WHITE PAPER

Automate ERPNext Without a Developer: Workflows, Alerts & Scheduled Jobs

The automation ERPNext ships with — approvals, alerts, auto-assignment and recurring documents — is configured in the browser, not written in code. Here's what's genuinely no-code, and where light scripting starts.

For operations & IT leaders · **8 min read**

Acube Innovations · Official ERPNext Partner · acubeinnovations.com · +91 62358 66111

EXECUTIVE SUMMARY

The most common myth about ERPNext is that every bit of automation needs a developer. It doesn't. ERPNext runs on the Frappe framework, and Frappe ships a set of automation tools that are configured entirely in the browser — no code, no deployment. This paper walks through the four you'll use most: Workflows (multi-level approvals with role-based states), Notifications (email and in-app alerts on document events), Assignment Rules (auto-distributing work round-robin or by load) and Auto Repeat (recurring documents on a schedule). Each is real, built-in Frappe functionality your own admins can set up. Then it draws the line honestly: where the no-code tools stop and light scripting — Server Scripts, Client Scripts and scheduled jobs — genuinely earns its place. The goal is to help operations and IT leaders see how much they can automate themselves, and to recognise the point where a developer or partner is the right call rather than a reflex.

The manual busywork that shouldn't be manual

Walk any operations team through their day and you'll find the same handful of tasks eating hours that no software should require a human for. A purchase order sits in someone's inbox waiting for a manager to eyeball it and reply "approved." A salesperson forgets to follow up because nothing reminded them a quotation was about to expire. Support tickets pile up in a shared queue while everyone assumes someone else will pick them up. The same journal entry gets keyed in by hand on the first of every month because "that's just how we do it."

None of this is a people problem — it's an automation gap. Each of these is a rule a computer can follow perfectly: if this document reaches this state, route it to that person; if this date is three days away, email a reminder; when a new ticket arrives, hand it to whoever has the fewest open. The reason it stays manual is usually a belief that automating it means a project — a developer, a budget, a change request.

For a large share of everyday automation in ERPNext, that belief is simply wrong. The tools to close these gaps are already in the system, configured through ordinary forms, by the same admins who manage users and permissions. This paper is about knowing which tool solves which problem — and being honest about the smaller set of cases where code really is the answer.

- Approvals waiting in inboxes — documents that can't move until a person manually signs off and tells the next person.
- Missed follow-ups and deadlines — renewals, expiries and due dates that depend on someone remembering.
- Work sitting unassigned — tickets, leads and tasks in a shared queue with no owner.
- Re-keying the same document — recurring invoices, journal entries and orders typed in from scratch every cycle.

The ERPNext automation toolkit — no-code vs light code

1. **Workflow (no code)**
   multi-level, role-based approvals as states and transitions; enforces who can move a document and notifies the next approver.

2. **Notification (no code)**
   email or in-app alerts triggered by document events, field changes, or days before/after a date, with a templated message.

3. **Assignment Rule (no code)**
   auto-distributes new documents to users by Round Robin, Load Balancing or a user field, so work always has an owner.

4. **Auto Repeat (no code)**
   regenerates recurring documents on a chosen frequency, optionally auto-submitting and emailing them.

5. **Server Script (light code)**
   Python for custom document-event logic, APIs or scheduled background jobs, entered in the UI but genuinely code.

6. **Client Script & Scheduled Jobs (light code)**
   JavaScript form behaviour and interval/cron logic for what the no-code tools can't express.

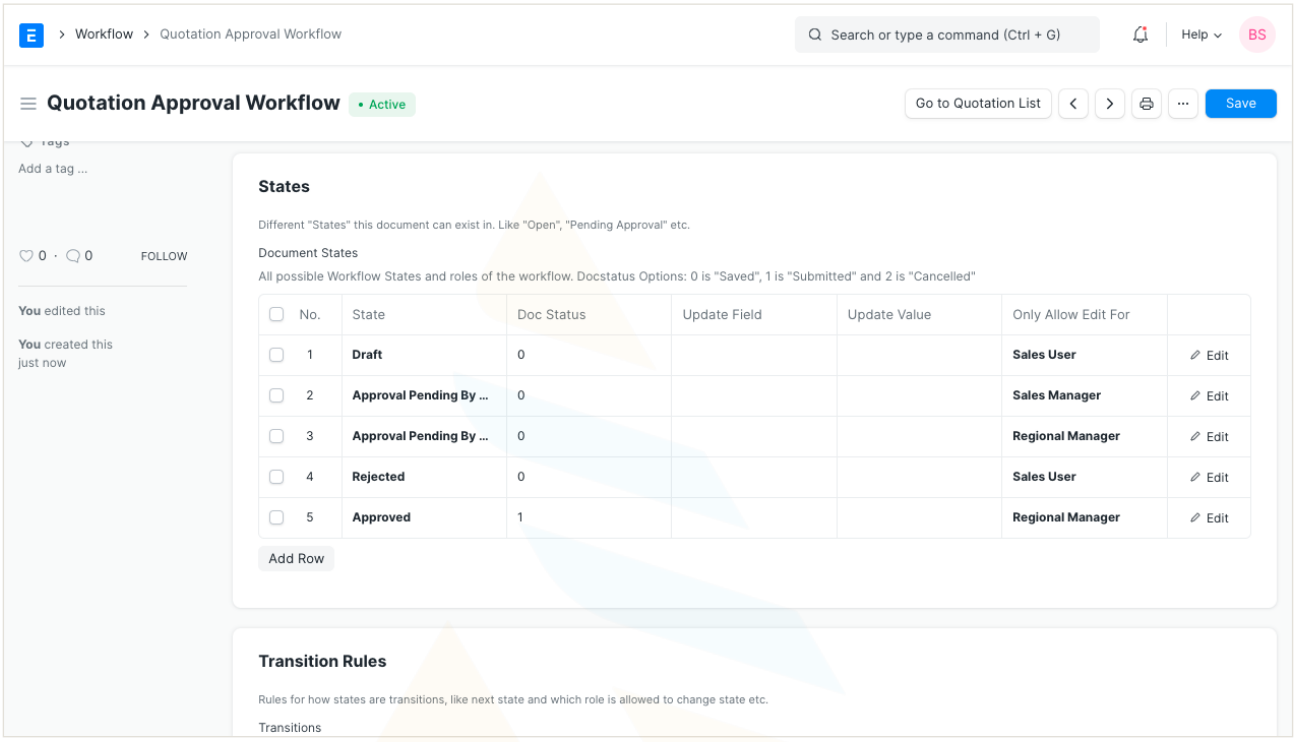
Workflows & approvals — no code

A Workflow is how you turn "email me and I'll approve it" into a controlled, auditable process — and in ERPNext it's built entirely by filling in a form. A Workflow is made of two things: states and transitions. States are the conditions a document can be in — Draft, Approval Pending, Rejected, Approved — and each state is tied to a document status (Saved, Submitted or Cancelled) and, crucially, to a role. The "Only Allow Edit For" setting on each state means a document sitting in "Approval Pending by Manager" can only be acted on by someone with the Manager role; everyone else sees it as read-only.

Transitions are the arrows between states: from Draft, a Sales User can move a quotation to Approval Pending; from there, a Sales Manager can move it to Approved or Rejected. You define who is allowed to make each move, so the approval chain is enforced by the system rather than by convention. ERPNext also emails the next approver their available actions, so nobody has to chase the document. All of this — states, roles, transitions, the approval chain — is set up in the Workflow screen without a line of code.

The one place light scripting can appear is conditional transitions: if you want "orders under a threshold skip manager approval," a transition can carry a short condition expression (for example, comparing the document's total to a number). That expression is a small piece of logic, not a program — but it's worth flagging honestly as the point where no-code shades into a formula. The core value — role-based, multi-level approvals with a full audit trail — needs none of it.

- States — the stages a document moves through (Draft, Pending, Approved, Rejected), each mapped to a status and an allowed-to-edit role.
- Transitions — who can move a document from one state to the next, so the approval chain is enforced, not assumed.
- Automatic notifications — the next approver is emailed the actions available to them, so nothing stalls in an inbox.
- Optional conditions — a short expression can branch the flow (e.g. small orders skip a level); the basic approval chain needs no expression at all.



A Workflow in ERPNext — each state (Draft, Approval Pending, Approved, Rejected) maps to a document status and the role allowed to act on it. Built entirely in this form, no code.

Notifications & alerts — no code

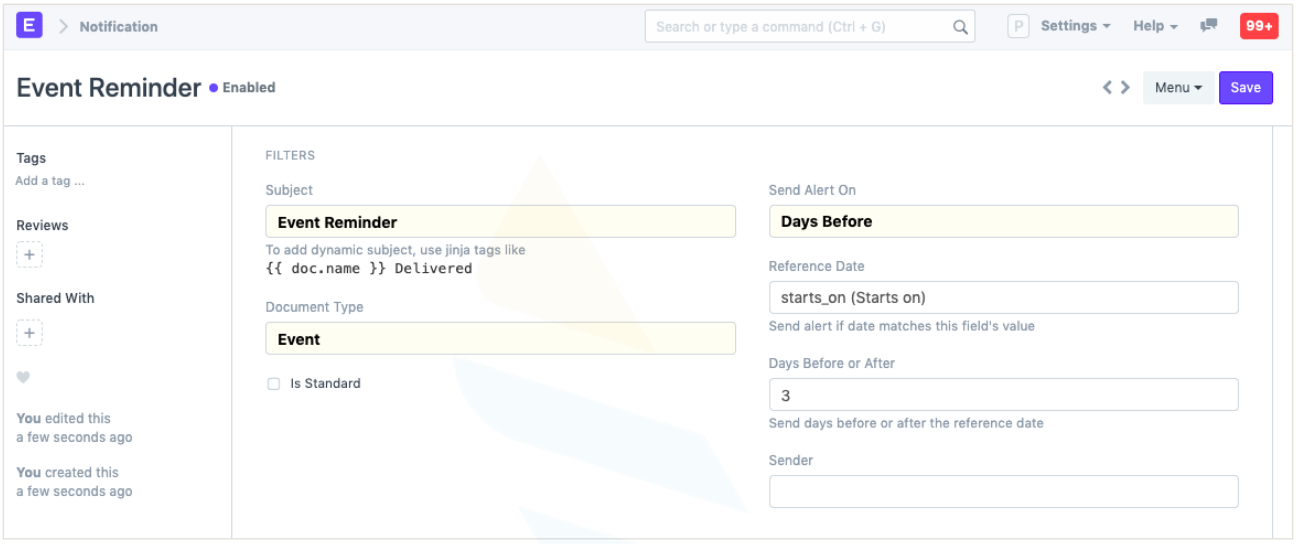
If Workflows control who can do what, Notifications make sure the right person knows at the right moment — and they're pure configuration. A Notification is a rule with a trigger, an optional condition, a recipient and a message. The trigger — "Send Alert On" — can be a document event (a record is created, saved, submitted or cancelled), a field value changing, or a date-based event: a set number of days before or after a reference date on the document. That last one is how you get "remind the account manager three days before an AMC expires" without anyone watching a calendar.

The condition narrows it further using a simple field comparison — only alert when the status is a certain value, for instance — and you can combine conditions with basic and/or logic. Recipients can be a person referenced on the document itself (the assigned salesperson, the account owner) or a fixed list of addresses. The message is written once as a template, and you can drop in live values from the document so each alert is specific rather than generic.

What makes this genuinely no-code is that none of it requires programming — you're choosing options and typing a message. Alerts aren't limited to email either: ERPNext can raise an in-app System Notification, and with the right setup route messages to Slack, SMS or WhatsApp. For the vast majority

of "we keep forgetting to…" problems, a Notification is the whole fix.

- Trigger on real events — new, save, submit, cancel, a field changing, or a set number of days before/after a date on the document.
- Condition — fire only when it matters (e.g. a particular status), combined with simple and/or logic.
- Recipients — a person named on the document or a fixed list; the message is a reusable template with live document values dropped in.
- Multiple channels — email and in-app System Notifications out of the box, plus Slack, SMS and WhatsApp with the relevant setup.



A Notification configured to alert a set number of days before a date on the document — a trigger, a condition, recipients and a templated message, all chosen in the form.

Assignment Rules & Auto Repeat — no code

Two more built-in tools close the gaps around ownership and repetition, and both are form-driven. An Assignment Rule automatically hands new documents to the right person instead of leaving them in a shared pile. You choose a distribution method — Round Robin passes each new document to the next user in sequence; Load Balancing gives it to whoever currently has the fewest assignments; or you can assign based on a user field on the document itself. Add an optional condition (only auto-assign open, high-priority tickets, say), a list of users, and you have a fair, automatic queue with no manual triage. You can even have it set a due date based on a date field, and use priority to layer several rules on one document type.

Auto Repeat solves the opposite problem — documents you create on a rhythm. Instead of re-keying the same monthly journal entry, recurring subscription invoice or standing order, you set one document to repeat at a chosen frequency: daily, weekly, monthly, quarterly, half-yearly or yearly. ERPNext then generates the next copy on schedule, can submit it automatically if the document type allows, and can email it to recipients with your chosen print format. Any document type that has Auto Repeat enabled in its customisation can use it.

Both are honest no-code tools. Assignment Rule conditions, like Workflow and Notification conditions, accept a short expression if you want fine-grained routing — but the everyday cases (round-robin a queue, repeat an invoice monthly) are set up entirely by choosing options.

- Assignment Rule — auto-distribute new documents by Round Robin, Load Balancing, or a user field, with an optional condition and priority.
- Fair queues with owners — support tickets, leads and tasks get an owner automatically instead of sitting unclaimed.
- Auto Repeat — turn one document into a recurring series (daily through yearly) so nobody re-keys the same invoice or journal entry.
- Hands-off generation — repeated documents can auto-submit and be emailed with a print format on schedule.

Where light scripting & scheduled jobs come in

The four tools above cover a genuinely large share of everyday automation, but they have edges. When your logic is more than "on this event, to this person, when this field equals that," you cross into light scripting — and it's important to be clear that this is real code, best written by a developer or partner even though ERPNext lets you enter it from the browser.

The main tool is the Server Script: Python logic that runs on the server, entered through the UI. Server Scripts come in types — a DocType Event script runs custom logic before or after a document is saved or submitted (validating combinations of fields, auto-populating values, blocking an action); an API script exposes a custom endpoint; a Scheduler Event script runs on a schedule you define, including cron-style timing, for background jobs like nightly recalculations or a daily digest that the simple tools can't express. There's also the Client Script — JavaScript that changes how a form behaves in the browser (showing or hiding fields, live calculations, prompts) — which again is code, just code that runs in the user's screen rather than on the server.

Scheduled jobs deserve a specific mention because they're where recurring automation gets powerful. Beyond Auto Repeat's fixed document-copying, a scheduled Server Script (or a scheduled task built into a custom app) can run any logic on an interval — reconcile something overnight, chase a set of records, push data to another system. That flexibility is exactly why it's code: it can do almost anything, which is also why it should be written, reviewed and tested properly rather than improvised in production.

- Server Script (Python) — custom logic on document events, a custom API, or a scheduled background job; entered in the UI but genuinely code.
- Client Script (JavaScript) — changes form behaviour in the browser: conditional fields, live calculations, prompts.
- Scheduled jobs — run arbitrary logic on an interval or cron schedule, well beyond Auto Repeat's document copying.
- The honest line — if the rule needs real branching, calculation or integration, it's code; treat it like code and have someone qualified build it.

Governance — automate deliberately, not accidentally

The flip side of "anyone can automate" is that automation set up carelessly becomes its own mess. A dozen overlapping Notifications turn into inbox noise everyone learns to ignore. A Workflow that mirrors an org chart nobody follows anymore blocks work instead of speeding it. An Assignment Rule quietly routing tickets to someone who left creates a silent backlog. Because these tools are so easy to switch

on, the discipline has to come from process, not from the software resisting you.

A little governance keeps automation an asset. Decide who is allowed to create and change Workflows, Notifications and Assignment Rules, and keep that a small, accountable group rather than everyone with admin access. Write down what each automation is for and who owns it, so a rule that fires at the wrong time has a name attached. Review them periodically — especially after any change to roles, processes or the org — because automation encodes assumptions, and assumptions expire. And test changes somewhere safe before they touch live documents: a Workflow or Server Script that misbehaves in production can block real transactions.

This matters most at the boundary with code. No-code changes are relatively easy to reason about and reverse; Server Scripts and scheduled jobs can have effects that are harder to see and undo. The heavier the automation, the more it deserves review, documentation and a testing step — the ordinary hygiene of any change that runs unattended.

- Limit who can build automation — a small, accountable group for Workflows, Notifications and Assignment Rules, not everyone with admin rights.
- Document purpose and owner — every automation should have a reason and a name attached to it.
- Review after change — roles, processes and org charts move; the rules that encode them need re-checking.
- Test before live — validate Workflow and script changes off real documents; code deserves more review than configuration.

Getting help — and knowing when you don't need it

The practical takeaway is a split. A large amount of the automation that would make your ERPNext calmer to run — approvals that route themselves, reminders that fire on time, work that assigns itself, documents that recur without re-keying — is genuinely within reach of your own team, configured through ordinary forms. If you've been quoting a developer for these, you may be paying for a project where a settings change would do. Knowing that is worth real money and time.

Where a partner earns their fee is at the edges and the seams: designing an approval matrix that matches how decisions are actually made, writing and testing the Server Scripts and scheduled jobs that no-code can't express, integrating ERPNext with other systems, and — often most valuably — bringing governance so your automation stays coherent as you grow rather than sprawling into noise. The skill isn't only writing code; it's knowing which problems don't need any.

As an official ERPNext partner working with Indian businesses, that's the line we try to draw for clients honestly: we'll set up the no-code automation with your team so they can own and extend it, and we'll build and govern the scripted, scheduled and integrated pieces properly where they're genuinely needed. The aim is an ERPNext that runs more of itself — without a change request for every rule.

KEY TAKEAWAYS

1. Most everyday ERPNext automation is genuinely no-code: Workflows, Notifications, Assignment Rules and Auto Repeat are all set up through ordinary forms by your own admins.
2. Workflows turn inbox approvals into enforced, role-based, auditable chains; Notifications fire alerts on events or dates so nothing depends on someone remembering.
3. Assignment Rules give unclaimed work an owner automatically (round-robin or by load); Auto Repeat regenerates recurring documents so nobody re-keys the same invoice or entry.
4. Light scripting — Server Scripts, Client Scripts and scheduled jobs — is real code for logic the no-code tools can't express; treat it as code, and have someone qualified build and test it.
5. Automate deliberately: limit who can create rules, document each one's purpose and owner, review after any org or process change, and test before it touches live documents.

FAQ

Can I really automate ERPNext without a developer?
For a large share of everyday automation, yes. ERPNext runs on the Frappe framework, which ships Workflows (role-based approvals), Notifications (email and in-app alerts), Assignment Rules (auto-distributing work) and Auto Repeat (recurring documents) — all configured through ordinary forms in the browser, with no code and no deployment. Your own admins can set these up. A developer becomes necessary only when your logic goes beyond what these tools express.

What is the difference between a Workflow and a Notification in ERPNext?
A Workflow controls what can happen to a document — the states it moves through (Draft, Pending, Approved) and which role is allowed to make each move — so it enforces an approval process. A Notification controls who gets told — it sends an email or in-app alert when a document event happens, a field changes, or a date approaches. They're complementary: a Workflow routes the approval, and Notifications keep the right people informed as it moves.

When does ERPNext automation actually need code?
When the rule is more than "on this event, to this person, when this field equals that." Custom validation across several fields, calculations, exposing an API, integrating with another system, or a background job that runs arbitrary logic on a schedule all need light scripting — Server Scripts (Python) or Client Scripts (JavaScript). Frappe lets you enter these in the UI, but they are genuinely code and should be written, reviewed and tested by a developer or partner.

Can ERPNext send automatic reminders and recurring invoices?
Yes, both without code. A Notification can send a reminder a set number of days before or after a date on a document — for example, before a contract or AMC expires — to a person named on the record. Auto Repeat regenerates a document on a chosen frequency (daily through yearly), so a recurring invoice or monthly journal entry is created automatically, optionally submitted and emailed, instead of

being keyed in by hand each cycle.

Talk to a real ERPNext expert.

Call or WhatsApp +91 62358 66111 · info@acube.co · acubeinnovations.com