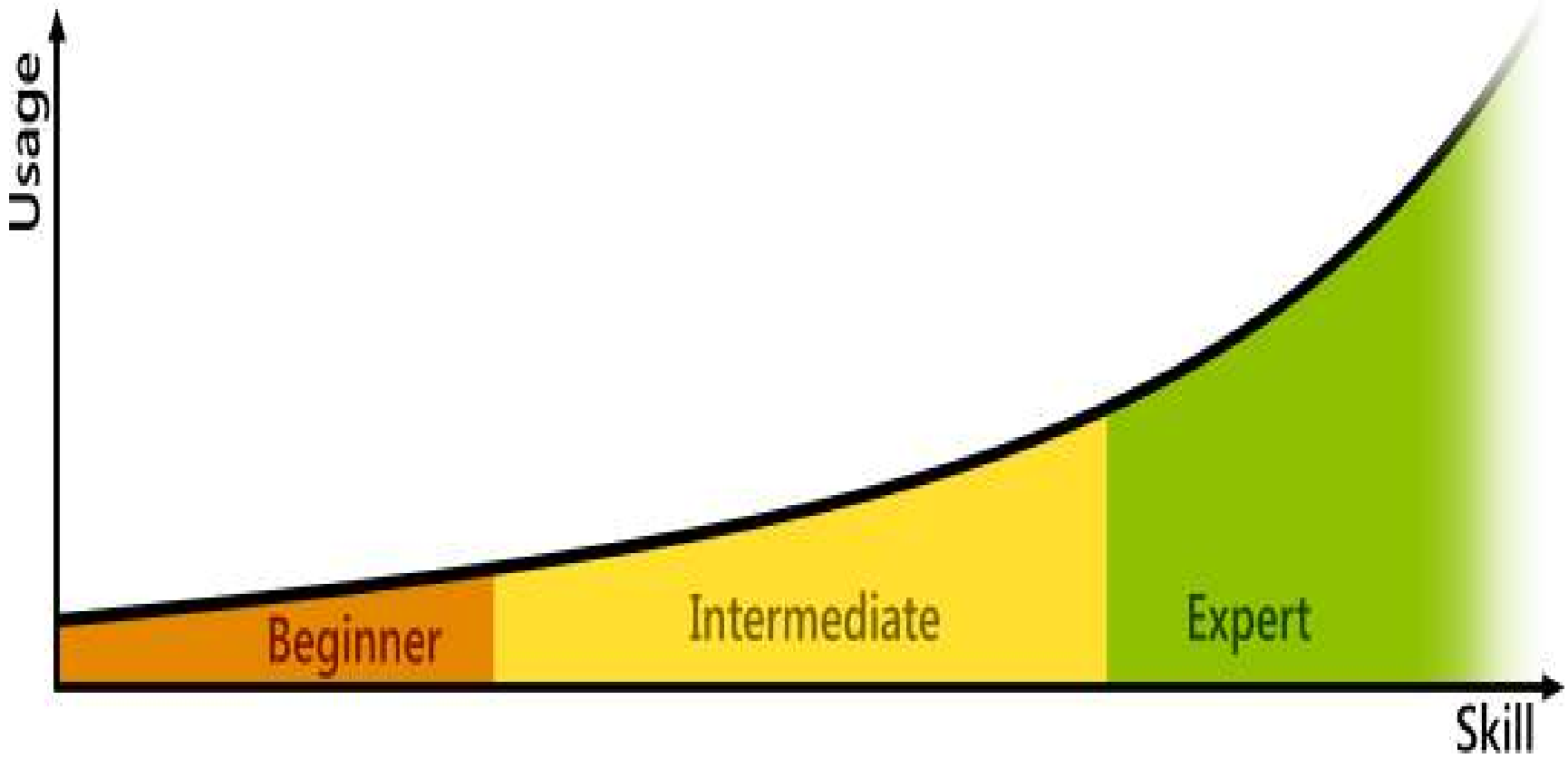


Derived from diagram in About Face 2.0 (by **Alan Cooper**)

Beginners, Experts, and Intermediates



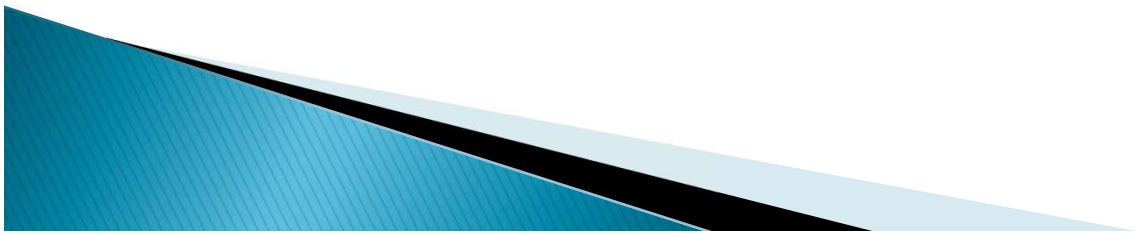
User Levels



Experienced User….

- Many experienced users are frustrated because that product always treats them like rank beginners.
- It seems impossible to find the right balance between the needs of the first-timer and the needs of the expert.
- One of the eternal conundrums of interaction and interface design is how to address the needs of both beginning users and expert users with a single, coherent interface.
- Some programmers and designers choose to abandon this idea completely,
  - segregate the user experiences by creating wizards for beginners
  - Bury critical functionality for experts deep in menus.

- The solution to this predicament lies in a different understanding of the way users master new concepts and tasks.



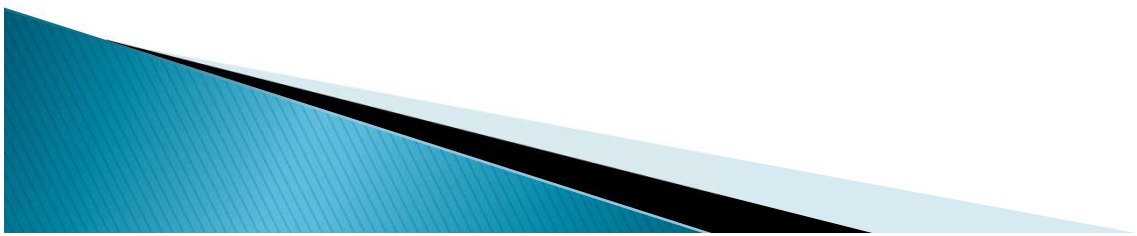
perpetual intermediates

Nobody wants to remain a beginner.

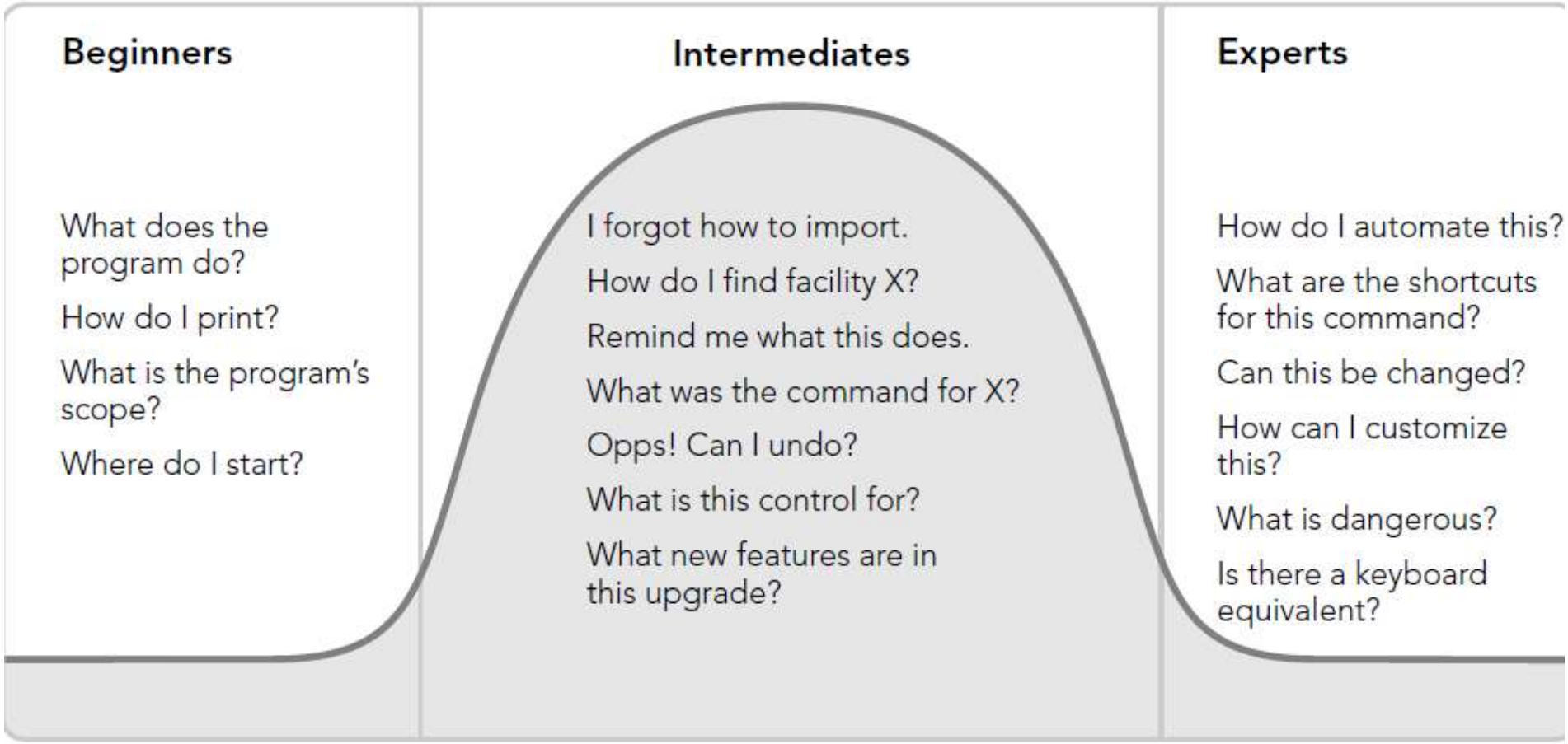
We prefer the term **perpetual intermediates** to **improving intermediates**, because although beginners quickly improve to become intermediates, they seldom go on to become experts.

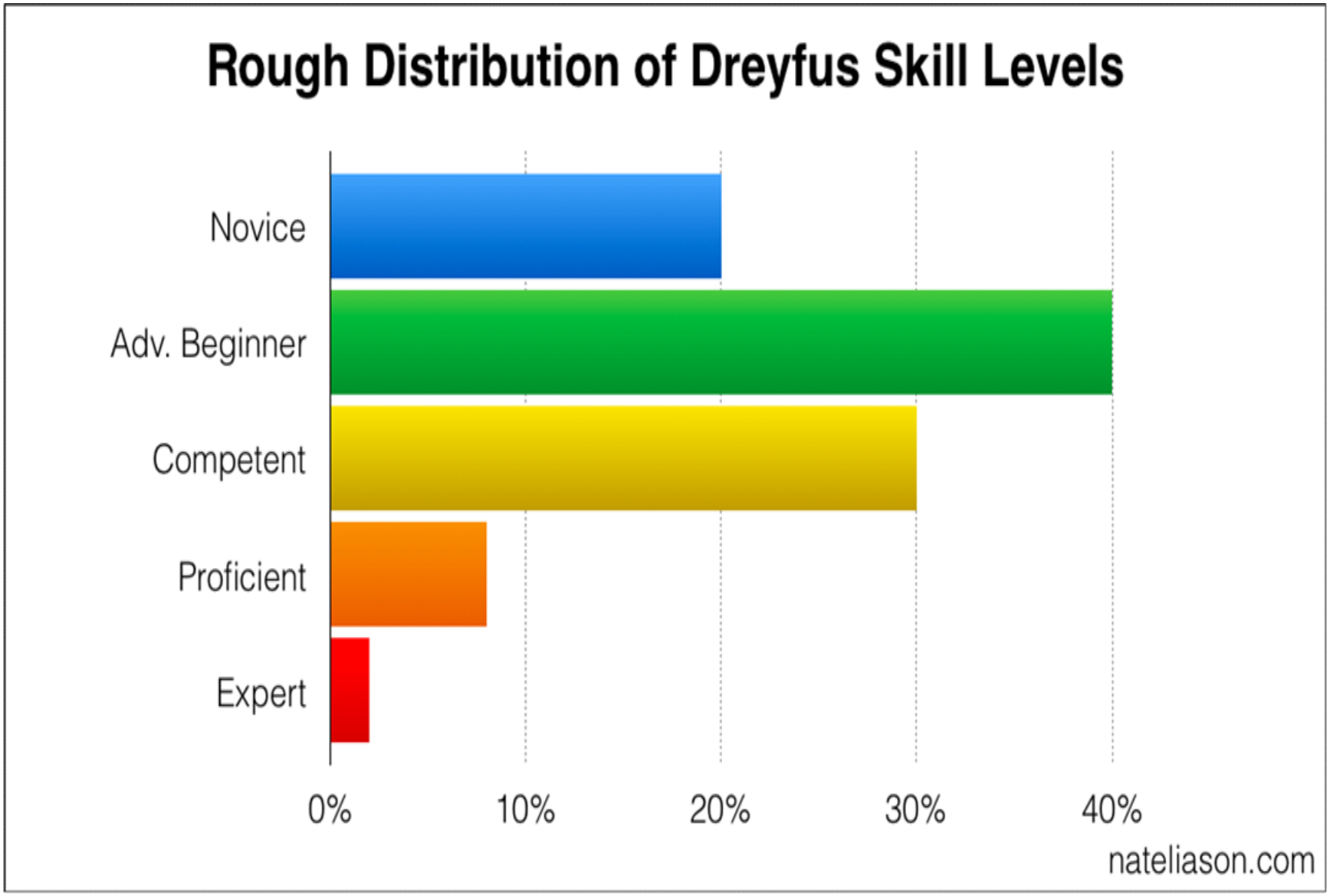
Perpetual Intermediates

- Most users are neither beginners nor experts; instead, they are *intermediates*.
- The experience level of people performing an activity tends, like most population distributions, to follow the classic statistical bell curve (see Figure 3-1).
- On a graph of people against skill level, a relatively small number of beginners are on the left side, a few experts are on the right, and the majority— intermediate users — are in the center.



Experience levels




Rough Distribution of Dreyfus Skill Levels
Novice
Adv. Beginner
Competent
Proficient
Expert
0%
10%
20%
30%
40%
nateliason.com

Designing for Different Experience Levels

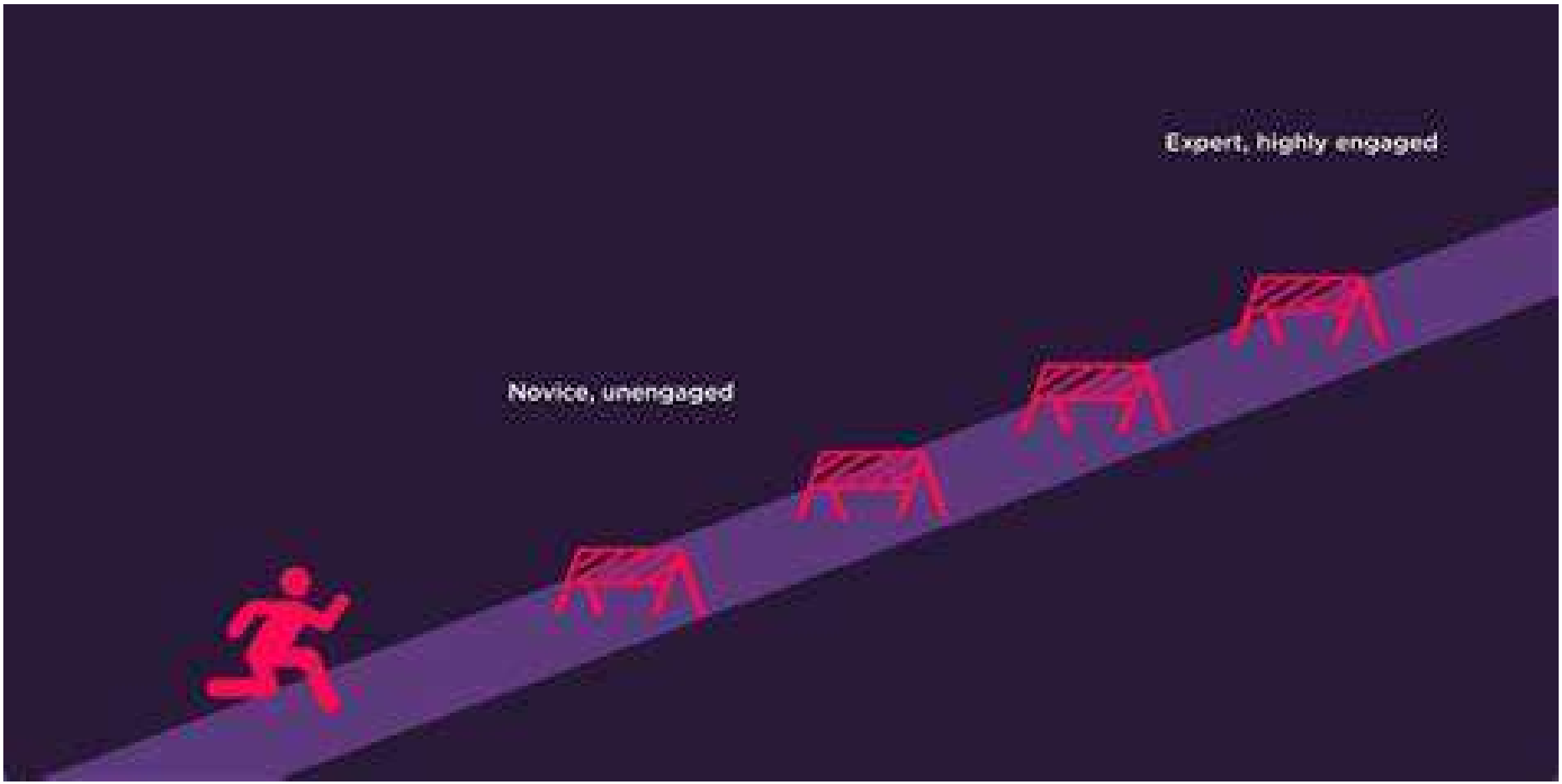
- **Programmers qualify as experts** in the software they code because they have to explore every possible use case. Their natural tendency is to design implementation model software ,which they, as experts, have no problem understanding.

- At the same time, **sales, marketing, and management** Because of their constant exposure to beginners, have a strongly biased view of the user community. They lobby for bending the interface to serve beginners.

- Programmers create interactions suitable only for experts, while the marketers demand interactions suitable only for beginners, but— as we have seen — the largest, most stable, and most important group of users is the **intermediate group.**

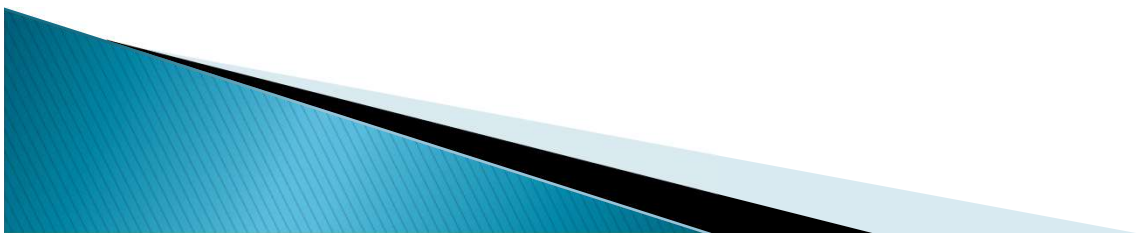
Optimize for intermediates

- Neither to pander to beginners nor to rush intermediates into expertise.
- The goal is threefold:
  - To rapidly and painlessly get beginners into intermediacy,
  - To avoid putting obstacles in the way of those intermediates who want to become experts, and
  - Most of all, to keep perpetual intermediates happy as they stay firmly in the middle of the skill spectrum.
- Need to spend more time making our products powerful and easy to use for perpetual intermediate users.
- Accommodate beginners and experts, too, but not to the discomfort of the largest segment of users


Expert, highly engaged
Novice, unengaged

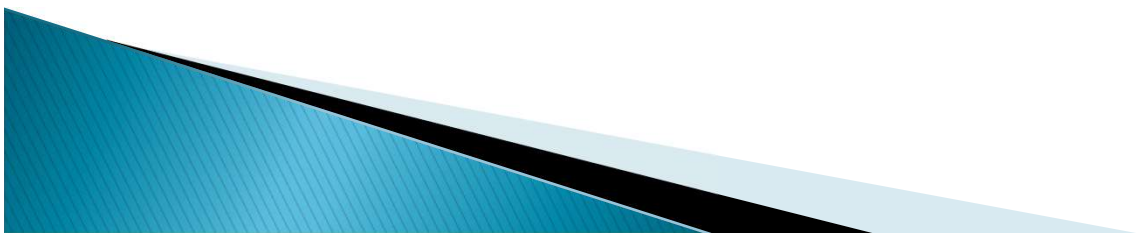
What beginners need

- Beginners are undeniably sensitive, and it is easy to demoralize a first-timer.
- Nobody wants to remain a beginner. It is merely a rite of passage everyone must experience.
- Good software shortens that passage without bringing attention to it.
- Imagine that users — especially beginners —are simultaneously very intelligent and very busy. They need some instruction, but not very much, and the process has to be rapid and targeted.



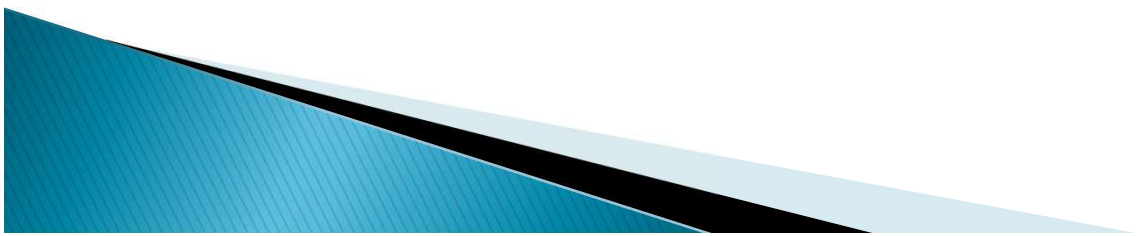
What beginners need

- If a ski instructor begins lecturing on snowpack composition and meteorology, he will lose his students regardless of their aptitude for skiing. Just because a user needs to learn how to operate a product doesn't mean that he needs or wants to learn how it works inside.



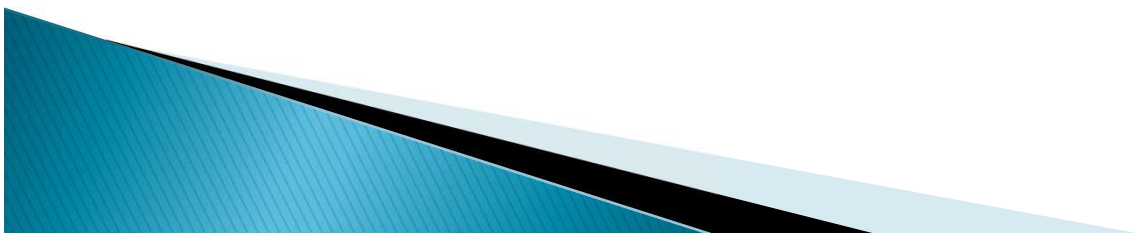
What experts need

- Experts have a disproportionate influence on less experienced users. When a prospective buyer considers your product, he will trust the expert's opinion more than an intermediate's.
- If the expert says, "It's not very good," she may mean "It's not very good for experts."
- The beginner doesn't know that, however, and will take the expert's advice, even though it may not apply.



What experts need

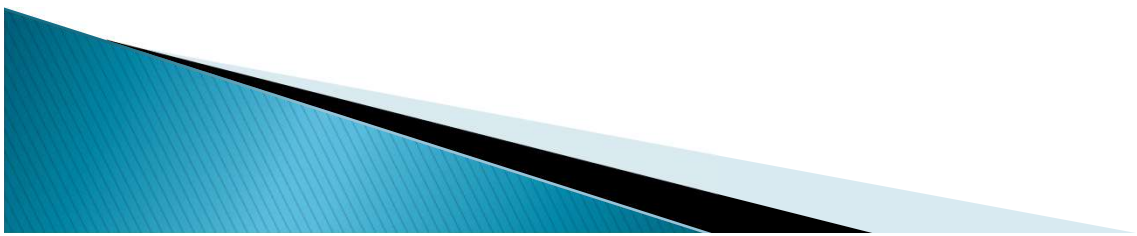
- Experts might occasionally look for esoteric features, and they might make heavy use of a few of them.
- However, they will definitely demand faster access to their regular working set of tools, which may be quite large.



What experts need

- In other words, experts want shortcuts to everything.

- Expert users constantly, aggressively seek to learn more and to see more connections between their actions and the product's behavior and representation.
- Experts appreciate new, powerful features. Their mastery of the product insulates them from becoming disturbed by the added complexity.

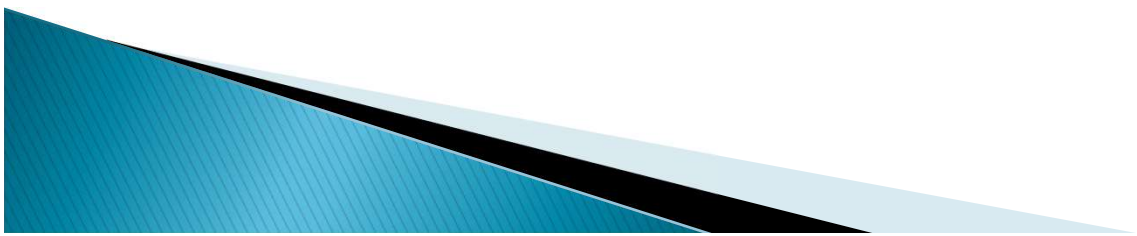


What perpetual intermediates need

- Perpetual intermediates need access to tools.They don't need scope and purpose explained to them.
- ToolTips are the perfect perpetual intermediate idiom.
- Perpetual intermediates know how to use reference materials.
- They are motivated to dig deeper and learn, as long as they don't have to tackle too much at once.
- Online help is a perpetual intermediate tool. They use it by way of the index, so that part of help must be very comprehensive.

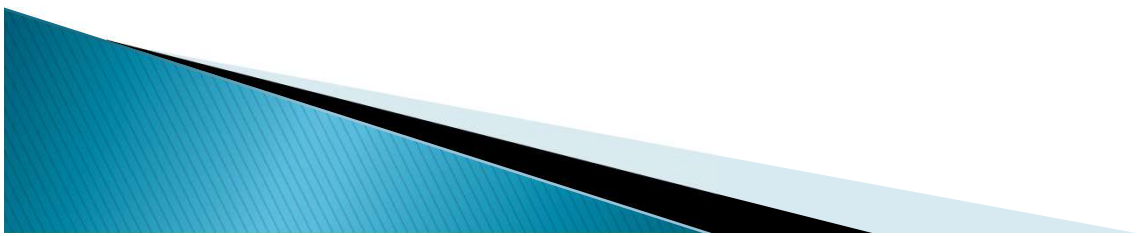
What perpetual intermediates need

- Perpetual intermediates establish the functions that they use with regularity and those that they only use rarely.
- The user may experiment with obscure features, but he will soon identify — probably subconsciously— his frequently used working set.
- The user will demand that the tools in his working set be placed front and center in the user interface, easy to find and to remember.

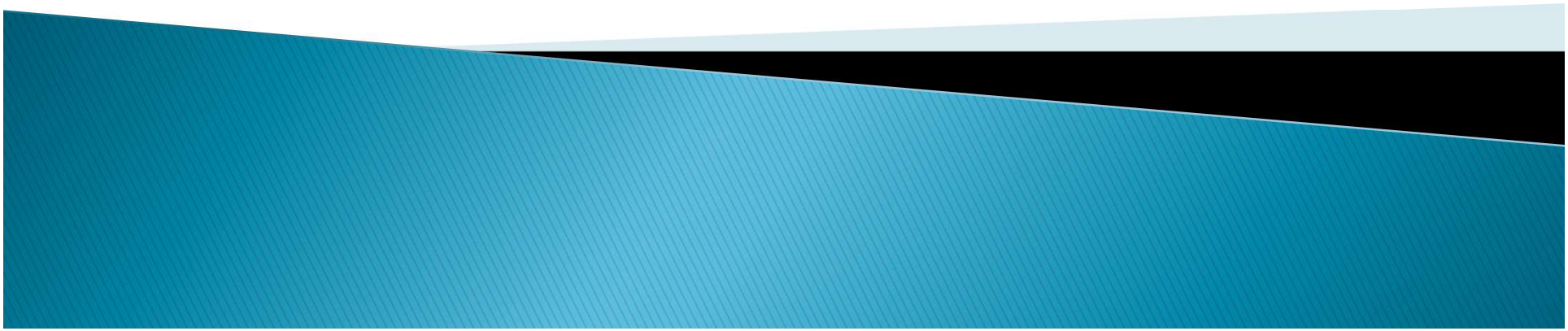


What perpetual intermediates need

- Perpetual intermediates usually know that advanced features exist, convincing him that he made the right investment.
- Your product's code must provide for both rank amateurs and all the possible cases an expert might encounter.
- You must provide those features for expert users and support for beginners.

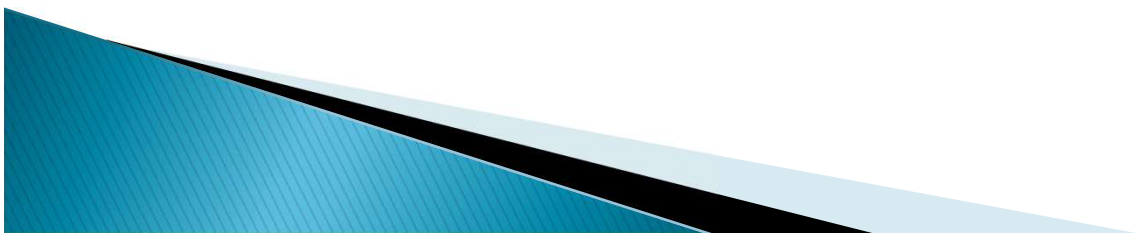


Understanding Users: Qualitative Research

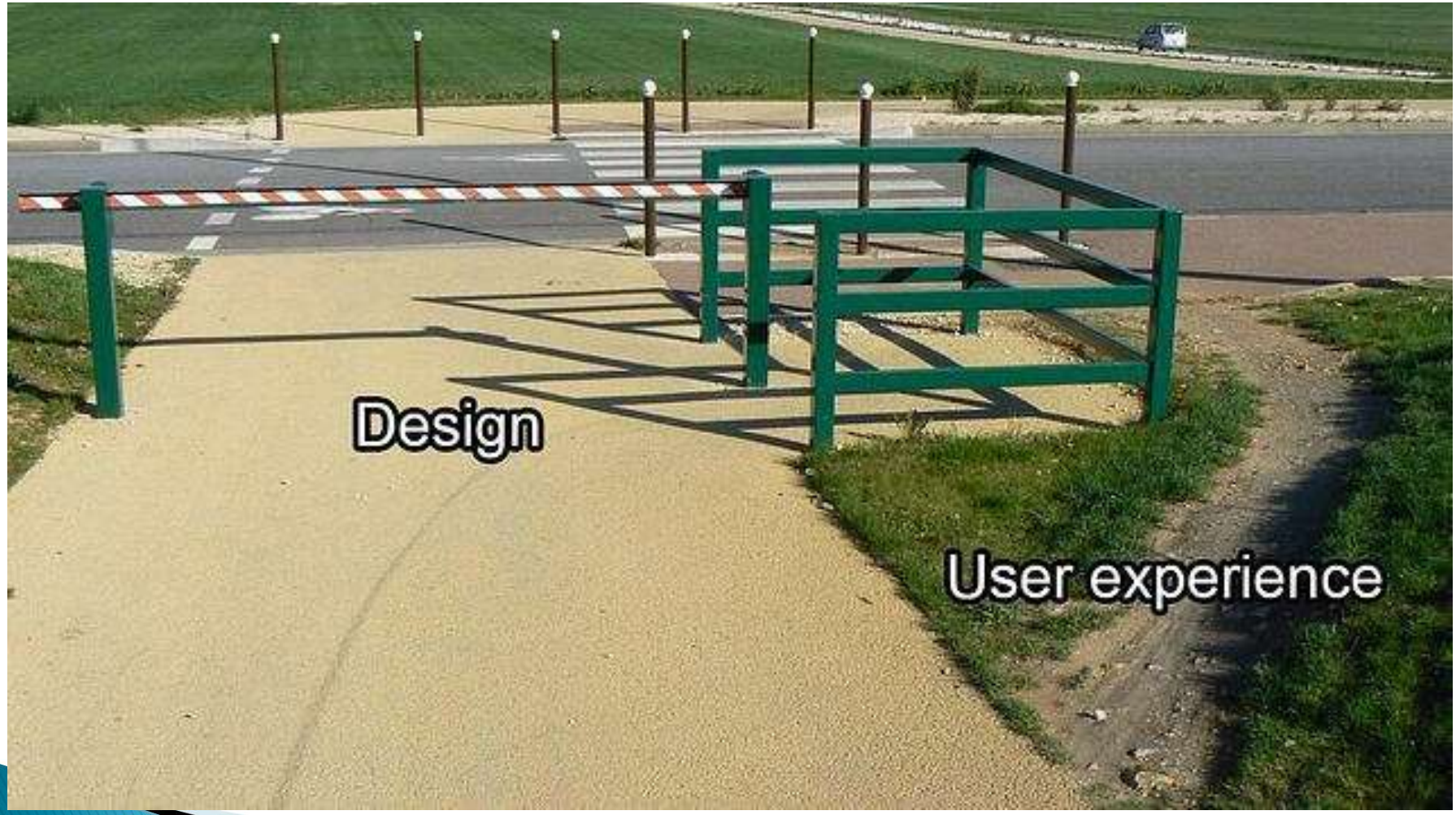


Understanding Users: Qualitative Research

- The outcome of any design effort is judged by how successfully it meets the needs of both the product user and the organization that commissioned it.
- If designer does not have clear and detailed knowledge of the users , the constraints of the problem and the business or organizational goals that are driving design activities, she will have little chance of success.



Design versus User Experience



F

RESEARCH

RESEARCH METHODS + THEORY

QUANTITATIVE

QUALITATIVE

Qualitative Research

- Real insight into user needs can't from a quantitative study like a market survey (though these can be critical for answering other kinds of questions).
- Rather, this kind of deep knowledge can only be achieved by *qualitative* research techniques.
- There are many types of qualitative research, each of which can play an important role in understanding the design landscape of a product.

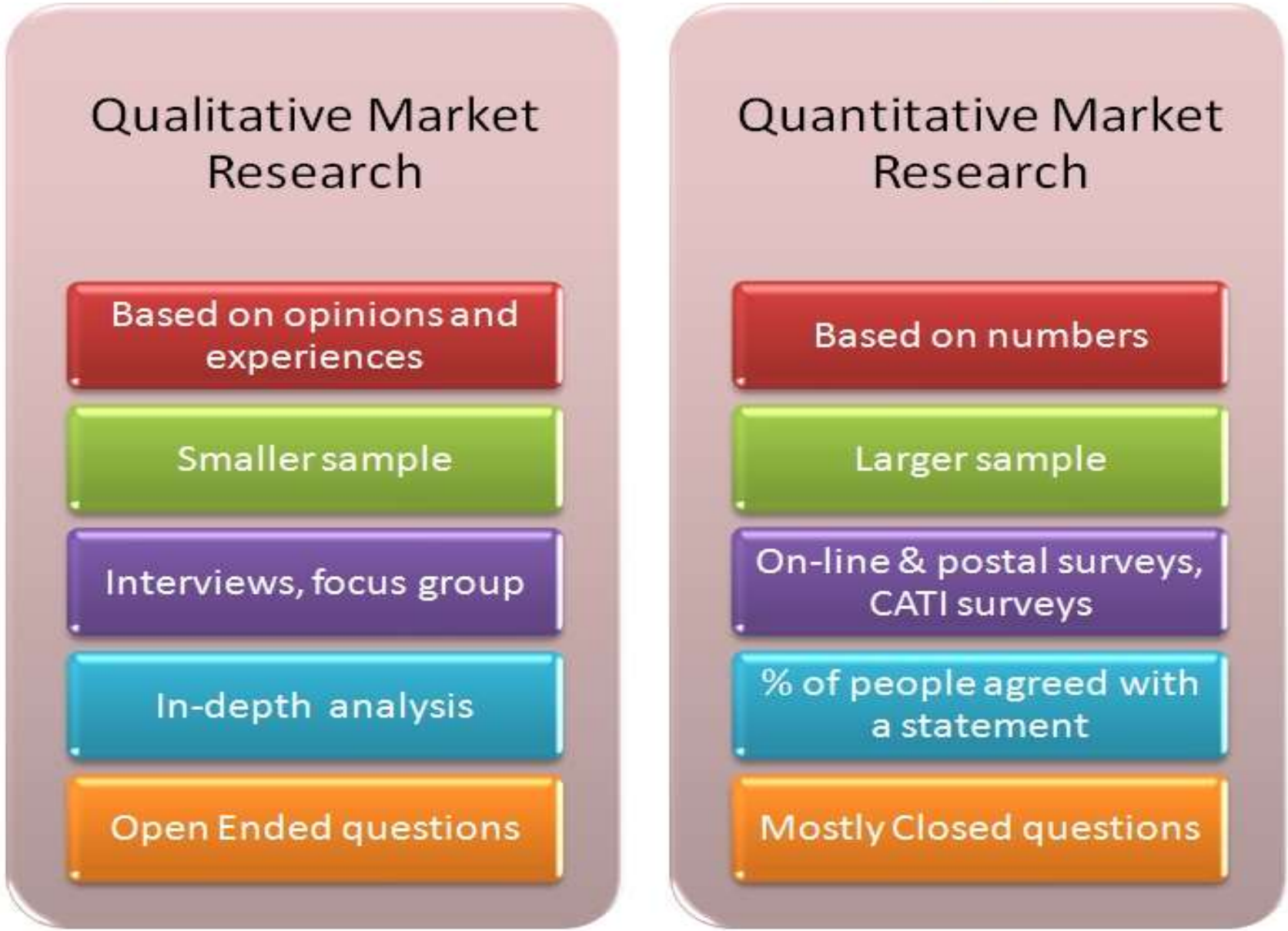
Qualitative versus Quantitative Research

- Research is not the only kind that yields the supposed ultimate in objectivity or quantitative data.
- numbers—especially statistics describing human activities—are subject to interpretation and can be manipulated at least as dramatically as words.
- Electrons don't have moods that vary from minute to minute,
- Any attempt to reduce human behavior to statistics is likely to overlook important nuances, which can make an enormous difference to the design of products.

Qualitative versus Quantitative Research

- Quantitative research can only answer questions about “how much” or “how many” along a few reductive axes.
- Qualitative research can tell you about what, how, and why in rich detail that is reflective of the actual complexities of real human situations.
- Social scientists have long realized that human behaviors are too complex and subject to too many variables to rely solely on quantitative data to understand them.

Type of Market Research: Qualitative vs. Quantitative



The value of qualitative research

- Qualitative research helps us understand the domain, context, and constraints of a product in different, more useful ways than quantitative research does.
- Helps us identify patterns of behavior among users and potential users of a product much more quickly and easily than would be possible with quantitative approaches.
- In particular, qualitative research helps us understand:
  - Behaviors, attitudes, and aptitudes of potential product users
  - Technical, business, and environmental contexts — the domain— of the product to be designed
  - Vocabulary and other social aspects of the domain in question
  - How existing products are used

Qualitative research can also help

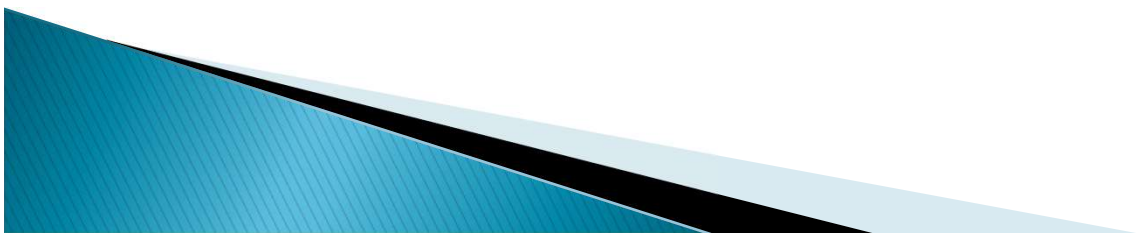
- Provides credibility and authority to the design team, because design decisions can be traced to research results
- Uniting the team with a common understanding of domain issues and user concerns
- Empowering management to make more informed decisions about product
- Design issues that would otherwise be based on guesswork or personal preference

qualitative methods tend to be faster, less expensive to answer…

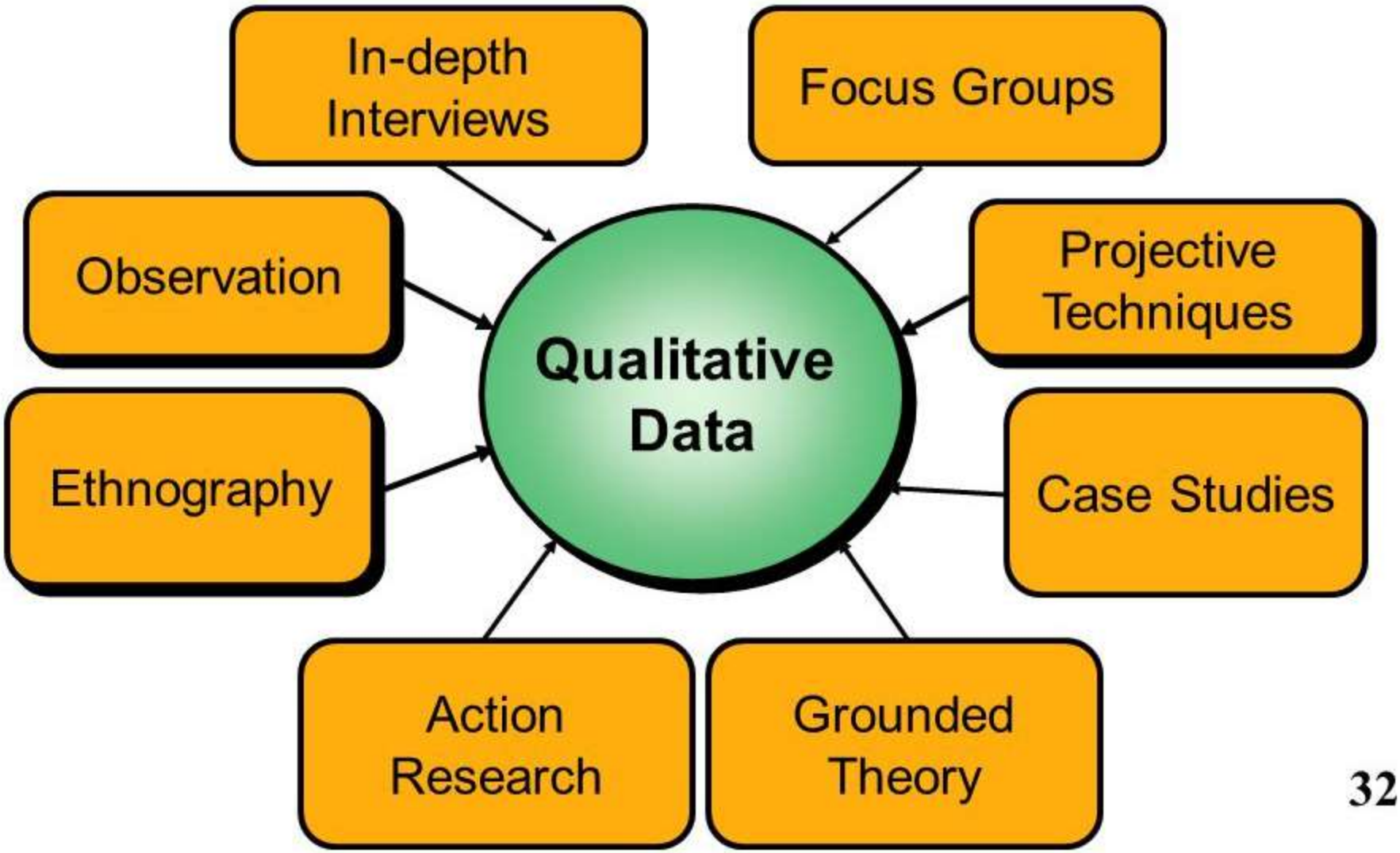
- How does the product fit into the broader context of people's lives?
- What goals motivate people to use the product, and what basic tasks help people accomplish these goals?
- What experiences do people find compelling? How do these relate to the product being designed?
- What problems do people encounter with their current ways of doing things?

Types of qualitative research

- The qualitative research activities we have found to be most useful in our practice are:
- Stakeholder interviews
- Subject matter expert (SME) interviews
- User and customer interviews
- User observation/ethnographic field studies
- Literature review
- Product/prototype and competitive audits

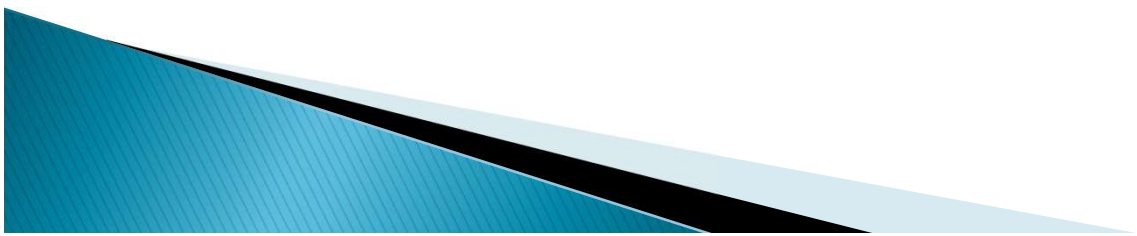


Qualitative Research Designs



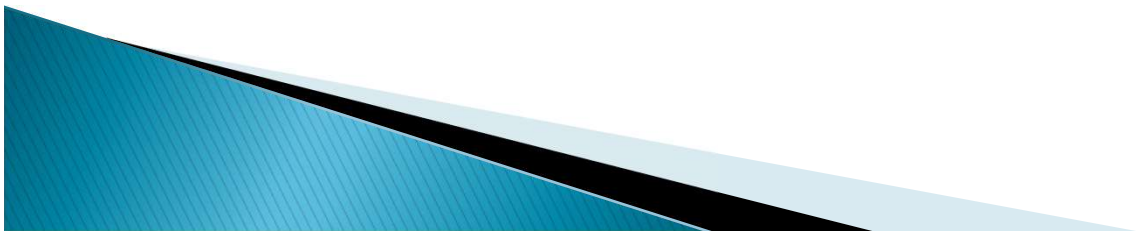
Ethnography

- Ethnography is a design of inquiry coming from anthropology and sociology in which the researcher studies the shared patterns of behaviors, language, and actions of an intact cultural group in a natural setting over a prolonged period of time. Data collection often involves observations and interviews.
- An ethnographic study could explore the **work culture of software developers in tech startups**. The researcher would immerse themselves in the day-to-day activities of a startup, observing coding practices, team dynamics, and communication styles. The outcome would reveal insights into how startup culture influences productivity, innovation, and work-life balance among developers.



Example of Ethnography:

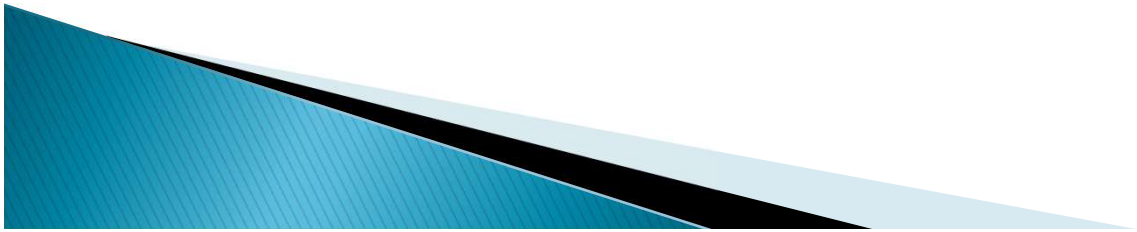
- In **Human-Computer Interaction (HCI)** research, an ethnographer might observe software engineers in their workplace over several months to understand how they collaborate when developing open-source software. The goal is to explore cultural norms, communication patterns, and how tools (like version control systems) are used in everyday practice. Ethnography can reveal insights into the "culture of coding," informal knowledge-sharing mechanisms, or even resistance to adopting new technologies within development teams.



Grounded Theory

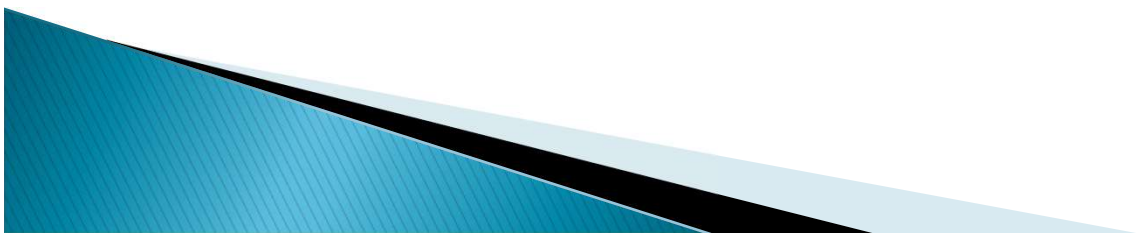
- Grounded theory is a design of inquiry from sociology in which the researcher derives a general, abstract theory of a process, action, or interaction grounded in the views of participants. This process involves using multiple stages of data collection and the refinement and interrelationship of categories of information (Charmaz, 2006; Corbin & Strauss, 2007).

•**Objective**: To generate a theory explaining how and why developers adopt or resist new programming tools and frameworks.
•**Method**: Analyzing interviews and surveys with developers about their experiences with new tools, focusing on decision-making processes and perceived benefits or barriers.
•**Outcome**: A theory explaining the stages of tool adoption, including factors like learning curves, community support, and perceived utility.



Example of Grounded Theory:

- In the field of **software development** or **user experience (UX)**, a researcher might interview a series of developers or users about their experiences with a particular tool or system. Instead of starting with predefined theories, they would analyze the data from these interviews to uncover recurring themes, such as “frustrations with debugging tools” or “strategies for managing cognitive load in complex interfaces.” As more data is collected, the theory about how developers manage complexity in software design, for example, would emerge directly from these findings.



Narrative research

- **Narrative research** is a design of inquiry from the humanities in which the researcher studies the lives of individuals and asks one or more individuals to provide stories about their lives (Riessman, 2008). This information is then often retold or restoried by the researcher into a
- narrative chronology. Often, in the end, the narrative combines views from the participant's life with those of the researcher's life in a collaborative narrative (Clandinin & Connelly, 2000).

•**Objective**: To explore and understand the personal and professional journeys of women in software engineering, focusing on the challenges, successes, and strategies they use to navigate their careers in a male-dominated field.
•**Methodology**:
•**Participants**: 10 women who have been working in software engineering roles for at least five years.
•**Data Collection**: In-depth, semi-structured interviews are conducted with each participant, encouraging them to share their life stories related to their career paths. Key prompts might include:

- •How did you first become interested in computer science or software engineering?
- •Can you describe some of the key challenges you have faced in your career?
- •How have you overcome these challenges?
- •How has your identity as a woman influence your career in tech?