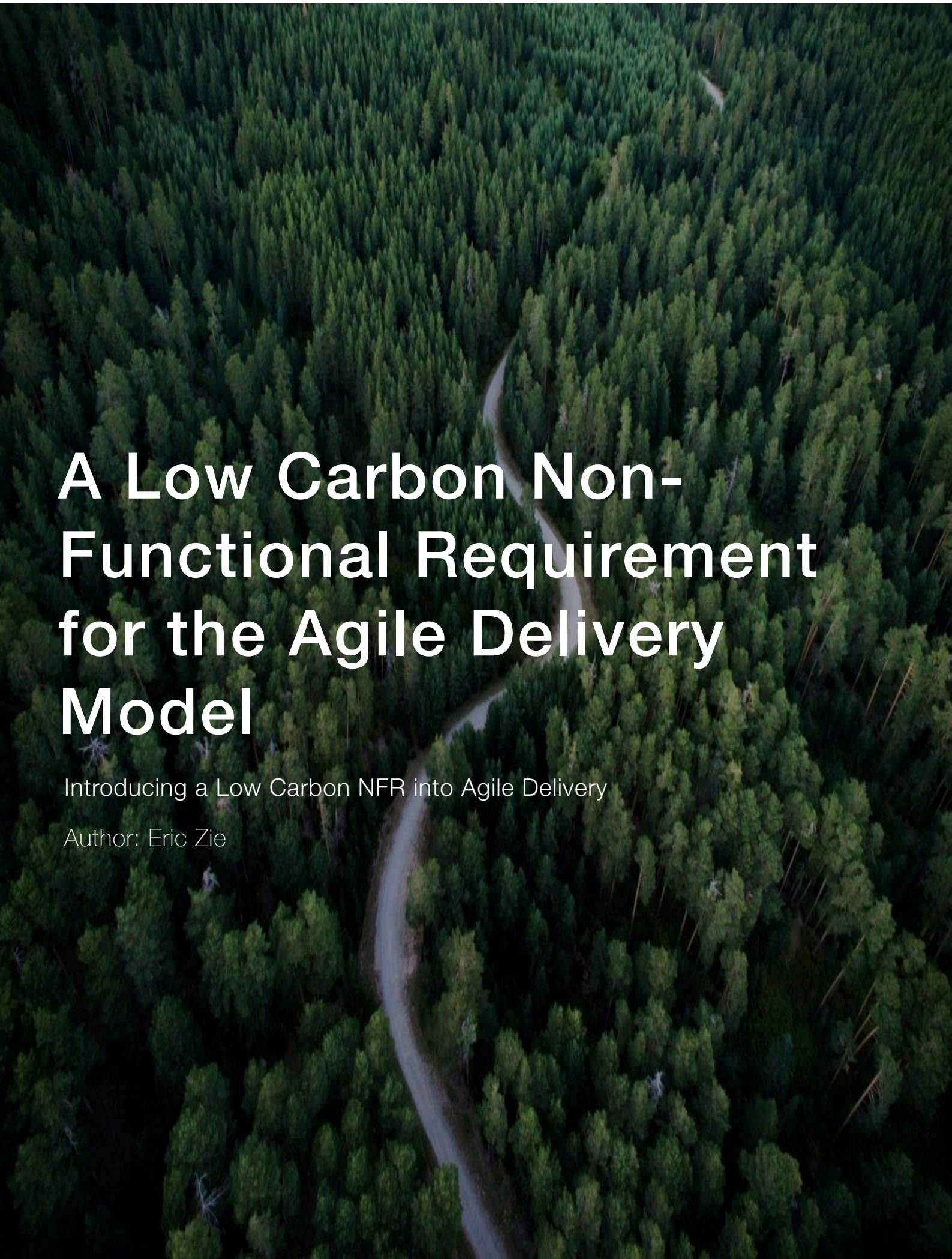


A Low Carbon Non-Functional Requirement for the Agile Delivery Model

Introducing a Low Carbon NFR into Agile Delivery

Author: Eric Zie

Defining a Low Carbon Non-Functional Requirement (NFR) as standard in the Agile Delivery Process

Why Does Low Carbon Software Matter?

Software runs our digital lives and businesses, but we use lots of energy to build it and need hardware, data centres and networks to keep it running. All of this consumes even more energy. The sector created 1.4 billion tonnes CO2e or 2.5% of global emissions in 2020 and consumed 4% of global energy – the same as all the air travel taken in the world in a year. Data centres use about 1% of global electricity – 160TWh of energy. That's the same as powering 32 million homes. And these numbers are increasing every year as our usage of digital services continues to grow.

We can help by lowering the energy demands of software on the technology value chain and reduce the creation of millions of tonnes of CO2e. By making more efficient use of ICT we not only lower energy consumption but also extend the lifespan of the embodied carbon in all devices and infrastructure components needed to run software. By designing and building carbon efficient software we can reduce total energy demands of the entire technology value chain.

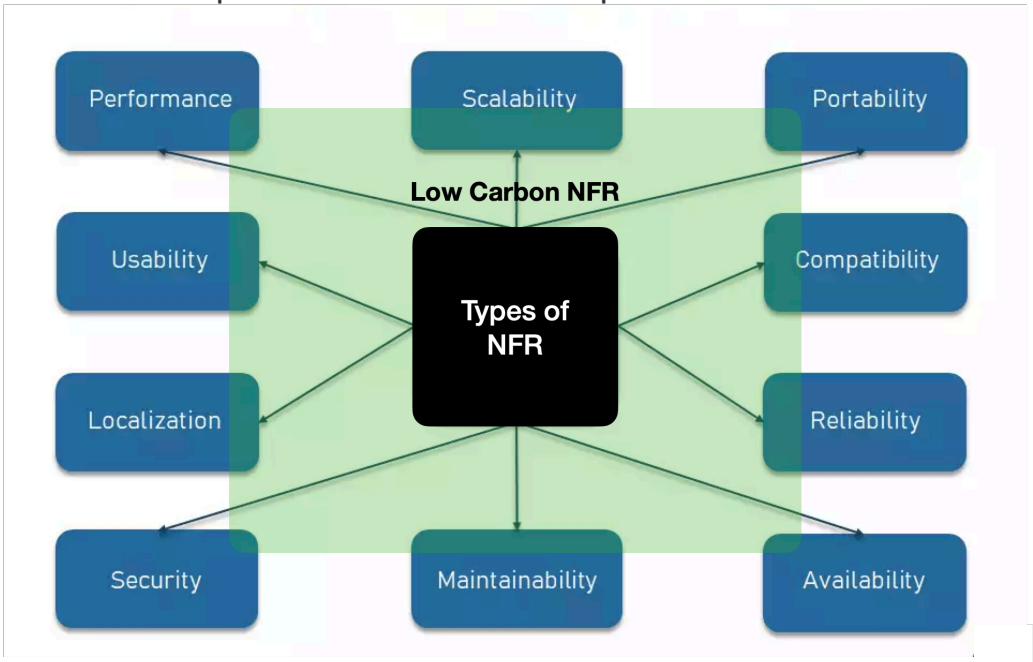
Including the Low Carbon requirements defined below as an NFR into the specification of a new software product at design stage and as part of the agile delivery process can be a major influencer on decarbonisation and help us move toward a lower carbon IT future. Please consider doing this in any new project, it is a simple way to ensure that sustainability is at the core of your delivery, and not seen as 'something extra to do'. You will not only gain the benefits of applying the rigour of agile practices but also enable your quality assurance processes to help track and measure compliance and progress. This is an action that will immediately help make a fundamental difference in the way we build more carbon awareness into our digital products and services and truly embed a culture of decarbonisation within the software development process.

Section 1: Low Carbon as a Non-Functional Requirement

Non-functional Requirements (NFRs)

Understanding why NFRs are vital to the delivery of customer-centric, reliable and robust software products is an important place to begin. The summarised points below cover the key definition and attributes:

- **NFRs commonly define system attributes** such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs. Also known as system qualities, non-functional requirements are just as critical as functional Epics, Capabilities, Features, and Stories. We believe that Low Carbon is now a key NFR that will have implications across all requirements.



- **NFRs ensure the usability and effectiveness of the entire system**. Failing to meet any one of them can result in systems that fail to satisfy internal business, user, or market needs, or that do not fulfil mandatory requirements imposed by regulatory or standards agencies. In some cases, non-compliance can cause significant legal issues (for example, privacy, security and safety).

- **NFRs are persistent qualities and constraints** that, unlike functional requirements, are typically revisited as part of the Definition of Done (DoD) for each Iteration, Program Increment (PI), or release. NFRs

influence all backlogs: Team, Program, Solution, and Portfolio. It is therefore critical that proper definition and implementation of NFRs is undertaken. Over-specification, and the solution may be too costly to be viable; under-specify or underachieve them, and the system will be inadequate for its intended use.

- **NFRs are not themselves backlog items**. They are constraints on development that limit some degree of design freedom for those building the system. These constraints are often defined in the acceptance criteria for multiple backlog items. In a Low Carbon IT example, reducing image sizing whilst maintaining an acceptable UI experience but balanced to achieve a target carbon intensity per software unit is a requirement for all products in the suite. A user friendly UI is a functional requirement, while reduced image sizing is a constraint to achieve the desired Low Carbon NFR outcome. Any backlog item building UI functionality would reference the reduced image sizing constraint in its acceptance criteria.

We believe that a Low Carbon requirement needs to be defined and incorporated alongside other traditional NFRs, and that the approach to introducing sustainability based qualities and constraints highlighted above become standard in the agile delivery process.

Low Carbon NFR and Software Quality

To deliver sustainably minded software products it is key to build a set of Low Carbon specifications that describe the system's operational capabilities and constraints and attempt to maintain its functionality. These are basically the requirements that outline how well it will operate alongside 'standard' NFRs like speed, security, reliability, data integrity, etc.

Note: the focus here is on a Low Carbon NFR for a software product. It is not intended to cover sustainability based functionality from a user perspective, e.g., a user story related to delivering a carbon footprint calculation for a client. The focus here is on managing the carbon impact of the software product itself.

Non-functional requirements specify the quality attributes of the system, hence their second name — *quality attributes*. Continuing the reduced

image sizing example, a non-functional requirement can be the Kb sizing constraints with which a system must perform visuals to satisfy user expectations, *“The standard image size across UI components must be limited to 50Kb, and use the appropriate caching techniques to reduce superfluous reloading of images once displayed”.*

NFRs are therefore implicit attributes of software quality. The IEEE standard 1061-1998 (Standard for a quality metrics methodology) states that *“Software quality is the degree to which software possesses a desired combination of quality attributes”* where the definition of these qualities depends on what is required of the system to be built. By incorporating a Low Carbon NFR into the specification and measuring delivery according to the overall quality criteria of the software product being developed, it will be possible to ensure that sustainability is at the heart of new product delivery.

The Impact of a Low Carbon NFR on Solution Development

Non-functional requirements can have a substantial impact on solution development and testing. Architects and developers should be aware of the implications when specifying them. For example, a statement like “0.0145 carbon intensity” may increase development effort exponentially more than “0.0156 carbon intensity.” The impact of the NFR must be well understood by those defining requirements.

In many cases, applying Set-Based Design can keep options open by initially specifying NFRs as a range, and that is why we recommend setting a target rating (e.g., A … E) to aid boundary and target setting. Teams explore the solution space and gain additional knowledge that leads to a better economic and climate based decision. This is a new paradigm and some education and understanding of the implications of these choices from a sustainable IT perspective may well be required. Imagine a future where carbon impact and availability are both considered with equal importance, that is what we are attempting to achieve with the Low Carbon NFR.

There will always be value in achieving the highest performing Low Carbon measure - and the value is beyond traditional economic indicators. We do however recognise that achieving the highest levels of performance may not always be achievable or cost-effective for a particular product in a specific scenario, and may require adjustments

elsewhere in the system's operational environment. The solution's Economic Framework should contain criteria to evaluate the Low Carbon NFR, and this should be viewed in the context of trade-offs with costs and other considerations.

NFRs will and should affect Suppliers and their knowledge and ability to achieve performance targets should inform NFR specifications and the Economic Framework applied to software selection. For more information on the GoCodeGreen Low Carbon Software Specification, aimed specifically at low carbon software vendor selection contact connect@gocode.green.

Specifying a Low Carbon NFR

Like all other requirements, NFRs must be quantified for clarity to ensure stakeholders' needs are clearly understood by everyone. The following steps should be considered as part of a Low Carbon NFR specification:

- **Define** the NFR's quality, including its name, scale, and method to measure.
- **Quantify** the NFR's measurable values, including the current measured value (baseline), the value to achieve (target), and the value that becomes unacceptable (constraint).
- **Bounded** – When they lack bounded context, NFRs may be irrelevant and lead to significant additional work.
- **(Inter)dependent** – Generally, NFRs should be independent of each other so that they can be evaluated and tested without consideration of, or impact from, other system qualities. However, for a Low Carbon NFR there will be balances and interactions to be made against other NFRs - this will make specification more challenging but is a necessity as compromises across both functionality and other NFR targets will be required.
- **Negotiable** – Understanding NFR business drivers and bounded context mandates negotiability.
- **Testable** – NFRs must be stated with objective, measurable, and testable criteria.

In addition to the steps above, two further additional considerations are key when writing and setting a Low Carbon NFR:

- **Customer Concern** - sustainability is increasingly becoming a key customer need - and demonstrating the low carbon credentials of the software products that now deliver so many of the digital services being consumed will be scrutinised and valued. A future where customer decisioning and choice will be based on the low carbon rating of a digital service is becoming a reality. To this end, it is critical that the Low Carbon NFR is introduced into projects early in the design so that customer expectations can be understood, validated and met.
- **Design Criteria** - to ensure correctness of approach (completeness, consistency, traceability, verifiability) it is important that clear metrics and measurement criteria are set to support the solution design to achieve the NFR requirement. It is highly recommended that appropriate measurement, baselining and tracking capabilities are identified at design stage and selection of tooling made to ensure targets can be managed through the software delivery process.
- **Quality Control** - being clear on the quality criteria to achieve the low carbon targets for a software product are key, and will ensure that the specification is being met through the lifecycle of the software delivery process as well as in use stage. Inserting the quality attributes and metrics into the overall project requirement will enable quality managers and product owners to track and trace appropriately as an embedded part of the agile delivery process.

These different aspects of a Low Carbon NFR specification will be demonstrated in the example in Section 2. It is recommended that this is directly inserted into the NFR specifications of your next project.

Section 2: A Low Carbon NFR

Low Carbon NFR Scope & Definition

The scope of a Low Carbon NFR should cover:

> **The Production and Use lifecycle stages of the software product being designed, built (covering both original construction and revisions) and operated.**

The Low Carbon NFR should ultimately be defined as:

> **The ability of the system to behave consistently in a user-acceptable manner whilst minimising energy consumption and therefore its carbon impact within the environment in which it was intended.**

Minimising energy consumption and carbon impact can be defined in terms of the following quality attributes and metrics:

Ref	Low Carbon Quality Attribute	Metric & Quantification
1A	Baseline Carbon Impact Rating for Production - Build	Standardised rating system aligned to characteristics of the software product being assessed for baseline measurement, example: *'A' Rated = #Software units x Complexity/Time x Specific Parameter Selection for Software Product being assessed / Software Carbon Intensity Score*
1B	Baseline Carbon Impact Rating for Production - Release / per Increment of Change	Standardised rating system aligned to characteristics of the software product being assessed for baseline measurement, example: *'B' Rated = #Software units x Complexity/Time x Specific Parameter Selection for Software Product being assessed / Software Carbon Intensity Score*

2	Baseline Carbon Impact Rating for Use - Operate	Standardised rating system aligned to characteristics of the software product being assessed for baseline measurement, example: *'C' Rated = #Software units x Complexity/Time x Specific Parameter Selection for Software Product being assessed / Software Carbon Intensity Score*
3A	Baseline Carbon intensity cost per unit of software in Production	Production Carbon Intensity Rating per unit of software during the development of the core software product. Clear boundaries and allocations should be identified to enable consistency of calculation. Example: *Carbon intensity = tCO2e per identifiable unit of software*
3B	Baseline Carbon cost per unit of software in Release / per Increment of Change	Production Carbon Intensity Rating per unit of software during the development of each release of the software product. Clear boundaries and allocations should be identified to enable consistency of calculation. Example: *Carbon intensity = tCO2e per identifiable unit of software*
4	Baseline Carbon cost per unit of software in Use	Production Carbon Efficiency Rating per unit of software during the operation of the software product. Clear boundaries and allocations should be identified to enable consistency of calculation. Example: *Carbon Efficiency = tCO2e per identifiable unit of software*
5A	Target Rating for Core Build	Standardised rating system aligned to characteristics of the software product being assessed for target measurement, example: *'A' Rated = #Software units x Complexity/Time x Specific Parameter Selection for Software Product being assessed / Software Carbon Intensity Score*

5B	Target Rating per Release / Increment	Standardised rating system aligned to characteristics of the software product being assessed for target measurement, example: *'B' Rated = #Software units x Complexity/Time x Specific Parameter Selection for Software Product being assessed / Software Carbon Intensity Score*
6	Target Rating for Use / Operation	Standardised rating system aligned to characteristics of the software product being assessed for target measurement, example: *'C' Rated = #Software units x Complexity/Time x Specific Parameter Selection for Software Product being assessed / Software Carbon Intensity Score*
7A	Target Carbon intensity cost per unit of software in Production	Production Carbon Intensity Rating per unit of software during the development of the core software product. Clear boundaries and allocations should be identified to enable consistency of calculation. Example: *Carbon intensity = tCO2e per identifiable unit of software*
7B	Target Carbon cost per unit of software in Release / per Increment of Change	Production Carbon Intensity Rating per unit of software during the development of each release of the software product. Clear boundaries and allocations should be identified to enable consistency of calculation. Example: *Carbon intensity = tCO2e per identifiable unit of software*
8	Target Carbon cost per unit of software in Use	Production Carbon Efficiency Rating per unit of software during the operation of the software product. Clear boundaries and allocations should be identified to enable consistency of calculation. Example: *Carbon Efficiency = tCO2e per identifiable unit of software*

These quality attributes are considered to be universal and should persist across the entire product development journey, with baselining and then targets for each being set and tracked for the duration of delivery and key to the final definition of done. They should be considered distinct and different to the underlying technical techniques for achieving the quality attribute, these will have their own measures as part of the software engineering approaches undertaken during product development. Consider these to be the master Low Carbon constraints supporting the NFR.

The quality attributes must be measurable and quantifiable if to be of value. To this end it is recommended that appropriate measurement and baselining capabilities focused on the carbon impact of software are selected and deployed. These must be selected based on cost of use, consistency and repeatability of measurement.

The metric targets should form part of the overall definition of done for the project, and where appropriate individual program increments (PI) if specific Low Carbon deliverables have been included in the PI.

Find out More

To discuss how to implement a Low Carbon NFR and gain access to an example NFR specification, contact connect@gocode.green, who will be happy to discuss an example of how to implement this in practice and methods for measuring and tracking performance against your Low Carbon NFR.