Case Study: Refining Initial Expectations – "Forgot Password" in Software Development

In software projects, stakeholders often express needs in vague, shorthand ways that do not fully capture the real requirement. A common example is the request for a *"forgot password"* button on the login page. Taken literally, this request makes little sense—passwords cannot be "remembered" by the system. What the user truly wants is a secure and simple way to regain access when their credentials are lost.

This gap between *stated request* and *true expectation* is exactly where Expectation-Driven Design (EDD) demonstrates its strength. EDD begins by acknowledging the initial expectation, then systematically refines it by asking: *What outcome is actually desired? What steps must exist to make that outcome reliable, measurable, and secure?*

From General Request to Concrete Expectation

Initial Expectation (Owner's request):
"Add a *Forgot Password* function to the login page."

Too vague. It describes a button, not an outcome.

Refined Expectation (after Product Owner and team refinement):
"As a user, I want to enter my email address and receive a link to securely set a new password."

Further Clarified Expectations:

- Security: Never reveal whether an email exists in the system. Always return the same confirmation message.
- Validity: Reset links expire after 72 hours.
- Usability: Password reset form requires double entry to prevent mistakes.
- Integrity: Links become invalid once used.
- Resilience: Requests are rate-limited (max 3/minute per address) to prevent abuse or denial-of-service attacks.
- Simplicity: Identification via email only, not usernames.

High-Level User Flow

1. User clicks *Forgot Password*.
2. System presents email entry form.
3. Input validated → confirmation message displayed.
4. Backend generates secure, time-limited reset link.
5. Email server delivers reset email.

6. User clicks link → system validates link and expiration.
7. User sets new password (double entry).
8. System saves new password, invalidates link.
9. Expectation fulfilled: the user regains account access.

This flow can be decomposed even further—for instance, email delivery involves multiple infrastructural steps (mail server notification, link signing, queue handling). Alternatively, the process can be grouped at a higher level (frontend, backend, infrastructure). EDD allows either approach depending on what level of granularity best supports validation and feedback.

Why EDD Matters Here

The *power* of Expectation-Driven Design lies in how it transforms ambiguous stakeholder language into a precise, testable design. Instead of coding "a button," the team builds a secure workflow that fulfills the *real* expectation: restoring account access without compromising security.

- **Bridging Language Gaps**: Owners may say "forgot password," but EDD uncovers the actual need—secure credential recovery.
- **Making Expectations Measurable**: Each clarified expectation (e.g., 72-hour link validity, rate limiting) becomes a measurable property of the system.
- **Aligning Perspectives**: Business owners think in terms of user satisfaction, developers think in terms of flows and validations, and security teams think in terms of attack surfaces. EDD provides a framework that aligns these perspectives by starting from the shared *end expectation*.
- **Designing Feedback Loops**: If too many reset requests come from a single source, monitoring tools can highlight the abuse and refine system defenses. If users complain about not receiving emails, the team knows which step in the expectation-flow requires adjustment.
- **Preventing Misinterpretation**: Without EDD, the project might deliver only a superficial feature (a button with unclear behavior). With EDD, the project delivers a secure, resilient recovery mechanism that truly fulfills the intended outcome.

Significance:
This case study illustrates that EDD is not just about documenting requirements—it is about *understanding the intent behind expectations* and reverse-engineering the processes to reliably meet them. In doing so, EDD helps teams move from vague requests toward precise, validated, and secure solutions that stand up in real-world conditions.