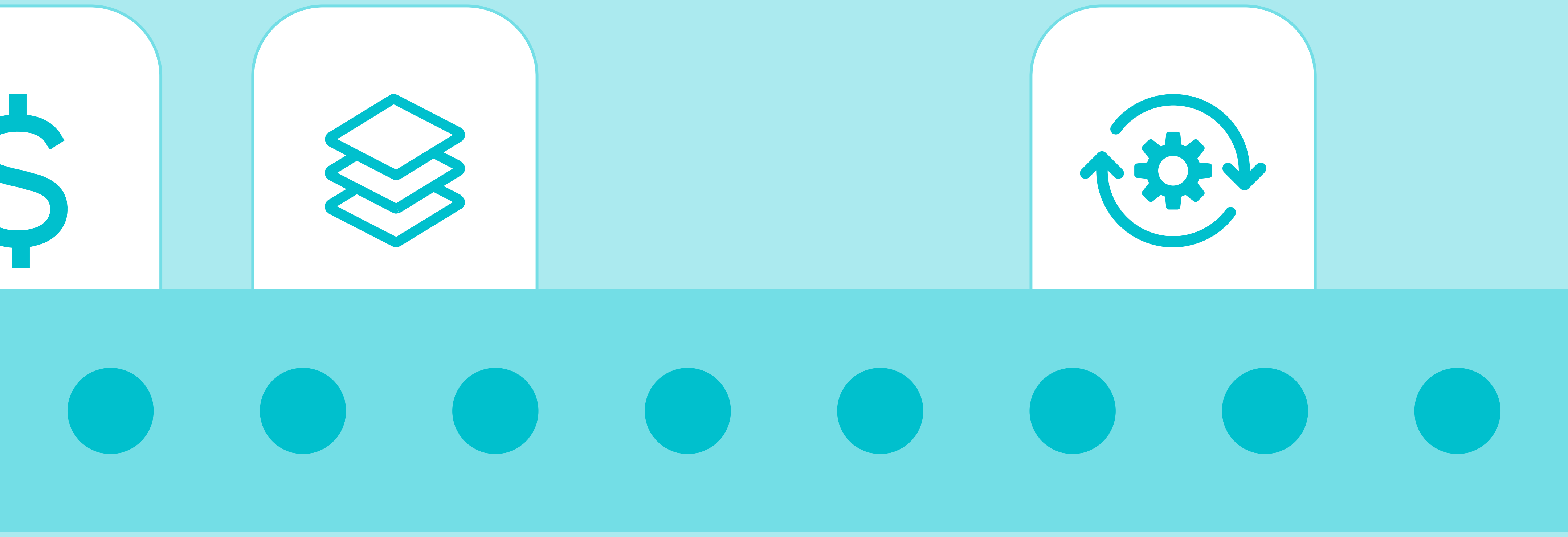
The Great Debate Buy vs Build Rich Text Editors

Agile businesses combine buying, subscribing, or renting, with building, to iteratively reshape and assemble their software stack. That scales and produces results. Fast.



FOREWORD

Transformation Is Driving Big Change

Business has changed. Historically, companies relied on people to get things done. Changes and growth gradually happened – as roles, hierarchies and skills evolved – but their progress was never a mission-critical impediment to business success.

Now, companies are equally composed of people and *digital* assets. Those assets are closely woven into the fabric of every company, and people are as reliant on them as themselves, to drive business growth.

Adaptability, agility and scalability are the valued skills – of both people and technology.

According to a 2020 McKinsey report, "With technology powering everything from how a business runs to the products and services it sells, companies in industries ranging from retail to manufacturing to banking are having to develop a range of new skill sets and capabilities. In addition to mastering the nuances of their industry, they need to excel first and foremost at developing software."

GARTNER REPORT

Gartner projects worldwide IT spending to total **$4.2 trillion in 2021** – an increase of 8.6% from 2020.

Or do they? For most of those businesses, investments in developers and software, haven't led to meaningful performance improvements. Instead, it's become an uncomfortable 'black box' where money pours in, and results are slow to trickle out.

Leading digital organizations have found a new way. They're shaping a new set of principles for scoping, procuring, renewing and operating their digital assets.

By assembling a plastic, not static, tech stack.

1

The Great Debate: Buy vs Build, or Something Else?

↑
Back to the table of contents

It's a familiar question. In almost every development project, no matter the size, the earliest question that arises is: **"Do we buy, or do we build?"**

Digital transformation has forced a shift in that thinking.

The two-part question now arising is: "What out-of-the-box software components can we selectively buy, integrate, and assemble to create a flexible software stack? and "Where does our development expertise and true market need lie, to build the rest?"

Those questions flow from agile thinking – how to speed a company's software development output, so they can focus on their core business capabilities and market opportunities.

What's changed? Transformation and speed-to-market have made the old belief, in wholly custom-building technology, redundant. Neither budget, nor developer resources, are elastic and it's impossible to match the 'bleeding edge' changes across every digital realm.

MULESOFT REPORT

The average enterprise spends approximately **$3.5M per year** on integration-related IT labor according to the Mulesoft 2021 Connectivity Benchmark Report, yet just 37% of organizations say IT completed all their projects, with fewer than 4 in 10 teams fulfilling their project commitments to business stakeholders.

So, what's the answer?

Forrester advises companies to move beyond all-in-one solutions-thinking. Leading digital organizations agree – they're no longer buying monolithic suites or building custom software beyond their specializations.

Agile businesses take a plastic approach – combining buying, subscribing, or renting, with building – and iteratively reshape their software stack, to fit the company's changing goals.

That means buying smaller specialised components – like a rich text editor – with APIs as building blocks, within the company’s larger IT/dev software toolkit. The APIs are stitched and restitched together, while other parts are developed inhouse.

Assembling reusable software tools, scales. And produces results.

McKinsey has dubbed this tech assembly approach a ‘digital factory’, where a company “brings together the skills, processes, and inputs required to produce high-quality outputs. [...] The best digital factories can put a new product or customer experience into production in as little as ten weeks. The innovation can then be introduced and scaled up across the business in eight to 12 months.”

Why does it produce results? Assembling maximizes the horsepower behind third-party specialization, empowers developer expertise to focus on what they do best and delivers the nimbleness to move beyond productivity obstacles.

Ultimately, buy vs build isn’t the conundrum.

It’s how do you agilely build everything you need… at scale?

Complexities of Rich Text Editors

↑ Back to the table of contents

Digital transformation is driving the need for a new set of tools.

According to Mulesoft, "the average business transaction now crosses 35 disparate systems. Digital transformation comes not from the implementation of any single technology, but from an architecture that is built for constant innovation, enabling companies to bring multiple technologies together again and again to create a compelling and consistent customer experience – quickly."

But being on the 'bleeding edge' of every digital realm isn't possible.

Rich Text Editors (RTEs) are one of the highest-risk areas for both cost and development time overruns. Feature and system complexities are notoriously underestimated, iterative bugs demand constant attention, and remote working is forcing rapid evolutions and adaptations.

A developer who's inexperienced in developing RTEs generally underestimates the myriad of edge cases, across both browsers and functionality. Even things that are perceived to be basic, are hard.

For example, in an average simple RTE, the 'Enter' key performs over 100 different actions depending on where you are and what you're doing. That doesn't take into consideration browser compatibility or other dependencies based on what you have built into your product.

- **Is your development team expert enough in rich text editing, to build that?**
- **Is it a skill you want them to maintain moving forward?**

Building a rich text editor is way beyond what most developers initially consider, which is why most rich text editors have large teams of developers working on them year-round. Or, if you utilize an open source project that has some components that your team can use as a framework, there are still numerous considerations (and decisions) you have to make.

When coding a RTE from scratch, these are the components you need to build, test and maintain:

STRIPE FORECAST

The latest forecast by Gartner shows that the worldwide low-code development technologies market is projected to total **$13.8 billion in 2021**, an increase of 22.6% from 2020. The surge in remote development during the COVID-19 pandemic continues to boost low-code adoption, despite ongoing cost optimization efforts.

Common Rich Text Editor Requirements

Core Components

These are your no-frills core components – the absolute basic minimum expectations to get a rich text editor running

Contenteditable

A contenteditable is a special HTML attribute that can turn any HTML div container into a basic text editor. There are no toolbars or menus. If you need to add any more functionality, Contenteditable becomes the base on which to build your features.

Basic Formatting

Bold, italic and underline are the most basic types of formatting that can be leveraged from ContentEditable, which provides these basic features through the execComand API.

`For example:` **`"Hello <u>world</u>"`**

`The rich text output will be` **`"Hello world"`**

This does not include Advanced Formatting, where undesirable formats are stripped out, and rules can be added that conform with brand style guides.

HTML Compliance

HTML is a structure that formats the text in the desired manner, otherwise the content is a wall of words without structures like paragraphs, formating or sections.

If the HTML is incorrect, it confuses the browser and renders incorrectly on the screen. Developers and Product Managers have to decide, scope and develop the rules for each use case.

Colors

The ability to change text color or create features so it remembers what color was last selected, has to be custom coded within the editor itself.

Undo

When should we set an undo point? When you undo – does it do every character or every word? How does it deal with emoji shortcuts (if you've developed emjoi's to work)? Is an undo point set on enter?

There’s no right answer to any of these questions and the reality is the undo algorithm should take into account all of these approaches to provide the best experience for the end user. When developing new features, you should always consider how undo will work, and what the user would expect when they undo.

Focus

One of the challenges working with ContentEditable is the browser focus. When you’re typing to create content, the browser's focus is in the content area. However, when your cursor is placed inside another editing field, that field assumes focus.

When a web page has three editable text areas, this is where the browser’s focus and text selection work together to find out which text area and which word needs formatting applied. Once the UI is added to an editor, it can confuse focus – because when a toolbar button is clicked (to make a word bold), the focus shifts to the toolbar button.

Therefore, focus needs to be normalised across all browsers, because you want the selected text inside the content to go bold when the toolbar bold button is clicked. On some browsers, you need to refocus the content then call the browser API to do bold, then refocus the toolbar button so users can keyboard navigate. Developers of rich text editors need to manage all aspects of what happens, to achieve a consistent editing experience, for every single feature created.

Selection

There are two kinds of selections to deal with, that tell the editor which text to apply a transformation, e.g. ‘make this selected word bold’:

- **Collapsed,** this is often denoted as a blinking cursor when you click on an editable text field.
- **Ranged selection,** is the blue highlight that can span across words, paragraphs, images, tables or the entire document.

Ranged selections are more complicated to normalise, because they select a start and end point and as you add more features to your editor, you need to consider managing selection. For example, when you add a toolbar that adds an interface to your editor, this introduces focus issues (see Focus section) which affects selection.

When you click on a toolbar button, say italics, the browser focus is shifted to the italics button. In effect this revokes focus from the content, therefore losing its selection, so when you try to apply italic you’re applying it to nothing.

Along with normalising selection, you also need to remember where selection was before it was lost, so you can apply italics to the previous selection in the content.

Loading Content

How do you actually get content into the editor itself? If someone's writing fresh content it’s fairly simple, but what happens when you want to add in images, what about styling, CSS and fonts? What happens when they begin trying to paste content over from MS Word or Google Docs – the underlying HTML isn’t clean, so it’s going to create formatting issues.

You need to scope and determine every single use case for how people will get content into a document and create custom solutions for each browser type to support and manage this task.

Input Filtering

When content is loaded into the editor, is it guaranteed to be HTML compliant? Is the content older, or does it meet the modern HTML standards? Creating input filtering allows your editor to inspect the validity of the content being loaded, and the more advanced you make the filtering the cleaner the content inserted into the editor becomes.

A more reliable way to do content input validation and filtration, is to use a schema. Compliant code means less confusion to the browser trying to work out what you want and it results in fewer bugs.

Exporting Content (output)

Consider how your content will display on the screen when published. At its heart, this is the core of WYSIWYG editing experiences, so how do you ensure it translates what your author has built into what he (and the reader) actually sees on screen?

How is the content hosted, and how will the web page recreate the contents that was created in the editor? In line with modern web development principles, content needs to be separated out into HTML file, CSS file, images stored separately and the published web page will pull all of these elements together to re-create the content. For all of this to come together, the editor needs to know where it needs to encode image links so images can be loaded, where the CSS files are, how to load custom fonts into the editor, and how to load those fonts on the published website.

Output Filtering

Output filtering exists to ensure published content looks and feels the same. During the editing session, the editor needs to make temporary markers in the content, to provide visual editing queues – like where spaces, line breaks or anchor links are located. Image drag handle resizers are also required for editing, but not when you’re publishing.

Browser Differences

Users of text editors expect consistency, although this can be hard to achieve when there are many browsers to support and each browser may yield a slightly different result.

Providing a consistent editing (writing) and reading experience (published) is where the complexities lay when developing an editor. When architecting an editor, an engineer needs to understand all of the minor differences between all supported browsers and have all their inputs and outputs produce the same or equivalent result for each browser.

Successful normalisation of browser behaviours occurs gradually over time, as developers find new inconsistencies to 'normalise'. This is where test automation can capture changes in browser behavior as soon as a new browser version is released.

The above requirements cover content and formatting. Next, your developers need to look deeper into what content is made of and how to render that within a rich text editing environment.

Handling Content Types

These are the additional features that individually need to be created inline with the scope above (they're not often in a core RTE experience) but are frequently requested by end users

- Links
- Embeds
- Lists
- Tables
- Images
- Uploading
- Advanced Formatting
- Emoji

UI Interactions

Surfacing up UI components, how are people going to interact and work with your editor

- UI General
- UI Toolbar
- UI Buttons
- UI Menu
- UI Context Menu
- UI Dialogs
- Accessibility
- Touch Devices (mobile/tablets)
- Advanced Keyboard Interactions

Look and Feel

How the editor looks in your application and how you configure it to meet your product requirements

- Content Editing
- Content Published
- Skins and Customizations
- Configuration

Enterprise Grade Features

The **advanced features** (over and above a core editing experience), that growing SaaS and established companies demand in order to compete

- Automated Testing
- Human Testing
- Localization
- Security
- Spell Checking
- Integration
- Performance

A rich text editor is a complicated Swiss watch – with many unseen moving parts that work in synchrony. On the surface, an RTE should always be visually pleasing and comfortable to work with, and users demand that they should never need to learn how to use an editor – instead preferring to trust it with their fingers.

A great editor experience must help users achieve an immersive state where they are fully focused on the content and not fighting against the editor to do a simple task.

However, to achieve that goal, it’s never a once-and-done project.

A trusted editing environment requires year-round ongoing maintenance – to keep up with browsers, technologies, changing technologies and how the content is displayed to its ultimate audience, your readers.

Developer Productivity Cost

↑ Back to the table of contents

Cost drives much of today's technology.

In specialized fields like RTEs, an inexperienced in-house development approach often adds uncertainty to the scoping equation. Then costs increase when teams get side-tracked on building deep user features, or the requirements transform due to a market change and user preference. Scope creep continues.

In the end, time-to-market suffers. And time is money.

$75K (projected initial monthly revenue) x 6 (month delay)
= $450K (lost potential initial ARR)

Example cost of delayed time-to-market per project or initiative

---

Teams eventually do deliver, but only after blow-outs in both costs and hours, and sometimes only with the help of additional costly contract resources. Then, as projects and initiatives increase – each with its own custom-build development focus – there's replication of effort and wastage of scarce talent resources.

According to a recent report by payment platform Stripe, "As technology fracks into every aspect of the world economy, software engineers are becoming one of the world's most precious resources. While businesses today face myriad issues – security vulnerabilities, trade tariffs, complex government regulations, increased global competition – how they deploy their developers may be the most overlooked factor impacting their future success."

The survey questioned thousands of enterprise leaders and they learned:

81%

of C-level executives think software needs to become more of a core competency for their company in the next 10 years

40%

of developers say they're hindered by custom technology

81%

say they're held back by legacy systems or technical debt

The report goes on to name developers as the ‘force-multipliers’, who when deployed effectively, “have the collective potential to raise global GDP by $3 trillion over the next ten years.”

While a 2020 McKinsey report on developer velocity noted that “Leading companies also use tools to unleash Developer Velocity by investing in low-code and no-code platforms. These platforms enable the average business user to develop applications without any software experience, freeing up seasoned developers to focus on the most challenging tasks.”

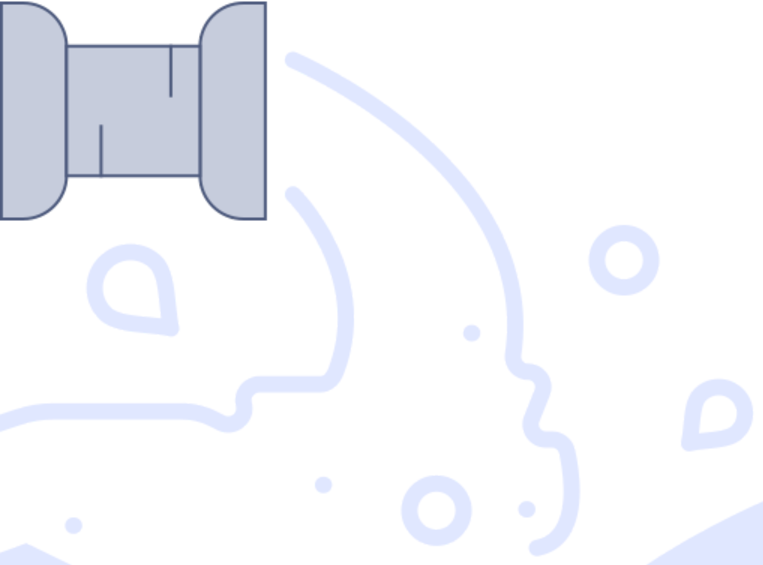
In short, businesses must get better at leveraging their existing software engineering talent, if they want to move faster, build new products, and tap into new and emerging trends.

Others believe that the true problem is not enough devs on their team. It’s not. It’s how they’re being leveraged. Failing to deploy the best talent, on the right project, is repeatedly becoming the biggest threat to a company’s digital transformation and market success.

It’s an even bigger threat than access to capital.

STRIPE SURVEY

The Stripe survey identified that “the average developer spends more than **17 hours a week dealing with maintenance issues**, such as debugging and refactoring. In addition, they spend approximately **four hours a week on “bad code”** which equates to nearly **$85 billion worldwide in opportunity cost lost annually**, according to Stripe’s calculations on average developer salary by country.”



↑ Back to the table of contents

4 Total Cost of Ownership

Final decisions likely come back to cost.

But to be a valid decision, all costs need to be included. When evaluating a buy vs build argument, it's critical to thoroughly understand total costs incurred during the software lifecycle – typically seven or eight years.

In an Infoworld article, Mark Lutchen, former global CIO of PricewaterhouseCoopers, cites this step as being vitally important, “because 70% of software costs occur after implementation. A rigorous lifecycle analysis that realistically estimates ongoing maintenance by in-house developers often tips the balance in favor of buying.”

So, when the majority of costs happen *after* you've built and implemented, it can be difficult to quantify the total cost of ownership (TCO).

It gets worse.

When the software requires deep domain knowledge and is as specialized and complex as building, maintaining and extending an enterprise grade rich text editor, the challenge to definitively know the time, effort and costs that lay ahead, is an even greater challenge.

However, some indicators are readily available.

Core Rich Text Editor – Build Cost Estimate

To build **just the basic open source components** of three leading rich text editors (excluding advanced features and plugins) Open Hub estimates it as:

OPEN SOURCE CODE	CODEBASE SIZE	ESTIMATED EFFORT	AVERAGE SALARY P/YR*	ESTIMATED BUILD COST*
TinyMCE	270,122 lines	115 person-years(36.6 months using 46 developers)	US$128,749** p/yrSenior Software Engineer	**US$14,890,054***
CKEditor	484,093 lines	220 person-years(44.9 months using 71 developers)	US$128,749** p/yrSenior Software Engineer	**US$28,283,035***
Summernote	43,994 lines	16 person-years(19.4 months using 12 developers	US$128,749** p/yrSenior Software Engineer	**US$2,023,716***

Note: Lines of code are indicative only. For RTEs, the quality of code also isn't directly reflective of either hours spent, or codebase size.
* Using the Basic COCOMO Model (Accessed 1 July 2022)
** Average base salary for a Senior Software Engineer is US$128,749 per year in Silicon Valley, CA
(Accessed 1 July 2022)
A person-month is equivalent to approximately 160 hours of labor, and is the amount of work performed by a single average worker in one month (ie. 12 person-month project will take 4 developers 3 months work to finish). A person-year is the total effort in person-months divided by twelve, to estimate the project length in years.

It should be noted the **above costs exclude the additional costs** of Senior Product Manager*** support (prior to the discovery/inception phases and also throughout the project) as well as Senior Product Designer**** support, during the development of a rich text editor.

That said, it doesn't just start-and-end there. For an enterprise grade RTE, more advanced features are also required.

An Advanced RTE Feature – Build Cost Estimate (excl. core editor)

One such advanced feature is a plugin that helps users cleanly transfer content from its source, to the rich text editor. Ideally, the plugin should automatically parse the content for security vulnerabilities, remove unnecessary style elements as well as generally clean up and modernise the HTML. That's no short order.

Using a normalised COCOMO Model, the estimated engineering requirements for building that single feature, using:

A Senior Software Engineer

DEVELOPMENT COST

$1,814,399

14.1 years, using one developer

- A Senior Software Engineer
- Average salary rate (**US$128,749**** p/yr excluding oncosts, RSUs and bonuses)
- **39836 lines of code**
  The total LOC includes 23085 LOC for the plugin itself, as well as 16751 LOC for the dependent libraries that are maintained ongoing, as part of the feature
- 169.1 person-months = **14.1 person-years, using one developer**
- Excluding ongoing maintenance and extensibility work

Equals = $1,814,399 in development cost

OR...

DEVELOPMENT COST

$1,814,399

18.8 months, using eleven developers

For more efficient production scheduling and quality outputs, as shown below:

- 169.1 person-months = **18.8 months, using eleven developers**
- Excluding ongoing maintenance and extensibility work

Equals = $1,814,399 in development cost

Advanced RTE Feature COCOMO Modeling

Software Development (Elaboration and Construction)

EFFORT	SCHEDULE	COST
169.1 person-months	**18.8 months**	**$1,814,399**

TOTAL EQUIVALENT SIZE	EFFORT ADJUSTMENT FACTOR (EAF)
39836 SLOC	**1.00**

Acquisition Phase Distribution

Phase	Effort	Schedule	Average Staff	Cost
Inception	10.1	2.3	4.3	$108864
Elaboration	40.6	7.0	5.8	$435456
Construction	128.5	11.7	11.0	$1378944
Transition	20.3	2.3	8.7	$217728

It should be noted that the **above $1.8M estimate for a single advanced feature, also excludes the additional support costs** required for the development of a product ready feature:

A full time Senior Product Manager*** for:	• 6-10 months prior to the Inception/Discovery Phase • 107.0 person-months throughout the project
A full time Senior Product Designer**** for:	• 1 week during the project

Given the nature of this particular advanced feature, it's definitely not a build-and-forget project.

An engineering team would be required to constantly monitor changes to both MS Word and Google Docs, and play catch up with those developments. Demands also change and evolve over time, after the initial feature has been developed and launched.

Therefore, extensive engineering time and resources would be required to both maintain and evolve the feature to keep pace with market changes.

Ongoing Maintenance – 1 x RTE Feature Cost Estimate (excl. core editor)

MAINTENANCE COST

$21,458

yearly for a single feature, ongoing

- Expect to dedicate a minimum 2 person-months full time per year, every year of the RTEs life, of Senior Software Engineer** resources, to keep the feature afloat.
- This work would include browser changes and various edge cases that emerge.

EQUALS = $21,458 yearly in development cost for a single feature, ongoing**

PLUS

- A full time Senior Product Manager*** throughout any maintenance work.
- A full time Senior Product Designer**** sporadically during maintenance work.

Long-term Extensibility – 1 x RTE Feature Cost Estimate (excl. core editor)

EXTENSIBILITY COST

$64,374

for a single feature, biannually ongoing

- Every other year at a minimum, expect to dedicate at least 6 person-months of full time Senior Software Engineer** resources, to account for larger changes in MS Word and Google Docs.
- This is required due to Microsoft, Google and Apple regularly changing their apps, which frequently breaks the paste feature, and the dev team must be able to react quickly to keep pace.

EQUALS = $64,374 in development cost for a single feature, biannually ongoing**

PLUS

- A full time Senior Product Manager*** throughout any extensibility work.
- A full time Senior Product Designer**** sporadically during extensibility work.

Note: All estimates exclude on-costs, RSUs and bonuses
* Using the Basic COCOMO Model (Accessed 1 July 2022)
** Average base salary for a Senior Software Engineer is US$128,749 per year in Silicon Valley, CA
*** Estimated base salary for a Senior Product Manager is US$157,340 per year in Silicon Valley, CA
**** Estimated base salary for a Senior Product Designer is US$145,700 per year in Silicon Valley, CA
(Accessed 1 July 2022)
A person-month is equivalent to approximately 160 hours of labor, and is the amount of work performed by a single average worker in one month (ie. 12 person-month project will take 4 developers 3 months work to finish). A person-year is the total effort in person-months divided by twelve, to estimate the project length in years.

As this shows, the typical cost of building even just a single feature of an RTE, isn't small.

An in-house build approach may deliver customisations. But for RTEs, it definitely adds greater uncertainty into a TCO equation that's already fraught with technical complexities and unknowns – RTEs are in a state of constant evolution.

Assembling reusable software tools that scale, is the solution.

THE HARVARD BR

The Harvard Business Review examined 1,471 projects and reported that one in six IT projects had an average cost overrun of 200% and a schedule overrun of almost 70%.

Complex Questions are Hard

↑
Back to the table of contents

Companies are under pressure. There simply isn't enough time to build everything their users need. However, a buy vs build decision (or any other approach) is never orderly, neat, or simple.

Hard choices are never easy. The companies focused on rapid transformation, have already made the move. They've gone from slow, project-focused iteration to swift innovation and continuous deployment. And they're growing. Fast.

They're leveraging the velocity of dev team talent focused on company differentiation, not projects outside their specialization. They've taken a plastic 'digital factory' approach and assembled an agile software stack, instead of custom-building. According to McKinsey, "the core principle of these approaches is to both maximize the value of what already exists and reduce the burdens of development and maintenance."

The same report notes "This tech fusion enables companies to radically simplify and accelerate the development process for both launching and scaling new businesses."

It's producing scalability, savings, and speed-to-market.

80%

of technology products and services will be built by professionals outside technology by 2024

For rich text editors, the buy vs build debate encapsulates

Burdens of RTE Custom-Build

- Complexity of build + maintenance + cost
- Slower time-to-market
- Stretches dev resources
- Lacks scalability + agility
- In-house maintenance of risk

Benefits of RTE Buy + Assemble

- Simplification of installation + maintenance
- Accelerates time to market
- Focuses dev resources
- Delivers scalability and agility
- Specialist maintenance of risks

For RTEs, a buy-and-assemble component approach minimises custom-build outside a company's expertise, facilitates continuous deployment of software from a reusable stack and maximises dev talent, who pick the exact tools they need for every project.

Then when a market opportunity opens, the decision is easy. It’s not whether to build the feature or product. It’s not whether to buy. Instead, the decision focuses on what aligns with the company’s market strategies.

By buying proven components, it scales a new way of working.

That incubates a democratized digital culture.

Growth in digital data, low-code development tools and artificial intelligence (AI)-assisted development are among the many factors that enable the democratization of technology development beyond IT professionals.

- Rajesh Kandaswamy
Gartner’s Research Vice President

↑
Back to the table of contents

Remote Work is Driving Cloud

There's never a single decision. It's always many.

Technically not part of the buy/build choice, the decision *afterwards is* equally contentious – Cloud (shared or owned) vs Self-Hosted (virtual or physical). It relates to how you'll consume and use a rich text editor.

Digital transformation is forcing hard decisions based on having an agile, scalable tech stack that's accessible from anywhere. Remote work is here to stay – so every organization is reassessing its IT infrastructure, to satisfy work-anywhere, sell-anywhere and learn-anywhere capabilities.

In a recent Gartner report their Senior Research Director, Ranjit Atwal, said "Through 2024, organizations will be forced to bring forward digital business transformation plans by at least five years. Those plans will have to adapt to a post-COVID-19 world that involves permanently higher adoption of remote work and digital touchpoints."

Given such momentous changes, it's fair to say Cloud is the future. If your tech stack doesn't keep up, you risk becoming redundant.

While G Suite (Google Docs, Sheet, Slides) and MS Office 365 are killing the offline Office Suites, equally Google Drive, One Drive, and Photos are affecting the sales of external hard drives. Traditional software and hardware setups are dying. Fast.

GARTNER REPORT

Gartner estimates remote working knowledge workers to be 51% of the worldwide workforce by the end of 2021 – up from 27% in 2019, while remote workers will represent 32% of all employees worldwide – up from 17% in 2019.

According to the ResearchGate "Self-Hosting vs Cloud Hosting" report, "The resource-elasticity offered by cloud providers eliminates the up-front costs of building a self-hosted infrastructure and removes delays by allowing tenants to scale up their resources on demand."

For rich text editors, Cloud offers a level of scalability that Self-Hosting can't.

It's easier to set up and requires no actual installation of code – APIs do everything. Updates automatically deploy, it's accessible anywhere, extra resources can be added as needed, in seconds, and scaling up is fast – as and when growth happens.

Fundamentally, Self-Hosting is the opposite. However, there's 100% control of when and what's deployed, developer resources do the work of deploying code to servers and updating it when they're released. The balls are all in your court, all the time.

The decision splits between agile growth, or control.

Which are you choosing?

TINY TECHNOLOGIES

Tiny is the creator of **TinyMCE, the world's most trusted WYSIWYG component** that enables rich text editing capabilities within an application. Scalable, adaptable and reusable, it powers 100M+ projects worldwide and more than 1.5M+ developers use it to add velocity to their tech stacks, so they can build and ship their projects faster.

There's tens of thousands of market-leading applications powered by Tiny globally. It's helped SaaS companies, large enterprises, content creators and publishers to launch, grow and scale their businesses, reduce their development and technical debt burdens, minimize ongoing support tickets and boost the productivity of their users.

COST ESTIMATE CURRENCY

All cost estimates quoted are in US$

ANNUAL COST ESTIMATE UPDATES

This post is accurate at the time of publishing. The TinyMCE Buy vs Build Whitepaper, COCOMO cost estimate articles (Accessibility Checker, Spell Checker Pro, PowerPaste) and Buy vs Build blog posts (Costs, Developer Velocity, APIs) are all updated annually, in July.