Identifying Qualifying R&D in Software

30/07/2025 7:56 am IST

In the fast-moving world of software development, it can be deceptively difficult to identify whether a specific activity counts as qualifying R&D. Just because a team is building something novel, using the latest frameworks, or releasing a product to market doesn't mean they are entitled to claim research and development tax relief or credits.

Different jurisdictions offer generous incentives for genuine innovation, but only where the underlying work meets strict criteria.

Across Ireland, the UK, Canada, and Australia, a consistent pattern emerges. The determining factor is **not** whether something is new to your business or even new to the market, but whether the work seeks to **overcome scientific/technological uncertainty** through a **structured, scientific or engineering-based investigation** aimed at achieving **scientific/technological advancement**.

As the Irish Revenue Commissioners put it in page 33 of their **Tax and Duty Manual Part 29-02-03**:

> *"The aim of the project must be resolution of a scientific and/or technological uncertainty on a systematic basis… Software developments using known methodologies in standard development environments… would not typically advance technology and would not address or resolve technological uncertainty."*

Similarly, HMRC guidance in the UK emphasises that not all software development qualifies, even if it is part of a sophisticated commercial project:

> *"Routine debugging, adding new user interfaces, or migrating a platform to the cloud using standard methods, without technological uncertainty, is unlikely to qualify" (HMRC "How to identify qualifying R&D activities", 2023).*

What Makes R&D Qualify in Software?

To determine whether software work qualifies as R&D, tax authorities focus on **four essential criteria**:

1. **Scientific/Technological uncertainty** must exist.
2. The uncertainty must not be easily resolved using **publicly available knowledge or routine engineering**.
3. The team must undertake a **systematic, investigative process**, typically involving hypothesis formulation, testing, and iteration.
4. The goal must be to achieve a **scientific/technological advancement**, not merely a functional outcome.

The Canadian Income Tax Act defines experimental development as:

> *"Work undertaken for the purpose of achieving technological advancement for the purpose of creating new, or improving*

> *existing, materials, devices, products or processes, including incremental improvements thereto" (248(1) ITA, cited in National R&D Inc. v. The Queen, 2020).*

In Australia, the same principle is enacted in legislation. The **R&D Tax Incentive (RDTI)** only covers activities where:

> *"The outcome of the work cannot be known or determined in advance based on current knowledge, information or experience, and can only be determined by applying a systematic progression of work" (RDTI Guide to Interpretation 2020, p. 12).*

Examples of Qualifying Software R&D

Many software use cases can and do qualify across jurisdictions, provided the conditions above are met. Here are some examples found in case law and official guidance:

- Developing **new algorithms or mathematical models** for functionality that is not straightforward, such as predictive analytics, AI decision trees, or real-time optimization engines.

  As Irish guidance notes, qualifying work may involve:

  > *"Development of mathematical models or algorithms to achieve a desired functionality goal(s)... translating such models or algorithms into code and ensuring that the desired goal(s) can be achieved."*

- Ensuring that applications work **across diverse environments** or platforms where compatibility is not assured by existing methods. For example, ensuring stable performance of a client app across mobile and embedded systems.

- Building **integration mechanisms** where off-the-shelf tools don't support the required interoperability or scale.

- System-level improvements to **latency, throughput, or concurrency**, especially when re-architecting code or frameworks under uncertain conditions.

An illustrative example comes from the Canadian court decision in *National R&D Inc.*. The claimant attempted to create a multi-tiered SR&ED tracking platform using outdated tooling. The development involved non-trivial efforts to:

> "Overcome the paging limitations of SQL Server 2000… using the ASP response buffer to incrementally push the data to the browser… The work undertaken resulted in the creation of what Mr. Saini referred to as the 'Stitching Mechanism', a piece of code which allowed SQL statements to be constructed dynamically and allowed simple queries which utilized single-column sorting to return results" (*Dckt_2017-3837-IT-G_07-Jul-2020.pdf*, pp. 6–7).

This attempt to build a custom solution using existing but limited tools demonstrated how **technological advancement may occur even within legacy environments,** provided the underlying issue is unresolved by public knowledge and requires true experimental development.

Example

A company intends to develop a new Health Management System to manage patient health records. To support optimal health outcomes for patients the company wants to include the health tracking capabilities of wearable fitness devices and diet apps available in the market.

Challenge

The company identifies a number of popular health tracking products that capture relevant data for its new Health Management System. Each product uses different sub-systems that are not inter-operable. Many of the device platforms also don't use an Application Programming Interface (API) to access the data they collect. The company's key challenge is to work out how to integrate several different operating systems and health data sources.

Unknown outcome

A relevant solution was not identified from product and code repository searches or consultation with industry experts. The company then decided to undertake systematic research and development to determine if the challenge could be solved.

You are expected to search widely for current knowledge information or experience and keep records to show how you did this search.

Example - Extracted from the May 2024 Australian "Software-related activities and R&D Tax Incentive A guide to help you assess whether your software-related R&Dis eligible for the R&D Tax Incentive "

What Doesn't Qualify

The most common missteps involve claiming routine development work. All four tax authorities explicitly exclude:

- **User acceptance testing** done to confirm expected outcomes (Irish Revenue).
- **Reimplementing known features** using standard approaches.
- **Using third-party APIs or libraries** unless their use involved unexpected technical challenges that required deep investigation.
- **Upgrading systems or platforms** where the effort lies in configuration, not problem-solving.

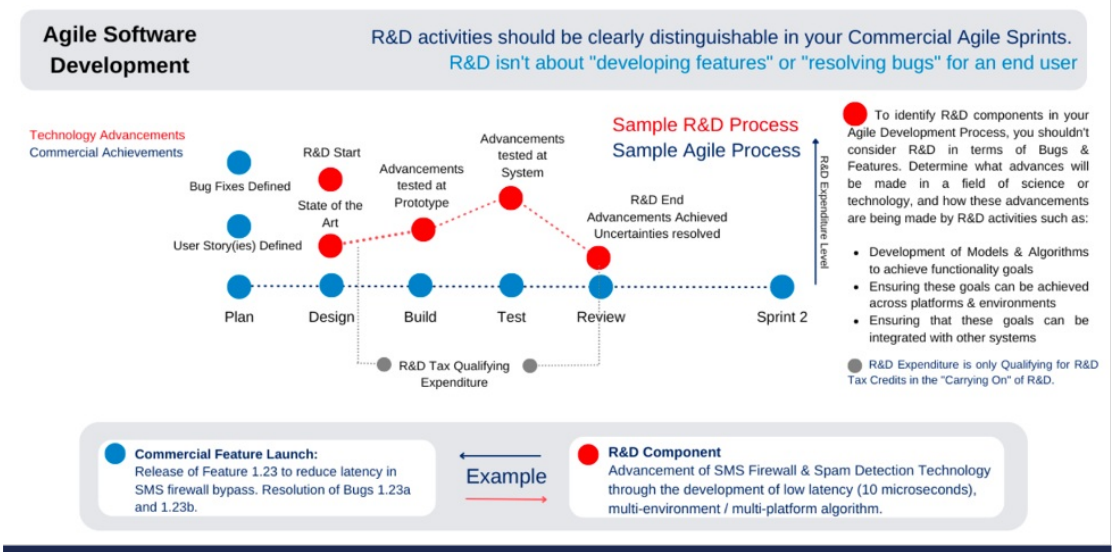
As the UK HMRC warns: *"You can't claim for work to package a product for sale or modify it for aesthetic or minor functional changes… Only the part of the work that sought to overcome the uncertainty qualifies"* (HMRC R&D guidance, p. 8).

Agile Development and Apportionment

Modern software teams often work in agile cycles, which blur the line between research and delivery. Both HMRC and Irish Revenue emphasize the importance of **apportioning** staff time and costs where R&D occurs in parallel with non-qualifying work.

From Irish Revenue: *"In agile development methodologies, qualifying and non-qualifying activity may take place simultaneously… project managers should apportion the staffing and other costs associated with each element of a development in a manner that appropriately reflects the balance of effort."*

This means that simply labelling an entire sprint or feature as “R&D” is not acceptable. Claimants must identify the investigative elements within that activity, show how they meet the qualifying conditions, and isolate their costs accordingly.



Extract from ReaDI-Watch Training Programme - R&D in Agile Product Delivery

How ReaDI-Watch Helps You Capture R&D in Software

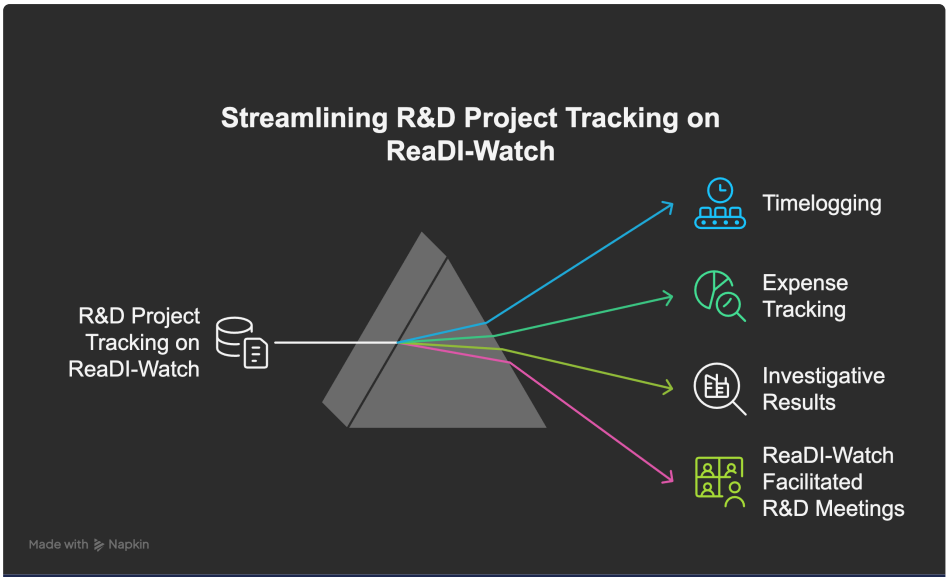
The challenge with identifying qualifying R&D in software is not just about interpretation, it's about **visibility**. The most innovative, uncertain, and potentially claimable work often doesn’t declare itself in your project tracker or show up in sprint retros. It hides in integration crises, undocumented workaround attempts, trial-and-error loops, and architecture decisions made under pressure.

ReaDI-Watch exists to solve precisely this problem. It brings **structure, recognition, and compliance-aligned tracking** to R&D, without breaking how your teams already work. Whether your uncertainty surfaces in a greenfield codebase or a Jira backlog firefight, ReaDI-Watch enables you to capture it.

We do this by:

- **Sequencing R&D meetings** around delivery cadences and milestones, surfacing real investigative work, even when it wasn't labelled as R&D at the time.
- **Tracking effort and cost** across software and engineering systems like Jira and ERP, allowing you to apportion qualifying time and expense with evidence-backed fidelity.
- **Capturing uncertainty and hypothesis-driven iteration** through guided interviews and facilitated documentation workflows.
- **Differentiating routine development from qualifying R&D** by aligning your real-world development activity with jurisdiction-specific tests, including UK, Ireland, Canada, and Australia.

This multi-pronged approach means your software teams don’t need to change their development process. Instead, ReaDI-Watch integrates with it, illuminating the qualifying work you’re already doing and backing it with the **narrative, structure, and auditability** required for tax authorities.



ReaDI-Watch Platform