

The Software Teaches Toolkit

Help your users understand, recover and finish. Keep Gerald off the menu.

Start with one feature, not the whole product.

Pick a form, dashboard, approval, notification or API-driven workflow. Walk through a difficult path as a person who does not already know your system.

- Use the SURVIVE checklist to locate the uncertainty.
- Compare the before-and-after examples with your own interface.
- Rewrite one message so it explains the situation and a useful next step.
- Choose the communication level using the notification purpose matrix.
- Map the workflow, including who acts next and what happens when an action fails.
- Record one improvement and verify it through the difficult path again.

The teaching lens

- What does the user know at this point?
- What does the system expect from them?
- What could they misunderstand?
- What feedback would help them recover?
- What should they be able to do next?
- Is the product teaching one clear version of the rules?

If failure is possible, recovery is part of the feature.

The zombie incident examples are fictional teaching scenarios. Adapt the wording and next actions to what your own system actually supports.

The SURVIVE checklist

Seven prompts for reviewing a feature beyond its happy path.

S

STATE | Show me where I am

Distinguish loading, empty, filtered, no access and failed states. Give each a useful next step.

U

UNDERSTANDING | Unpack what this means

Use familiar words. Explain technical terms and what a message means for the user's task.

R

RULES | Remove the guesswork

Show requirements and useful examples before the attempt. Ask what applies, then show what applies.

V

VALIDATION | Show me what went wrong

Locate the problem and explain it specifically. Preserve the user's work.

I

INTERVENTION | Help me fix it

Offer a route forward: request access, resume after sign-in, or continue safely with missing details.

V

VISIBILITY | Tell the right people what changed

Match communication to importance. Distinguish sent, delivered, seen and acknowledged.

E

END STATE | Tell me how it ended

Show what succeeded, what failed, who acts next and whether the user needs to do anything.

If failure is possible, recovery is part of the feature.

Review one difficult path. Can the person still understand, recover and finish?

From guesswork to guidance

Illustrative rewrites for forms, dashboards, permissions, notifications and API outcomes.

Forms

Before: There were errors with your submission.

After: Select an incident location. For Lab Three, choose West Wing - Lab Three; we will fill the facility code and department for you.

Why: Explain success before the attempt. Derive facts the system already knows instead of testing whether users remember them.

Review: Do field guidance and backend rules describe the same requirement?

Dashboards

Before: A blank table with no explanation.

After: No incidents match these filters. Clear filters to see all incidents you can access.

Why: Filtered results are a different state from no incidents, no access, loading or a failed request.

Review: Can the user tell which state they are in and what to do next?

Permissions

Before: 403 Forbidden.

After: You do not have approval access. Request access or return to the list.

Why: Give an authorised route forward without revealing private record details.

Review: Is an access request actually available, and will sign-in or approval return the person to their task?

From guesswork to guidance

Illustrative rewrites for forms, dashboards, permissions, notifications and API outcomes.

Notifications

Before: Your request has been updated.

After: Your equipment request needs a decision. Review the request and approve or return it for changes.

Why: Name the change, the required action and its destination.

Review: Does this need history, a notification, a direct message, an urgent alert or escalation?

API responses and outcomes

Before: SUCCESS.

After: Report 2026-0847 was submitted. First Aid: delivered, awaiting acknowledgement. Containment doors: failed to lock. Evacuate using the west stairwell.

Why: A saved report is not proof that every downstream action succeeded. Represent each outcome separately.

Review: Does each message reflect a state the system can verify?

Adapt the pattern, not just the sentence.

- Use your users' words, then explain necessary technical terms.
- Only offer actions that exist. Do not promise saved work or retries unless implemented.
- Keep frontend guidance and backend rules aligned.
- Avoid leaking private records, internal errors or security-sensitive details.

Make the error useful

Locate the problem. Explain the effect. Offer a way forward.

1. Identify the actual state

Is the input invalid, access missing, the service unavailable, or submission uncertain? These need different messages.

2. Say what happened in familiar words

Name the relevant field or action. Replace a generic failure with something the user can recognise.

3. Explain the impact on the task

Say whether the person can continue. If you cannot confirm whether submission succeeded, say that instead of guessing.

4. Provide a supported recovery route

Point to the field, access request, saved draft, original task or status check. Preserve valid input where the feature supports it.

5. Check truth and privacy

Do not expose private record details, stack traces or internal identifiers. Do not claim an alert was accepted when it was only sent.

6. Put the message where it can be used

Associate it with the relevant field or task. Give an error summary when several issues need attention; make it usable with a keyboard and assistive technology.

Example: "Enter a description of 500 characters or fewer. Your other answers are still here." Use the second sentence only if the form preserves those answers.

Rewrite one message

Use this worksheet with an error from your own product.

Original message / where it appears

What happened, and what can the system verify?

What does this mean for the user's task?

What supported action can the user take next?

Rewrite: situation + impact + next action

Check: specific, truthful, understandable, safe and actionable. Test the recovery route.

Choose the communication

The right truth, to the right people, at the right time.

Start with purpose and urgency. A history entry, notification and emergency alert are not interchangeable.

Level	Use when	Include / route	Response
History	Routine change; no immediate action	Record the change in the item timeline What changed, when, who/what changed it	No acknowledgement required
Notification	A user has something to review or do	Show a targeted in-app notification What changed, why it matters, action link	Track completion of the task
Direct message	A named person must coordinate or respond	Contact the responsible person through an agreed channel Context, owner, request, response expected	Require a reply where needed
Urgent alert	Immediate danger or time-critical action	Use the agreed urgent alert channel Risk, immediate action, location, owner	Request explicit acknowledgement
Escalation	Response is overdue or cannot proceed	Route to the next responsible role Original issue, elapsed time, prior attempts, action needed	Track acceptance by the new owner

Make delivery states explicit.

Sent -> Delivered -> Seen -> Acknowledged

Expose only states your system can verify. Reading a message is not the same as accepting responsibility. Choose owners, expected response times and escalation rules appropriate to your workflow.

Trace the user's journey

One workflow. Clear steps. Visible hand-offs.

User / goal / trigger

What does the user already know? What do they need to learn?

Map three important steps.

Step / action	What must be clear?	Feedback / state	Who acts next?
1.			
2.			
3.			

At each hand-off: show the stage, responsible role, when an update is expected and whether the user needs to act.

Map failure to intervention

Try the journey when something goes wrong.

Failure / interruption / missing information

What will the user see? What might they misunderstand?

Recovery action and how their work is preserved

Final outcome: what succeeded, failed or remains pending?

Next owner / expected update / escalation if overdue

One improvement / owner / evidence that it works

Review a real feature

Choose one journey. Find the uncertainty. Design a useful response.

Feature / user goal

Choose a difficult path

For example: missing permission, ambiguous search, timeout or partial failure.

S - State: Show me where I am

U - Understanding: Unpack what this means

R - Rules: Remove the guesswork

V - Validation: Show me what went wrong

Turn failure into a way forward

Continue the review, then choose one improvement to make.

I - Intervention: Help me fix it

V - Visibility: Tell the right people what changed

E - End state: Tell me how it ended

Failure map -> intervention map

When _____ happens, the user can _____; their work is preserved by _____.

One improvement / owner / how we will check it

Finish by walking through the difficult path again. Can the person understand, recover and finish?