Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Monitorización remota de ensayos térmicos en cámaras climáticas

Autor: Carlos Herrera Cíes
Tutor: Fernando Muñoz Chavero

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Monitorización remota de ensayos térmicos en cámaras climáticas

Autor:
Carlos Herrera Cíes

Tutor:
Fernando Muñoz Chavero
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Grado: Monitorización remota de ensayos térmicos en cámaras climáticas

Autor: Carlos Herrera Cíes

Tutor: Fernando Muñoz Chavero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis amigos

A mis profesores

A mis compañeros

Agradecimientos

Después de estos últimos cuatro años de mi vida solo me queda agradecer a todas las personas que han estado ahí apoyándome en todo lo que estaba en sus manos, porque sin su ayuda esto no habría sido posible.

En primer lugar, me gustaría agradecer el cariño y ánimo que me han brindado mis padres y mi hermano con esas llamadas telefónicas que me hacían sentir como en casa.

No puedo olvidar a mis compañeros y amigos, siempre dispuestos a compartir sus experiencias, haciéndome ver que no estaba solo.

Por último, si bien no menos importante, me gustaría agradecerle tanto a Fernando Muñoz Chavero, mi tutor, como a la empresa Alter Technology TÜV NORD la oportunidad que se me ofreció de realizar este proyecto. Gracias también a todos los trabajadores, en especial a Jose Luis, por su ayuda totalmente desinteresada.

Carlos Herrera Cíes

Resumen

Este proyecto tiene como objetivo exponer el desarrollo de una aplicación software para la monitorización remota de ensayos térmicos en cámaras climáticas. Además, se detalla la infraestructura software necesaria para el correcto funcionamiento del sistema, así como la explicación del cálculo de las incertidumbres asociadas a los equipos empleados, imprescindible para conseguir un seguimiento y una caracterización fiables de la evolución de los ensayos. Por último, se propondrán posibles líneas futuras alcanzables a partir de las herramientas y conocimientos desarrollados en este proyecto.

El cálculo de incertidumbres cumple con la Norma Europea EN 60068-3-11:2008 para las condiciones en cámaras de ensayos climáticos.

Abstract

The aim of this project is to expose the development of a software application created to monitor remotely thermal tests in climatic chambers. In addition, the infrastructure needed to run the system properly will be shown, as well as the uncertainty calculations related to laboratory equipment, essential to achieve an accurate tracking of test progress. Finally, possible future goals will be proposed to continue this work.

Uncertainty calculations keep to European Norm 60068-3-11:2008 for the conditions in climatic chambers.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xv**

Índice de Tablas **xviii**

Índice de Figuras **xx**

Notación **xxiii**

1 Introducción **1**
1.1. Motivación *1*
1.2. Objetivos *1*
1.3. Estructura del trabajo *2*

2 Conceptos relevantes **4**
2.1. Cámaras climáticas *4*
2.2. Termorresistencias y termistores *5*
2.2.1. Termorresistencias 5
2.2.2. Termistores 6
2.3. Virtual Lab *6*

3. Monitorización **11**
3.1. Ensayos *11*
3.1.1. Ensayo cíclico compuesto de temperatura y humedad 11
3.1.2. Ensayo cíclico de calor húmedo 14
3.1.3. Variación de temperatura 18
3.1.4. Frío 21
3.1.5. Calor seco 22
3.2. Equipos utilizados *23*
3.2.1 Cámara climática VCL 7006 23
3.2.2 Sondas de temperatura PT100 25
3.3. Calibración de los instrumentos *26*
3.3.1. Calibración de las sondas 26
3.3.2. Calibración de la cámara 27
3.4. Cálculo de incertidumbre *28*
3.4.1. Documentos de referencia para el cálculo de incertidumbres 28
3.4.2. Obtención de la incertidumbre expandida 28

3.4.3. Fuentes de incertidumbre 30
3.4.4. Ejemplo práctico: ensayo de frío 31

4. Herramientas y arquitectura software 34
4.1. Base de datos 34
4.1.1. Conceptos sobre bases de datos 34
4.1.2. Microsoft SQL Server Management Studio 2017 36
4.1.3. Diseño de la base de datos 38
4.2. IDE y soporte software 46
4.2.1. .NET Framework 46
4.2.2. Lenguaje C# 46
4.2.3. Microsoft Visual Studio 47

5 Aplicación desarrollada 49
5.1. Objetivos de la aplicación 49
5.2. Paquetes de NuGet 50
5.2.1. AutoMapper 50
5.2.2. EntityFramework 50
5.2.3. Ninject 50
5.3. Solución propuesta 51
5.3.1. View Models 51
5.3.2. Mapping 52
5.3.3. Infrastucture 53
5.3.4. Data Readers 53
5.3.5. Formularios 55
5.4. Funcionamiento de la aplicación 65

6 Conclusiones y líneas futuras 70
6.1. Conclusiones 70
6.2. Líneas futuras 70

Referencias 73

Anexo 75
Data Readers 75
ITempReader 75
TempData 75
TempReader 75
Formularios 77
Form1 77
TestInProgress 83
SegmentsForm 98
ErrorForm 99
CancelMonitoringForm 99

Índice de Tablas

Tabla 1: Elementos más comunes en la fabricación de RTDs. 6
Tabla 2: Valores de calibración de las sondas PT100. 27
Tabla 3: Valores de calibración de la cámara climática LE0609.000 27
Tabla 4: Cálculo de incertidumbre para un ensayo de frío real. 32
Tabla 5: Tabla Alarm. 38
Tabla 6: Tabla BatchIDTestChamber. 39
Tabla 7: Tabla BatchIDTestChamber_Probe. 39
Tabla 8: Tabla Chamber. 40
Tabla 9: Tabla Chamber_Probe 40
Tabla 10: Tabla Probe. 42
Tabla 11: Tabla ProbeCalibration. 43
Tabla 12: Tabla ProbeType. 43
Tabla 13: Tabla ProgramTest. 44
Tabla 14: Tabla ProgramTest_Alarm. 44
Tabla 15: Tabla Segment para IdProgramTest igual a 4. 45

Índice de Figuras

Figura 1: Cámara climática de estabilidad del fabricante MPcontrol. 4
Figura 2: Cámara ambiental Vötsch VC 4018 para ensayos climáticos. 5
Figura 3: Prototipo de interfaz de Virtual Lab para un ensayo genérico. 8
Figura 4: Prototipo de la interfaz de Virtual Lab para ensayos con el microscopio acústico. 8
Figura 5: Fase de preacondicionamiento (fuente: UNE-EN 60068-2-38). 12
Figura 6: Humedad relativa (arriba) y temperatura (abajo) del ciclo con exposición al frío (fuente: UNE-EN 60068-2-38). 13
Figura 7: Humedad relativa (arriba) y temperatura (abajo) del ciclo sin exposición al frío (fuente: UNE-EN 60068-2-38). 14
Figura 8: Periodo de estabilización en el ensayo cíclico de calor húmedo (fuente: UNE-EN 60068-2-30). 15
Figura 9: Ciclo con variante 1 (fuente: UNE-EN 60068-2-30). 16
Figura 10: Ciclo con variante 2 (fuente: UNE-EN 60068-2-30). 17
Figura 11: Cálculo del tiempo de exposición del ensayo (fuente: UNE-EN 60068-2-14). 18
Figura 12: Ciclo del ensayo Na (fuente: UNE-EN 60068-2-14). 19
Figura 13: Ciclo del ensayo Nb (fuente: UNE-EN 60068-2-14). 21
Figura 14: Cámara climática VCL 7006 de Vötsch. 23
Figura 15: Límites operacionales de temperatura y humedad de la VCL 7006. 24
Figura 16: Pantalla de SIMPATI. 25
Figura 17: Esquema y distribución espacial de las sondas en la cámara. 25
Figura 18: Curva ensayo de frío real. 32
Figura 19: Ventana Object Explorer de Microsoft SSMS. 36
Figura 20: Consulta de ejemplo en Microsoft SSMS. 37
Figura 21: Pestaña de modificación de los registros de la tabla PrimaryMeasures. 37
Figura 22: Pestaña de diseño de los campos de la tabla PrimaryMeasures. 38
Figura 23: Perfil del ensayo de IdProgramTest igual a 4. 45

Figura 24: Desarrollo de un paquete NuGet (fuente: https://docs.microsoft.com/en-us/nuget/what-is-nuget) 50
Figura 25: View Models del proyecto. 52
Figura 26: Archivos para el mapeo. 52
Figura 27: Archivos de DataReader. 53
Figura 28: Menú principal de la solución. 56
Figura 29: Método asociado al evento SelectedIndexChanged. 57
Figura 30: Formulario de los segmentos del programa. 59
Figura 31: Cuadro de error. 60
Figura 32: Ventana de monitorización. 60
Figura 33: Argumentos del método ReportProgress. 62
Figura 34: Ventana de cancelación del análisis. 64
Figura 35: Menú principal. 65
Figura 36: Menú principal para el ensayo de frío. 65
Figura 37: Ventana de monitorización al inicio del análisis. 66
Figura 38: Datos de los dos primeros tramos analizados. 66
Figura 39: Tabla CorrectedMeasures de las sondas para el ensayo de frío. 67
Figura 40: Ventana de monitorización al finalizar el tercer segmento. 67
Figura 41: Ventana de monitorización al completar el análisis. 68
Figura 42: Registros al finalizar el análisis. 68

Notación

DUT	Device Under Test
ATN	ALTER TECHNOLOGY TÜV NORD
ICH	International Conference on Harmonisation
ºC	Grado centígrado
h	Hora
min	Minuto
K	Kelvin
kW	Kilovatio
kWh	Kilovatio hora
l	Litro
h.r.	Humedad relativa
kg	Kilogramo
mm	Milímetro
RTD	Resistance Temperature Detector
ITS-90	International Temperature Scale of 1990
T	Temperatura
PTC	Positive Temperature Coefficient
NTC	Negative Temperature Coefficient
CEM	Centro Español de Metrología
EIT	Escala Internacional de Temperatura de 1990
IEC	International Electrotechnical Commission
VIM	Vocabulario Internacional de Metrología
min	Minuto
GUM	Guide to the expression of Uncertainty in Measurement
XML	Extensible Markup Language
SQL	Structured Query Language
SSMS	SQL Server Management Studio
NoSQL	Not Only SQL
IDE	Integrated Development Environment
CLR	Common Language Runtime

1 INTRODUCCIÓN

If knowledge can create problems, it is not through ignorance that we can solve them.

- Isaac Asimov -

Pese al gran avance que ha sufrido el campo de la automatización en los últimos años, aún hoy sigue habiendo procesos que requieren el trabajo o la supervisión manual de un usuario para funcionar correctamente. Sin embargo, muchas de estas actividades carecen de la elevada eficiencia que podrían disponer si estuviesen más automatizadas. De esta manera, no es extraño observar como poco a poco las empresas están desarrollando sistemas automáticos para la supervisión remota de sus procesos productivos. Por tanto, este documento ha de tomarse como un ejemplo práctico que pone de manifiesto esta evolución hacia alcanzar el mayor rendimiento posible en tareas comúnmente poco o nada automatizadas.

1.1. Motivación

El presente documento detalla la realización de una aplicación software para la monitorización, supervisión y obtención de datos de ensayos térmicos para cámaras climáticas genéricas. De esta manera, se sientan las bases para la automatización de otros ensayos, tanto climáticos como de cualquier otro tipo, realizados en Alter Technology TÜV NORD.

Hasta ahora, la mayoría de los ensayos debían ser realizados sin saber hasta su finalización parcial o total si habían estado en las condiciones establecidas por las normas de referencia. Por consiguiente, podría ocurrir que desde etapas iniciales se incumpliese la norma, quedando invalidados los resultados de dicho ensayo para el cliente. Este hecho implicaría una pérdida en rendimiento pues se invierte tiempo y recursos en una actividad que finalmente no reporta ningún beneficio. En ensayos de larga duración, como muchos de los ensayos climáticos, es más patente el perjuicio que pueden ocasionar estos intentos fallidos, donde el tiempo de realización puede ser de días enteros.

Por estos motivos, es de gran interés establecer unas pautas sobre el camino a seguir para la supervisión automática completa, así como una aplicación práctica y funcional para ensayos térmicos.

1.2. Objetivos

El objetivo principal de este proyecto es la realización de una aplicación software que monitorizará y realizará los cálculos necesarios para comprobar que las condiciones de temperatura del ensayo que supervise son las establecidas por el cliente. Para ello, deberá obtener los datos primarios – medidas directas de los sensores -, las calibraciones de los instrumentos y las especificaciones de la monitorización de una base de datos. Posteriormente, deberá procesar los datos primarios a partir de las calibraciones para calcular las incertidumbres asociadas, generando unas medidas corregidas y comprobando que son acordes a las características del ensayo. Estos datos procesados se guardarán a su vez en la base de datos.

La aplicación software será una aplicación .NET de Windows Forms que estará desarrollada en lenguaje C#, a partir del IDE de Visual Studio 2019. Dispondrá de una infraestructura específica para la conexión con la base de datos. Además, la aplicación deberá ofrecer al usuario la posibilidad de elegir el ensayo que se desea monitorizar, las alarmas y los instrumentos de medida - cámara y termómetros -, a través de una interfaz sencilla y práctica.

Por otra parte, la base de datos será de tipo relacional, realizada gracias al entorno proporcionado por Microsoft SQL Server Management Studio 17. Deberá almacenar datos relevantes del test, los datos primarios recopilados, los programas de los ensayos, las alarmas disponibles y las medidas corregidas, entre otros. En capítulos posteriores se explicará en detalle la estructura de esta base de datos.

1.3. Estructura del trabajo

La estructura que seguirá este documento es la siguiente:

- **Capítulo 1: Introducción.**

  Contiene una breve introducción sobre los motivos de realizar este proyecto, así como los objetivos principales a alcanzar y la estructura del documento.

- **Capítulo 2: Conceptos relevantes.**

  Expone las características principales de las cámaras climáticas, qué utilidad tienen y que tests pueden realizar. Además, se describe el concepto *Virtual Lab* desarrollado por Alter Technology TÜV NORD, idea que contextualiza el alcance de este proyecto.

- **Capítulo 3: Monitorización.**

  Explica los tests que la aplicación es capaz de monitorizar, en concreto el ensayo de calor seco, de frío y el ciclado térmico. A continuación muestra los equipos de medida utilizados, tanto las cámaras como los sensores de temperatura. Por último, trata las calibraciones de los instrumentos y el cálculo de las incertidumbres asociadas.

- **Capítulo 4: Herramientas y arquitectura software.**

  Desarrolla la base de datos SQL, los motivos de su elección y el programa de administración Microsoft SQL Server Management Studio, además del entorno de desarrollo proporcionado por Visual Studio 2019 para la creación de aplicaciones .NET de Windows Forms.

- **Capítulo 5: Aplicación desarrollada.**

  Esta sección ilustra todo lo relacionado con la aplicación final, desde sus pantallas, pasando por el cálculo de la incertidumbre hasta la conexión con la base de datos.

- **Capítulo 6: Conclusiones y líneas futuras.**

  Recopila las conclusiones y se reflexiona sobre las posibles líneas futuras a explorar a partir de este proyecto.

2 CONCEPTOS RELEVANTES

En este capítulo se exponen las características principales de las cámaras climáticas convencionales, qué función tienen y por qué son tan importantes en la industria. A continuación, se introducirá el concepto de *Virtual Lab*, desarrollado por Alter Technology TÜV NORD en su afán por obtener un mayor rendimiento y calidad en los servicios que presta a sus clientes, en el que se engloba el contexto de este trabajo.

2.1. Cámaras climáticas

Las cámaras climáticas son equipos diseñados para establecer condiciones en su interior tanto de temperatura como de humedad relativa en las que efectuar ensayos que verifican la calidad y el comportamiento de dispositivos bajo testeo o DUTs. Hay otras cámaras que pueden incluso controlar la iluminación del habitáculo para variar el fotoperiodo y así realizar estudios con entes biológicos.

Se suele hacer una distinción entre las cámaras de estabilidad y las cámaras de ensayos. Las cámaras climáticas de estabilidad son aquellas que facilitan estudios de estabilidad. Un claro ejemplo de este tipo se puede encontrar en la industria farmacéutica, donde los estudios de estabilidad para productos médicos tienen una larga duración en condiciones ambientales prácticamente invariantes. De hecho, uno de los estudios marcados por la normativa de la ICH establece que el producto debe ser almacenado durante 12 meses en condiciones de 25 ºC ± 2 ºC de temperatura y 60 % ± 5 % de humedad relativa.

Figura 1: Cámara climática de estabilidad del fabricante MPcontrol.

Por otra parte, las cámaras climáticas de ensayos tienen la función de reproducir condiciones ambientales extremas y normalmente con bastante gradiente térmico, es decir, con cambios de temperatura relativamente rápidos - en torno a 1 o 2 ºC por minuto -. Asimismo, estos cambios de temperatura suelen repetirse en periodos programados previamente. El número de estos ciclos puede variar dependiendo de las condiciones que impongan la normativa o el cliente. Por ejemplo, en algunos ensayos de ciclado térmicos realizados en ATN, se pueden llegar a producir 500 ciclos de 15 minutos cada uno con rampas de subida y bajada de temperatura de 2 ºC por minuto entre puntos de operación extremos de 150 y -60 ºC.

Figura 2: Cámara ambiental Vötsch VC 4018 para ensayos climáticos.

2.2. Termorresistencias y termistores

En este apartado se definirán dos dispositivos encargados de la medición de la temperatura: las termorresistencias y los termistores.

2.2.1. Termorresistencias

Una termorresistencia es un sensor que se basa en la variación que sufre la resistencia eléctrica con la temperatura. También se denominan RTD. La termorresistencias más comunes están compuestas por un alambre fino soportado por un material aislante y todo eso, a su vez, encapsulado. Este elemento se introduce en un tubo metálico. Luego se rellena con un material aislante y se sella con cemento para prevenir la absorción de humedad.

La linealidad de la variación de la resistencia depende del material, siendo el más exacto el platino, seguido del cobre y del níquel, como indica la tabla 1.

La expresión de la resistencia del dispositivo tiene forma polinómica, con la forma de las ecuaciones de Callendar – Van Dusen. Tomando como referencia la ITS-90 tal y como indican los certificados de calibración de las sondas que se emplearán en los ensayos, las ecuaciones para la obtención de la resistencia a partir de la temperatura son:

$$R(T) = R_0 \left[1 + AT + BT^2 \right]$$

Ecuación 1: Ecuación de Callendar – Van Dusen para temperaturas positivas.

$$R(T) = R_0 \left[1 + AT + BT^2 + CT^3 \left(T - 100 \right) \right]$$

Ecuación 2: Ecuación de Callendar – Van Dusen para temperaturas negativas.

Siendo R_0 la resistencia a la temperatura de referencia, T la temperatura medida y A, B y C coeficientes propios de cada termómetro en particular obtenidos a partir de las diferentes calibraciones de la sonda.

Tabla 1: Elementos más comunes en la fabricación de RTDs.

Material	*Rango de operación (ºC)*	*Precisión (ºC)*
Platino	De -200 a 900	0.01
Níquel	De -150 a 300	0.50
Cobre	De -200 a 120	0.10

2.2.2. Termistores

Aunque el concepto es el mismo – buscar una variación de la resistencia con la temperatura –, los termistores se diferencian de las RTD en su composición: los termistores están fabricados con un material semiconductor, normalmente como una mezcla sintetizada de óxidos metálicos. Los metales utilizados más comunes son níquel, manganeso, titanio, cobre o hierro.

Frecuentemente se les encuentra montados en sondas para protegerlos adecuadamente debido a su pequeño tamaño. Estos recubrimientos suelen ser de acero inoxidable o aluminio.

Se pueden diferenciar dos clases de termistores: los NTC y los PTC. Los termistores NTC tienen como característica principal un coeficiente de temperatura negativo. De esta manera, un incremento de temperatura generaría un decremento de resistencia. Los termistores PTC tienen una relación entre temperatura y resistencia proporcional.

Los termistores obedecen a unas ecuaciones del tipo exponencial que se derivan de un comportamiento poco lineal - para pequeños incrementos de temperatura se produce un gran aumento en el valor de la resistencia -.

2.3. Virtual Lab

Durante los últimos años se ha ido forjando la concepción de que una nueva revolución industrial, la que sería la cuarta, estaba surgiendo. Esta nueva forma de ver la industria y los procesos productivos se ha denominado Industria 4.0. Cada una de las anteriores revoluciones industriales ha tenido características únicas: la primera se llevó a cabo principalmente gracias a la invención de la máquina de vapor y la mecanización, la segunda se distinguió por la producción en masa y las cadenas de montaje, y la tercera y más reciente se caracterizó por los avances en el campo de las tecnologías de la comunicación y la información. En cambio, la Industria 4.0 tiene como elementos centrales la inteligencia artificial, el procesamiento de grandes cantidades de datos – *big data* – junto con algoritmos para procesarlos y la interconectividad inteligente de un número descomunal de dispositivos. Este nuevo planteamiento ha sido adoptado por muchas empresas buscando la modernización de sus servicios así como de sus infraestructuras informáticas.

El concepto de Virtual Lab nace a partir de esta idea: ofrecer al cliente un servicio integral donde pueda ver la evolución del producto a lo largo de todas las etapas por las que pase, notificando en cualquier momento los

inconvenientes que pudiera percibir del proceso. Todo esto sería posible gracias la recopilación remota de datos, al procesamiento automatizado y a la presentación online de resultados. Cabe remarcar que el contexto en el que se desarrolla este gran proyecto es el de la industria electrónica, en concreto el del testeo de dispositivos para el espacio y otros ambientes hostiles.

Virtual Lab descansa sobre cuatro pilares fundamentales:

- **Flexibilidad y rapidez.**

  La ayuda y experiencia ofrecidas estarán disponibles al instante.

- **Innovación.**

  El cliente podrá acceder a los datos de manera remota dondequiera que esté. Además, se ejecutará un análisis predictivo con el fin de prevenir errores y notificarlos.

- **Seguridad.**

  La seguridad de las redes de comunicación y de los datos estará garantizada de principio a fin.

- **Eficiencia.**

  Se ofrecerá una conectividad total con el fin de poder planificar los resultados de los tests y los equipos disponibles en cualquier momento según los intereses del cliente.

De esta manera, los procesos que tradicionalmente se han elaborado de una manera relativamente opaca para el cliente - no se ofrecían los datos hasta haber finalizado las pruebas requeridas - deberán ser transformados paulatinamente hasta llegar al objetivo de automatización y supervisión descritos.

En el caso concreto de los ensayos climáticos, que pueden llegar a durar días enteros, no detectar fallos durante su curso puede derivar en una pérdida de recursos totalmente innecesaria y evitable. Por ejemplo, un ensayo cíclico de calor húmedo que siga los criterios de la norma UNE-EN 60068-2-30 tendría un ciclo de 24 horas que constaría de tres partes diferenciadas:

1) Elevación de la temperatura al valor superior prescrito por la especificación particular. Dicha temperatura se deberá alcanzar en un tiempo de 3 h ± 30 min.
2) Estabilización de la temperatura alrededor del valor superior dentro del rango de ± 2 K hasta 12 h ± 30 min desde el comienzo del ciclo.
3) Disminución de la temperatura hasta 25 ºC ± 3 K en 3 h ± 15 min.

El tiempo de ejecución del ensayo rondará por tanto las 18 horas. Si la discrepancia en las condiciones ambientales se ha producido al inicio – la temperatura está fuera del rango previsto en la norma – de este ensayo y no se comprueba hasta haber finalizado, la máquina habrá consumido energía inútilmente, provocando un incremento de costes debido al pobre tratamiento de fallos. Así pues, suponiendo que la cámara utilizada tuviese un consumo medio de 1’5 kW - valor del orden de magnitud de las cámaras convencionales - y que el error se produciese a las 3 horas de iniciar el ensayo, se habría malgastado un total de 22’5 kWh.

Con Virtual Lab, al supervisarse automáticamente, dispondrá de tratamiento de errores y activación de alarmas automáticas. En el ejemplo anterior, la monitorización de los datos proporcionados por los sensores habría dado la posibilidad de habilitar una alarma que indicase que las medidas estaban fueran de rango y así abortar el ensayo, lo que habría ahorrado energía y tiempo. De esta reflexión se revela que uno de los objetivos clave de la aplicación desarrollada en este trabajo será el tratamiento de alarmas con el fin de optimizar los procesos que monitorice. Todos estos avisos de error serán notificados al técnico para que disponga de toda la información disponible de lo que sucede en el laboratorio, ya sea por medio de interfaces gráficas como por correo electrónico o mensajes al móvil.

El cliente tendrá acceso a todos los tests que haya solicitado por medio de una aplicación que muestre los resultados en tiempo real del laboratorio. El técnico también contará con una interfaz en la que podrá validar los datos para el cliente o ponerse en contacto mutuo. La figura 3 muestra un prototipo de interfaz propuesto para *Virtual Lab* en un ensayo genérico. Se puede observar que el componente bajo testeo está en la parte superior junto a datos identificativos como pueden ser el fabricante, el nombre del dispositivo, una descripción o el nombre del test.

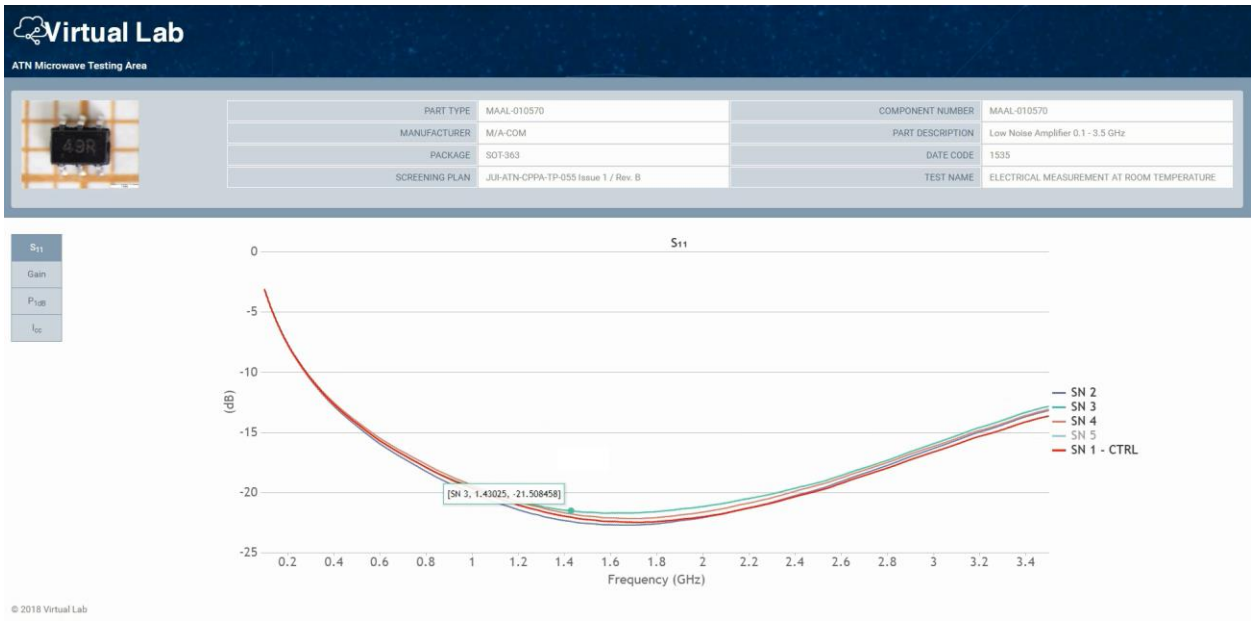

Figura 3: Prototipo de interfaz de Virtual Lab para un ensayo genérico.

Además de gráficas calculadas a partir de medida de sensores, también se podrán consultar imágenes u otro tipo de información que sea de interés para el usuario. Por ejemplo, en el aréa de ATN de microscopia acústica se elaboran ensayos a partir de los cuales se adquieren imágenes de los circuitos de estudio. De esta modo, el usuario podría consultar remotamente todas las imágenes disponibles. La figura 4 muestra el prototipo de la interfaz para las imágenes del microscopio acústico.

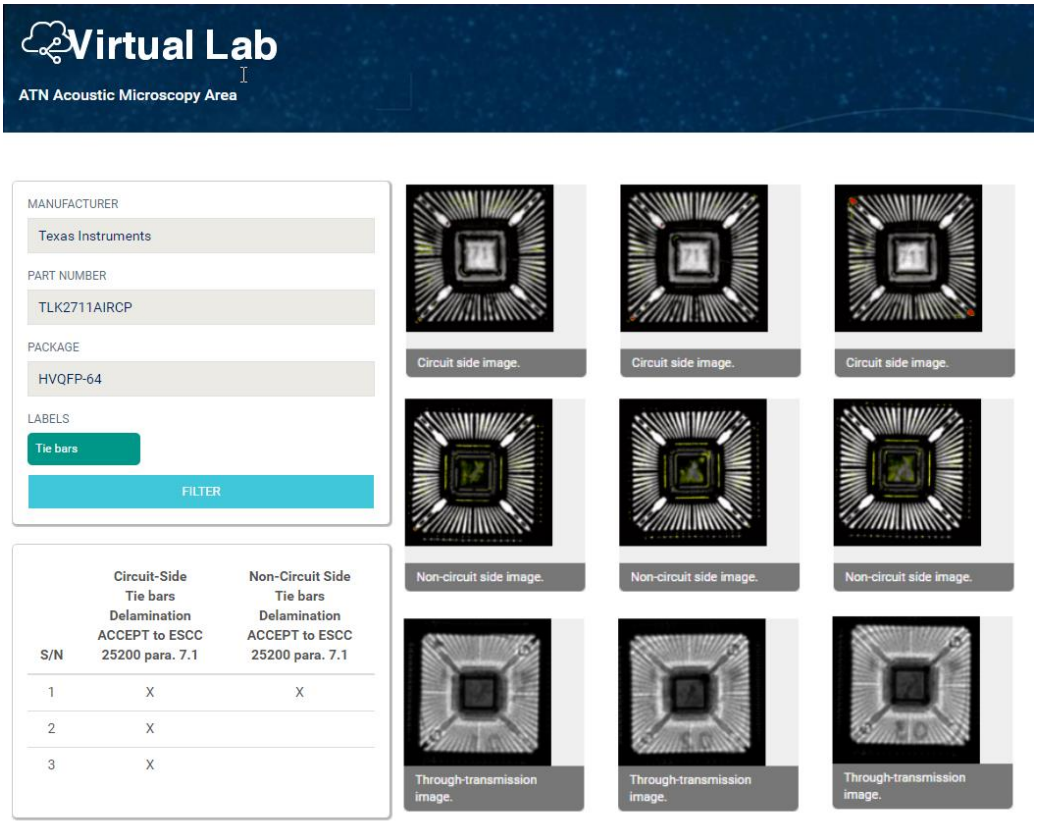

Figura 4: Prototipo de la interfaz de Virtual Lab para ensayos con el microscopio acústico.

Toda la información que se muestre será recopilada a través de la infraestructura informática necesaria para la obtención, procesamiento y presentación de los datos. Esta infraestructura contará con sistemas de adquisición de datos, instrumentos de medida, servidores y múltiples equipos interconectados con el fin de proporcionar un servicio integral desde que el producto es introducido en los laboratorios hasta que termina todo el proceso.

Tal y como se comentó brevemente, el cliente tendrá la posibilidad de abrir una comunicación directa con el técnico para que éste le ofrezca una asistencia inmediata, de calidad y personalizada según sus necesidades.

Todas estas ideas, que se han puesto en marcha en ATN recientemente, marcan la línea a seguir del presente trabajo.

3. MONITORIZACIÓN

El presente capítulo describirá los ensayos que la aplicación desarrollada es capaz de monitorizar, así como las características de los equipos presentes en estos ensayos - cámaras climáticas y sondas de temperatura -. Finalmente se expondrá detalladamente el cálculo de incertidumbre asociado a los instrumentos de medida y las calibraciones de estos últimos.

3.1. Ensayos

La aplicación ha sido concebido con el fin de realizar la monitorización de la temperatura a lo largo del tiempo en ensayos ambientales. La humedad no se ha implementado puesto que, si las condiciones ambientales del laboratorio son estables – en laboratorios controlados lo son –, no debería sufrir grandes fluctuaciones. En futuras ampliaciones del proyecto sí se realizará el análisis de la humedad relativa del sistema, necesario para alguno de los siguientes ensayos.

Por tanto, los ensayos que puede monitorizar sin ningún tipo de modificación son el ensayo cíclico compuesto de temperatura y humedad, el de variación de temperatura, el cíclico de calor húmedo, el de frío y el de calor seco. A continuación se exponen el procedimiento y las características de dichos ensayos.

3.1.1. Ensayo cíclico compuesto de temperatura y humedad

3.1.1.1. Objetivos del ensayo

Este ensayo sigue la norma UNE-EN 60068-2-38, gracias a la cual se han detallado los siguientes apartados. También conocido como ensayo Z/AD, es un ensayo cíclico de temperatura y humedad que tiene como objetivo comprobar las imperfecciones y fallos producidos en los dispositios de estudio debidos al "bombeo", efecto considerado diferente al de absorción de humedad según la norma. Este fenómeno se favorece gracias a los cilos de temperatura con alta humedad relativa de los que está compuesto este ensayo.

3.1.1.2. Características principales

Tiene las siguientes diferencias respecto a los ensayos cíclicos de calor húmedo:

- Mayor cantidad de ciclos de fluctuación de temperatura.
- Valores extremos más elevados para el ciclo térmico.
- Mayor gradiente de temperatura.
- Varias ocasiones en las que la temperatura puede llegar a valores negativos.

Las consecuencias de este ensayo son principalmente el "bombeo" acelerado y la congelación del agua presente en las cavidades y grietas del DUT. Este último fenómeno sólo tiene lugar en el caso de que las dimensiones de las fisuras tengan un mínimo de tamaño.

El grado de la condensación varía con la constante térmica de la superficie del especímen. En el caso de DUT de tamaño pequeño, la condensación se puede despreciar, mientras que si es el dispositivo es grande, ésta puede ser elevada.

3.1.1.3. Descripción general

El especímen será introducido en la cámara climática donde será sometido a 10 ciclos de temperatura y humedad de 24 horas de duración cada uno.

Existe la restricción de que durante cinco ciclos cualesquiera de los nueve primeros, el DUT deberá sufrir condiciones de frío después de la exposición a condiciones de humedad. El resto de ciclos - cuatro en total - no se expondrán al frío.

Sin embargo, antes de someterlo a los ciclos, se completará una fase de preacondicionamiento que consistirá en conservar el dispositivo bajo testeo a 55 ºC ± 2 ºC y una h.r. inferior al 20 % durante las 24 horas previas a los ciclos estándar.

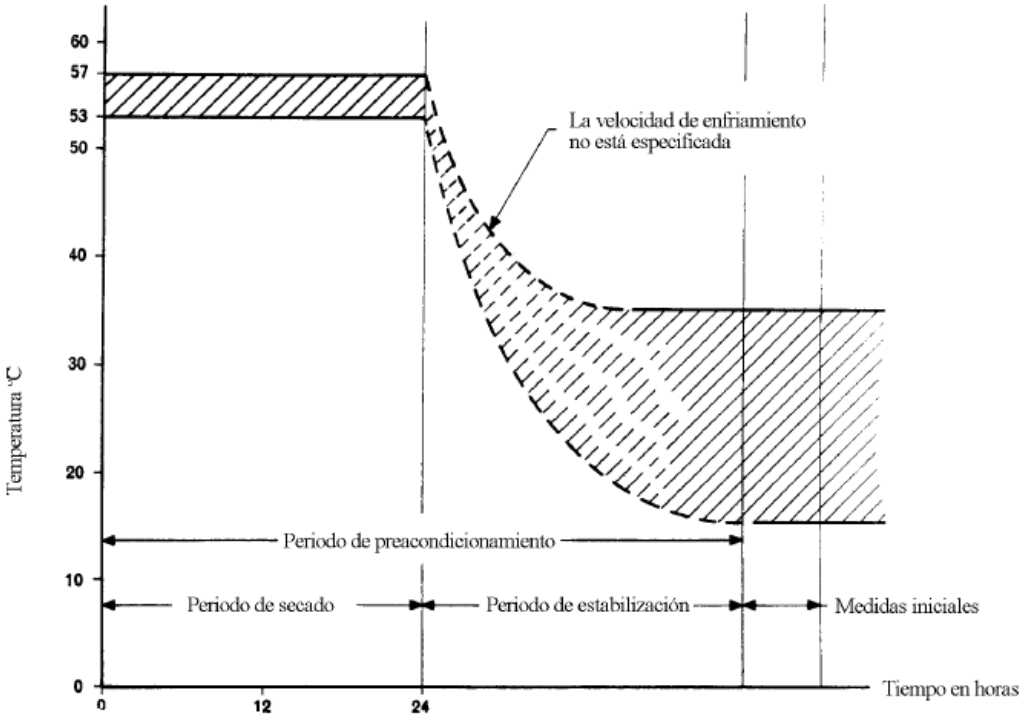

Figura 5: Fase de preacondicionamiento (fuente: UNE-EN 60068-2-38).

En la fase de temperatura y humedad se producirán los siguientes eventos.

1. En el instante inicial de cada ciclo de 24 horas, la cámara deberá estar a 25 ºC ± 2ºC con una humedad relativa de 93 % ± 3 %.
2. Seguidamente, la temperatura aumentará hasta 65 ºC ± 2 ºC en un tiempo comprendido entre 1’5 y 2’5 horas, manteniendo la humedad relativa en los límites anteriores. Deberá permanecer en estas condiciones hasta 5’5 horas después de haber comenzado el ciclo.
3. Posteriormente, se disminuirá la temperatura hasta 25 ºC ± 2 ºC en un tiempo de 1’5 a 2’5 horas. La h.r. en este momento se encontrará entre el 80 % y el 96 %.
4. 8 horas después del inicio, la temperatura deberá elevarse otra vez hasta los 65 ºC ± 2 ºC en un tiempo comprendido entre 1’5 y 2’5 horas. La humedad deberá estar entre 93 % ± 3 %. Permanecerá en este estado hasta las 13’5 horas a partir del comienzo del ciclo.
5. A continuación, se produce un enfriamiento similar al del punto 3.
6. Finalmente, la cámara deberá estabilizarse en torno a 25 ºC ± 2 ºC y 93 % ± 3 % de h.r. hasta el arranque de la fase de frío o el fin del ciclo de 24 horas, según se necesite.

A su vez, la fase de frío se detalla a continuación.

1. Al final de la fase de temperatura y humedad expuesta anteriormente, la cámara se mantiene a 25 ºC ± 2 ºC y 93 % ± 3 % de h.r. durante un tiempo mayor a 1 h e inferior a 2 h.
2. 17,5 horas después del inicio del ciclo de 24 horas, la temperatura deberá disminuir hasta los -10 ºC ± 2 ºC, permaneciendo en este rango durante las 5 horas siguientes.
3. A las 21 horas después del comienzo del ciclo, la temperatura deberá elevarse hasta 25 ºC ± 2 ºC. La h.r. será de 93 % ± 3 %. Estas condiciones se mantendrán hasta finalizar el ciclo de 24 horas.

En el caso de que el ciclo no incluya exposición al frío, el especímen permanecerá a 25 ºC ± 2 ºC y 93 % ± 3 % de h.r. desde el final de la fase de temperatura y humedad hasta el final del ciclo de 24 horas.

El ciclo final – el décimo – contará con una fase de temperatura y humedad para luego establecer las condiciones de 25 ºC ± 2 ºC y 93 % ± 3 % de h.r. durante 3’5 horas. Después de este ciclo, el DUT será retirado de la cámara y esperará en condiciones atmosféricas durante 24 horas antes de efectuar las medidas finales descritas en la norma mencionada.

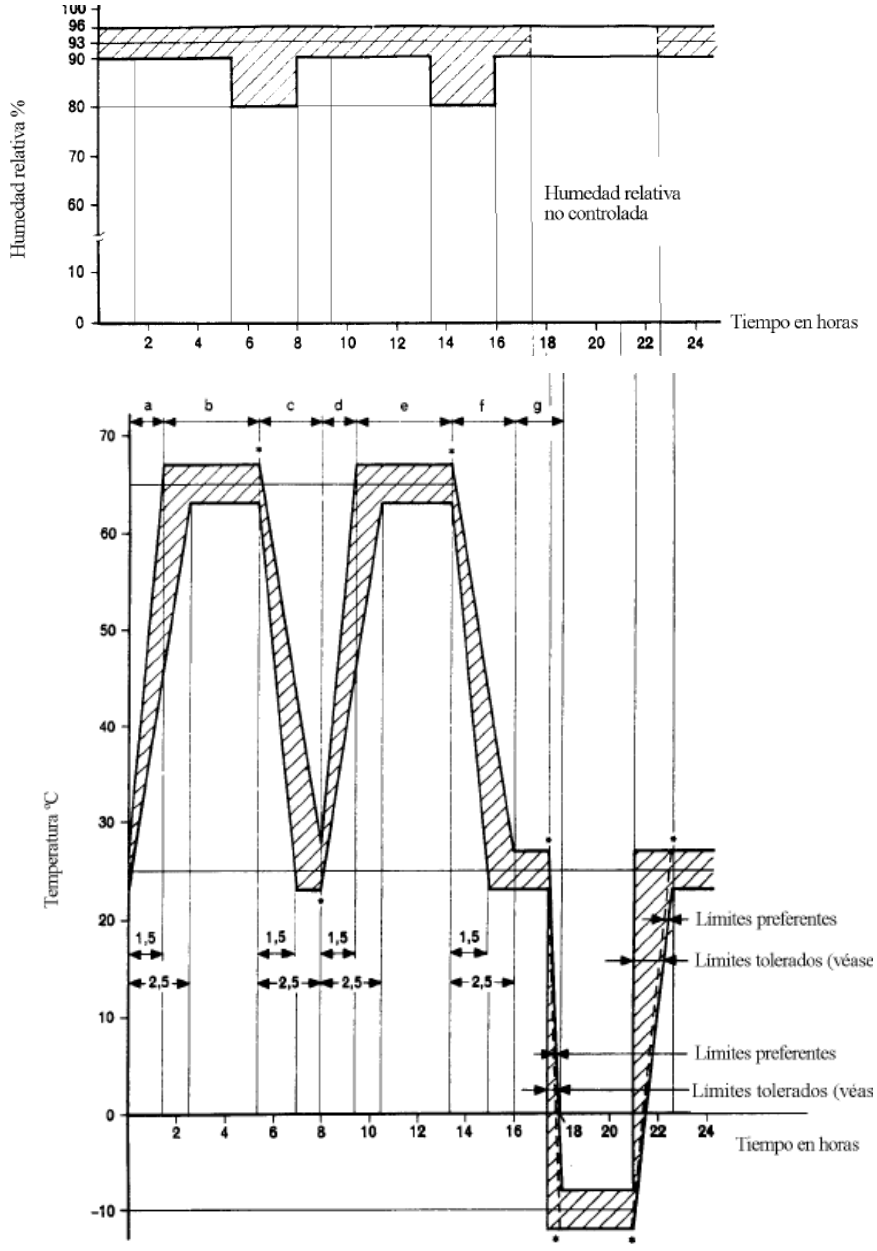

Figura 6: Humedad relativa (arriba) y temperatura (abajo) del ciclo con exposición al frío (fuente: UNE-EN 60068-2-38).

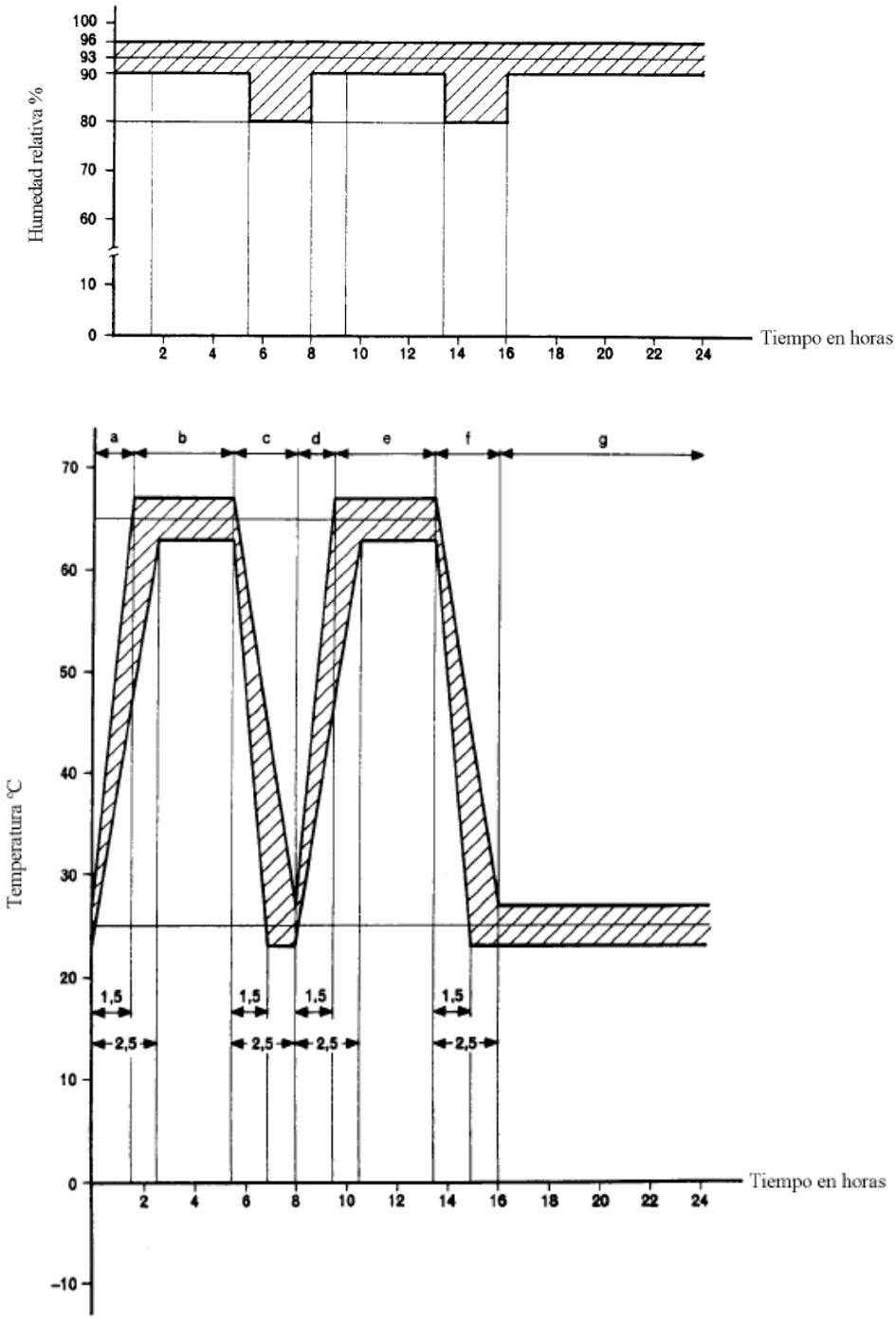

Figura 7: Humedad relativa (arriba) y temperatura (abajo) del ciclo sin exposición al frío (fuente: UNE-EN 60068-2-38).

3.1.2. Ensayo cíclico de calor húmedo

3.1.2.1. Objetivos del ensayo

El ensayo cíclico de calor húmedo sigue la norma UNE-EN 60068-2-30, a partir de la cual se han tomado los datos que se exponen en este apartado. También conocido como ensayo Dd, consiste en uno o más ciclos de temperatura de 24 horas en los que la humedad relativa permanece en un valor elevado.

3.1.2.2. Características principales

El ciclo tiene dos variantes cuya única diferencia es el periodo de disminución de temperatura: la variante 1 dispone de tolerancias más reducidas sobre la h.r. y el gradiente de temperatura.

En cuanto a la severidad del ensayo, ésta viene determinada por la combinación de la temperatura más alta y el número de ciclos. Según la norma, se puede elegir entre los siguientes valores:

a) Temperatura superior: 40 ºC.

   Número de ciclos: 2, 6, 12, 21, 56.

b) Temperatura superior: 55 ºC.

   Número de ciclos: 1, 2, 6.

Los valores de temperatura en dos puntos del espacio de trabajo no deberán diferir entre sí en más de 1 K. Si esto fuera así, las condiciones de humedad requeridas no podrán cumplirse en todo el habitáculo. Además, puede ser imprescindible mantener las fluctuaciones de corta duración dentro de ± 0'5 K con el fin de obtener la humedad deseada.

3.1.2.3. Descripción general

Antes de proceder a realizar el ensayo, es necesario que el especímen esté a una temperatura estable y controlada. De esta manera, es necesario someter al DUT a un periodo inicial de estabilización con el fin de que alcance una temperatura de 25 ºC ± 3 K.

Durante este periodo, la h.r. debe mantenerse dentro de los límites de tolerancia prescritos para las condiciones atmosféricas normalizadas del ensayo.

Después de la estabilización, se debe aumentar la h.r. hasta un valor igual o superior al 95 % con una temperatura de 25 ºC ± 3 K.

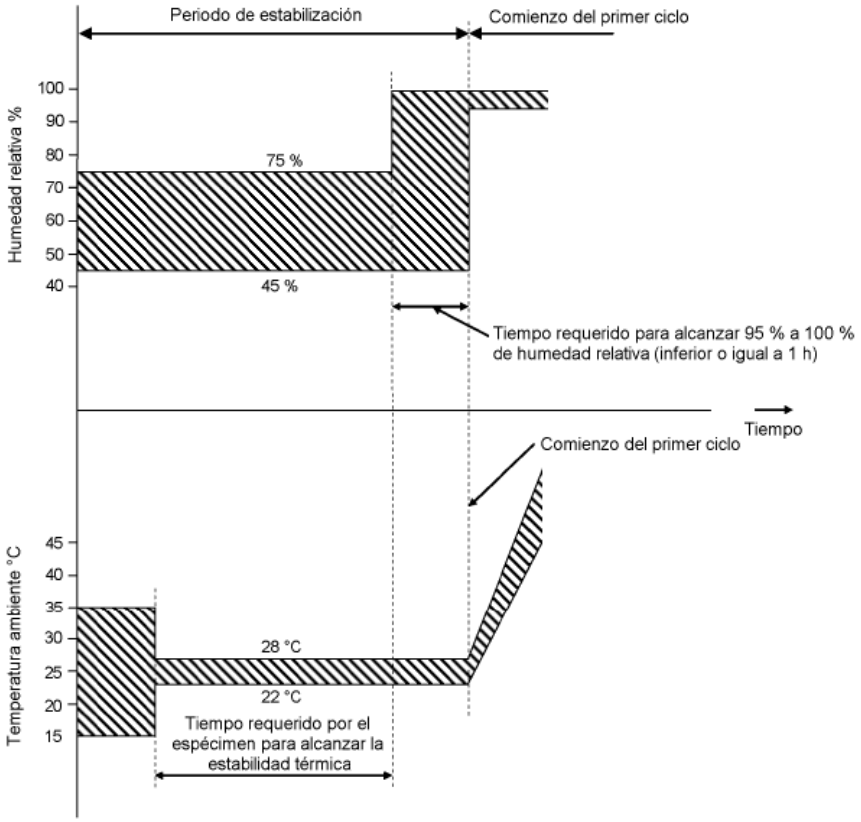

Figura 8: Periodo de estabilización en el ensayo cíclico de calor húmedo (fuente: UNE-EN 60068-2-30).

El ciclo de 24 horas se compone de varias fases. Durante la primera de ellas, la temperatura de la cámara debe elevarse de forma continua hasta el valor superior elegido en la severidad del ensayo en un tiempo de 3 h ± 30 min. La humedad relativa tiene que ser de al menos el 95 %. Durante los últimos 15 min no debe ser inferior del 90 %.

La segunda parte del ciclo consiste en mantener la temperatura dentro de los límites de temperatura superior hasta 12 h ± 30 min desde el comienzo del ciclo. En cuanto a la h.r., ésta debe encontrarse en el rango de 93 % ± 3 %. Durante los primeros y los últimos 15 min, debe estar entre el 90 % y el 100 %.

La tercera fase es la de disminución de temperatura, la cual dispone de dos variantes.

- **Variante 1.**

  Se disminuye la temperatura hasta 25 ºC ± 3 K en un tiempo comprendido entre 3 h y 6 h. El gradiente de temperatura durante la primera hora y media debe ser tal que la temperatura de 25 ºC ± 3 K se alcanza en 3 h ± 15 min. La h.r. debe sobrepasar el 90 % durante los primeros 15 min y el 95 % en el resto del tiempo. La figura 9 muestra el ciclo de 24 horas con la variante 1 de esta fase.

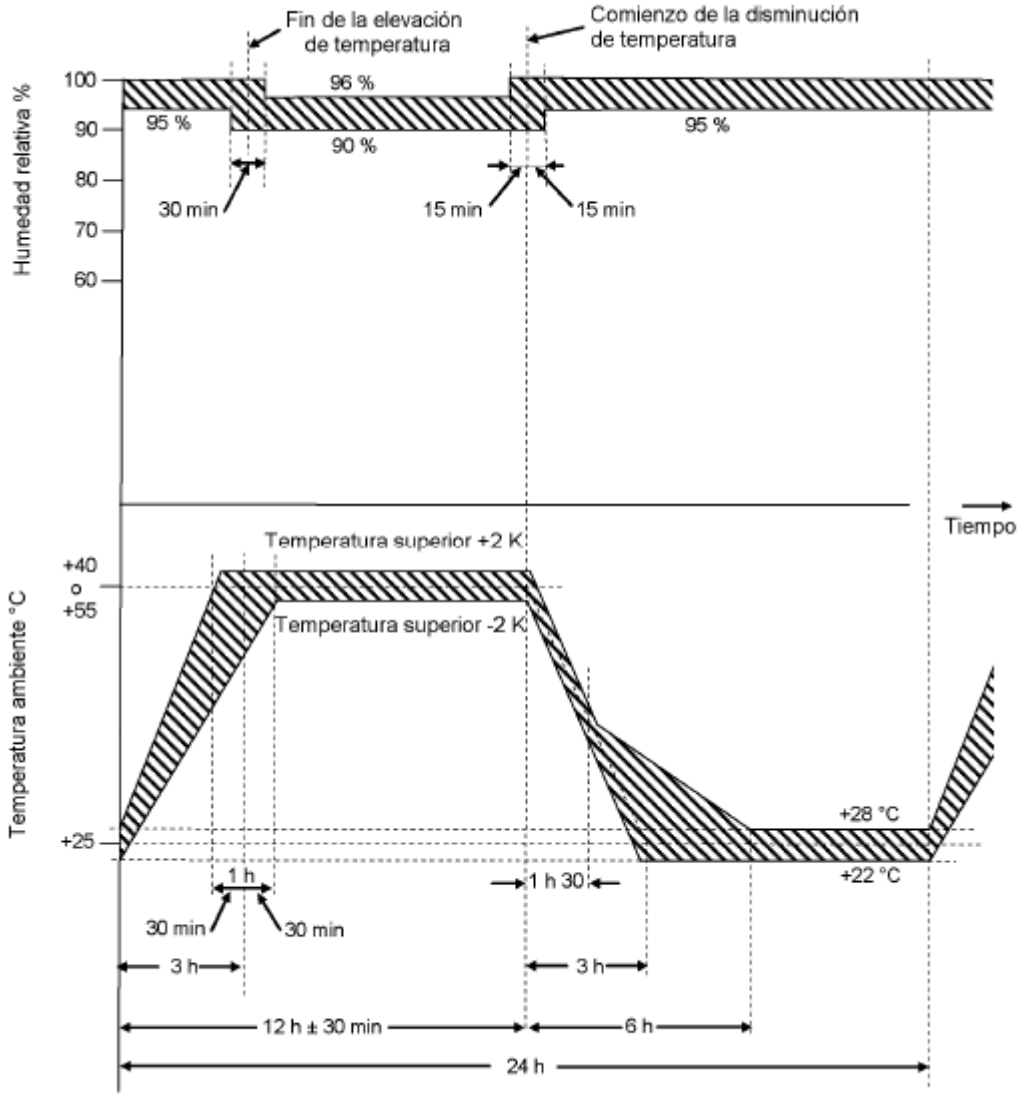

Figura 9: Ciclo con variante 1 (fuente: UNE-EN 60068-2-30).

- **Variante 2.**

  Es similar a la variante 1 pero sin el requisito de la primera hora y media. Además, la humedad relativa no debe ser inferior al 80 %. La figura 10 representa el ciclo de 24 horas con la variante 2 de esta fase.

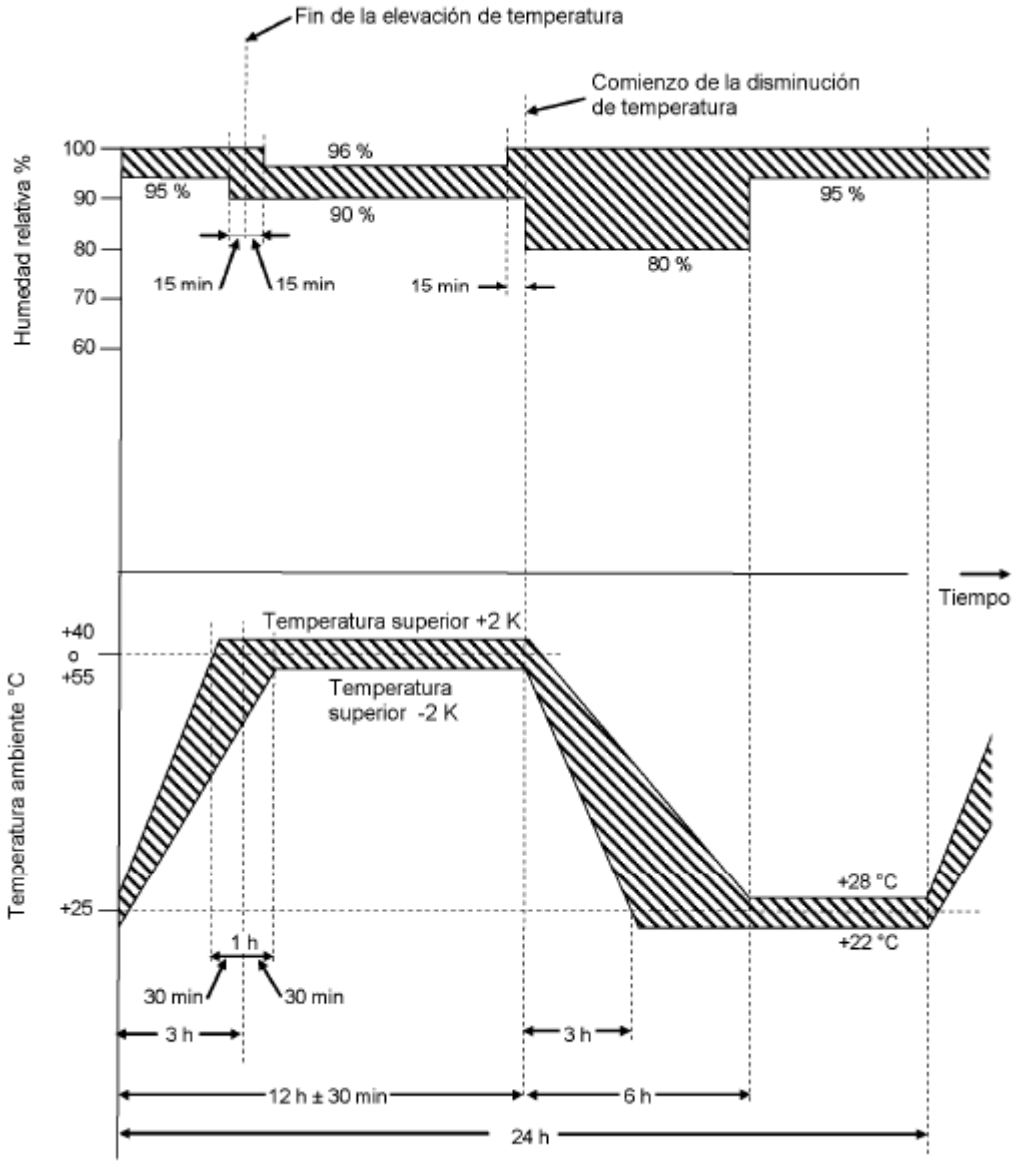

Figura 10: Ciclo con variante 2 (fuente: UNE-EN 60068-2-30).

Finalmente, la última fase del ciclo de 24 horas consiste en permanecer a 25 ºC ± 3 K con una humedad relativa superior al 95 % hasta que dé comienzo el siguiente ciclo.

Una vez terminado el número de ciclos correspondiente, se debe realizar una etapa de recuperación siguiendo las directrices de la Norma IEC 60068-1, ya sea en condiciones atmosféricas normalizadas de ensayo o en condiciones controladas de recuperación. Para ello, la h.r. tiene que reducirse hasta el 75 % ± 2 % en no más de 1 hora. Luego, la temperatura debe ajustarse a un valor igual a la del laboratorio ± 1 K en la hora siguiente. En el caso de que el especimen posea una constante térmica elevada, el periodo de recuperación debe ser lo suficientemente largo para alcanzar la estabilidad térmica, tal y como indica la Norma IEC 60068-1.

3.1.3. Variación de temperatura

3.1.3.1. Objetivos del ensayo

La norma de referencia para este ensayo ha sido UNE-EN 60068-2-14. El ensayo de variación de temperatura o ciclado térmico, cómo también es llamado, tiene como objetivo determinar las aptitudes tanto de los dispositivos electrónicos como de las interconexiones soldadas para resistir temperaturas extremas, sometiendo al DUT a exposiciones cíclicas a dichas temperaturas. De esta manera, como resultado del estrés impuesto, pueden haberse producido cambios físicos o mecánicos que comprometerían la fiabilidad del componente a corto o largo plazo. Los daños que se pueden ocasionar son entre otros:

- Grietas o fracturas debidas a la dilatación térmica.
- Cortocircuitos.
- Circuitos abiertos por desconexión de las pistas.

Las condiciones del test pueden ser modificados de acuerdo a las especificaciones que desee el fabricante, siempre dentro de los límites de las normas que se quieran certificar en el ensayo.

Se suele aplicar a componentes no polarizados en un rango de temperaturas que sobrepasa las condiciones ambientales de funcionamiento. En algunos casos, este rango se extiende a valores que ascienden de las especificaciones máximas con el fin de probar a los componentes en situaciones aún más extremas.

Existen tres variantes del ensayo que son:

- Variación rápida de la temperatura con tiempo prescrito de transferencia. Nomenclatura: Na.
- Variación de la temperatura con una velocidad de variación especificada. Nomenclatura: Nb.
- Variación rápida de la temperatura, método de los dos baños de fluido. Nomenclatura: Nc.

A continuación, se realizará una descripción específica para Na y Nb. Nc no se explicará porque es un ensayo en el que el DUT debe sumergirse en un líquido y en este proyecto no se ha previsto ese tipo de situaciones.

3.1.3.2. Características principales

La cámara climática debe mantener la temperatura en el rango indicado, sirviéndose de sensores que confirmen que cumple la norma. Estos sensores estarán distribuidos a lo largo de todo el interior de la cámara. Además, los DUT se colocan de tal manera que el flujo de aire en el habitáculo no se vea obstruido.

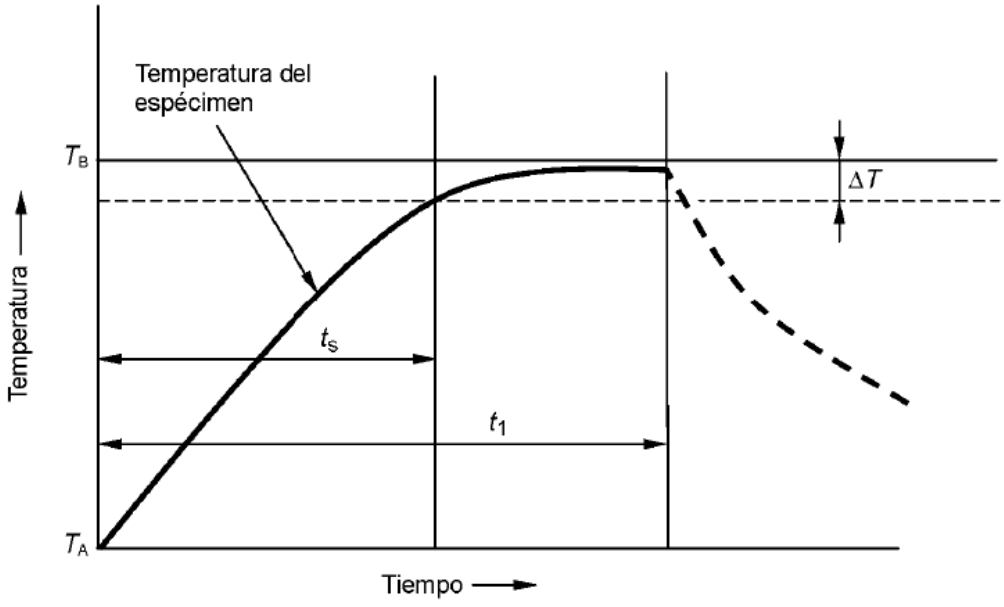

Figura 11: Cálculo del tiempo de exposición del ensayo (fuente: UNE-EN 60068-2-14).

Entre los parámetros de ensayo se encuentran la temperatura ambiente del laboratorio, el valor de la alta y la baja temperaturas, la duración de la exposición o el número de ciclos del ensayo. Un ciclo se considera la variación desde la temperatura ambiente a la primera temperatura de acondicionamiento, luego a la segunda de acondicionamiento y finalmente la vuelta a la ambiente.

La duración de la exposición deberá basarse en los requisitos de cada una de las variaciones del ensayo o como establezca la especificación particular, considerando los siguientes puntos:

- La exposición comienza una vez el DUT esté en el nuevo ambiente.
- Cuando el espécimen se encuentra a una diferencia de temperatura ΔT con respecto al medio de ensayo de entre 3 K y 5 K, se estima que está en estabilización. El periodo de estabilización t_s es el tiempo que pasa entre el inicio y el momento en el que se alcanza la estabilización.
- La duración del ensayo t_1 debe ser superior a t_s. La figura 11 representa gráficamente la determinación del tiempo de duración del ensayo o tiempo de exposición.

Como medidas iniciales y finales se deberá efectuar una inspección visual y una comprobación eléctrica y mecánica, según como requiera la especificación particular.

3.1.3.3. Ensayo Na

El espécimen es sometido a variaciones rápidas de la temperatura del aire. De esta manera, la severidad del ensayo está determinada por la combinación de las dos temperaturas extremas, el tiempo de transferencia, el tiempo de exposición y el número de ciclos.

La temperatura más baja T_A debe elegirse entre las temperaturas indicadas en las normas IEC 60068-2-1 y IEC 60068-2-2. Del mismo modo deberá escogerse el valor de la temperatura más alta T_B.

El tiempo de exposición t_1 dependerá de capacidad calorífica del DUT. Los valores que puede tomar son 10 min, 30 min, 1, 2 o 3 horas. Si no se especifica, se supondrá como 3 h.

Preferentemente, el número de ciclos será 5, a menos que la especificación particular requiera otro valor.

Antes de comenzar los ciclos de ensayo, el espécimen debe estar a la temperatura ambiente del laboratorio, es decir, 25 ºC ± 5 K.

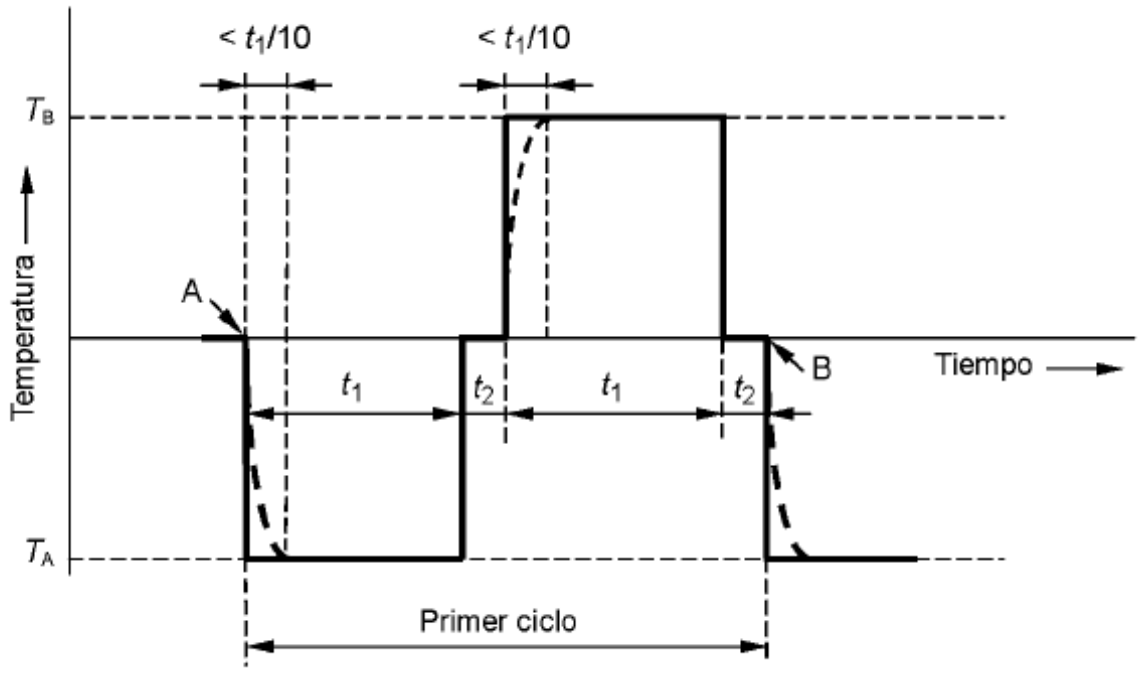

Figura 12: Ciclo del ensayo Na (fuente: UNE-EN 60068-2-14).

Cada ciclo de ensayo tiene el siguiente procedimiento:

1. Se expone el DUT a baja temperatura, T_A.
2. Dicha temperatura se mantiene durante t_1. Esta duración incluye un tiempo inicial, no superior a 0'1 t_1, para la estabilización de la temperatura del aire en la cámara.
3. Se expone el especímen a una temperatura alta, T_B, en un periodo de tiempo t_2 no superior a 3 min.
4. T_B es mantenida durante otro periodo t_1 en las mismas condiciones que el paso 2.
5. Para el siguiente ciclo, se debe exponer el espécimen a T_A en un tiempo de transferencia t_2.

El ciclo se puede observar en la figura 12.

Una vez se termina el último ciclo, se somete al DUT a un proceso de recuperación que consiste en permanecer en condiciones atmosféricas normales de ensayo durante un tiempo que permita al sistema alcanzar la estabilidad de temperatura.

3.1.3.4. Ensayo Nb.

Este ensayo es similar al ensayo Na con la diferencia de que los equipos bajo testeo pueden funcionar durante las variaciones de temperatura.

La temperaturas T_A y T_B son, al igual que en el apartado anterior, temperatura baja y alta respectivamente.

La temperatura del aire debe variarse entre el 90 % y el 10 % de D, donde D es igual a la diferencia entre las temperaturas extremas con una tolerancia del 20 % de la velocidad de variación de la temperatura. Los valores de este gradiente pueden ser, salvo indicación:

- 1 ± 0'2 K/min
- 3 ± 0'6 K/min
- 5 ± 1 K/min
- 10 ± 2 K/min
- 15 ± 3 K/min

El tiempo de exposición t_1 dependerá de la capacidad calorífica del DUT y podrá ser uno de la serie de valores que se indicó en el ensayo Na.

El espécimen se expondrá a dos ciclos consecutivos, a menos que se especifique otro caso.

La fase de acondicionamiento será igual que en Na, manteniendo el DUT a la temperatura ambiente del laboratorio.

El ciclo de ensayo está compuesto por las siguientes fases:

1. La temperatura del aire de la cámara se reduce hasta la temperatura extrema inferior T_A con el gradiente especificado.
2. Una vez la estabilidad térmica se alcanza, el espécimen debe permanecer en esas condiciones durante t_1.
3. Al contrario que en la fase 1, se aumenta la temperatura hasta T_B.
4. Se procede de igual manera que en el paso 2.
5. Finalmente, se disminuye la temperatura hasta el valor de la temperatura ambiente del laboratorio 25ºC ± 5 K.

Después del último ciclo y al igual que en el ensayo Na, el DUT permanece en condiciones atmosféricas normales de ensayo hasta alcanzar la estabilidad térmica.

La figura 13 muestra un ciclo del ensayo Nb.

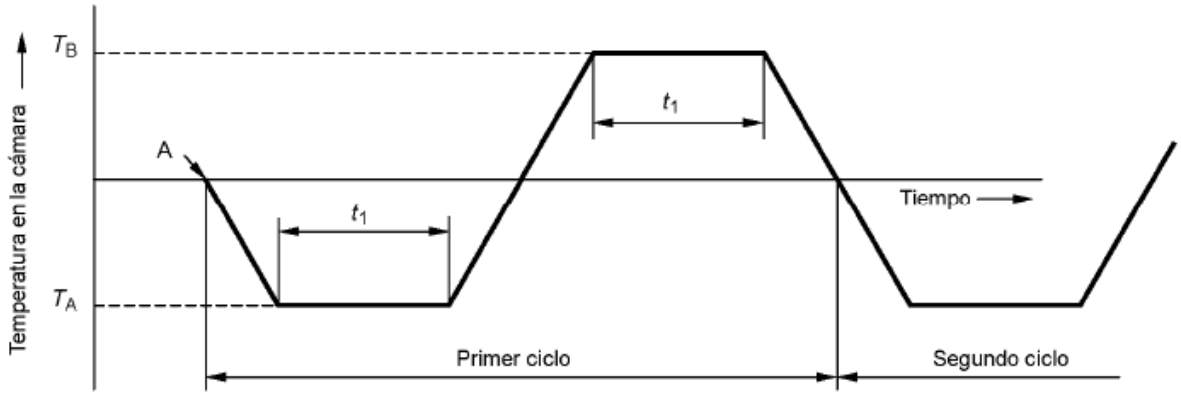

Figura 13: Ciclo del ensayo Nb (fuente: UNE-EN 60068-2-14).

3.1.4. Frío

3.1.4.1. Objeto del ensayo

Para este ensayo es aplicable la documentación de referencia UNE-EN 60068-3-1 y UNE-EN 60068-2-1. A diferencia de otros casos, en el ensayo de frío se tiene en cuenta la disipación de calor por parte del DUT. De esta manera, existen las variaciones siguientes.

- Ensayo de frío para especímenes que no disipan calor con variación lenta de temperatura. Nomenclatura del ensayo: Ab.
- Ensayo de frío para especímenes que disipan calor con variación lenta de temperatura. Nomenclatura del ensayo: Ad.
- Ensayo de frío para especímenes que disipan calor con variación lenta de temperatura, para especímenes alimentados durante todo el periodo de ensayo. Nomenclatura del ensayo: Ae.

Según la norma, un espécimen es disipante de calor sólo si la temperatura del punto más caliente de su superficie es mayor que la temperatura de la atmósfera circundante más 5 K, siempre una vez alcanzada la estabilidad térmica.

3.1.4.2. Características principales

El gradiente de temperatura en el interior de la cámara no debe exceder 1 K/min medido en un intervalo de no más de 5 minutos.

Además, deberá verificarse la velocidad del aire dentro de la cámara de ensayo mediante el procedimiento del apartado 4.2 de la Norma UNE-EN 60068-2-1.

3.1.4.3. Ensayo Ab

Cuando el DUT se introduce en la cámara, ésta debe estar a la temperatura del laboratorio. A continuación, la temperatura se ajusta al valor deseado al que se quiera someter al espécimen. Una vez se alcanza la estabilidad térmica, el DUT se expone a estas condiciones durante el tiempo especificado.

Normalmente, los especímenes están en condiciones de no operación.

La circulación de aire que se suele usar en este ensayo es a alta velocidad.

3.1.4.4. Ensayo Ad

Se procede de igual manera que en el ensayo Ab. Sin embargo, después de alcanzar la estabilidad térmica, el dispositivo bajo estudio se pone en funcionamiento durante un periodo concreto con el fin de verificar su comportamiento.

En este ensayo, la velocidad de circulación del aire que se utiliza es baja.

3.1.4.5. Ensayo Ae

A diferencia de las variantes anteriores, en el ensayo Ae el espécimen debe permanecer operando durante todo el ensayo. Por tanto, después de introducir el DUT en la cámara y antes de alcanzar la estabilidad térmica, se debe realizar un test funcional, es decir, comprobar su correcto funcionamiento. Posteriormente, se someterá al espécimen a baja temperatura la duración que sea especificada.

Por regla general, la circulación del aire se realiza a velocidad baja.

3.1.5. Calor seco

3.1.5.1. Objeto del ensayo

La documentación aplicable es UNE-EN 60068-3-1 y UNE-EN 60068-2-2. Al igual que en el ensayo de frío, se hace distinción entre tres posibilidades dependiendo de la disipación de calor por parte del espécimen. Las variaciones son:

- Ensayos de calor seco para especímenes que no disipan calor con variación lenta de temperatura. Nomenclatura del ensayo: Bb.
- Ensayo de calor seco para especímenes que disipan calor con variación lenta de temperatura. Nomenclatura del ensayo: Bd.
- Ensayo de calor seco para especímenes que disipan calor con variación lenta de temperatura, para especímenes energizados durante todo el periodo de ensayo. Nomenclatura del ensayo: Be.

3.1.5.2. Características principales

Similares al ensayo de frío, la norma especifica un valor máximo del cambio de temperatura dentro de la cámara de 1 K/min medido sobre un periodo de no más de 5 minutos.

Además, deberá verificarse la velocidad del flujo de aire dentro del habitáculo tal y como indican los documentos de referencia.

3.1.5.3. Ensayo Bb

El DUT se introduce en la cámara a la temperatura del laboratorio. Posteriormente, se ajusta la consigna de la cámara al grado de severidad seleccionado de la especificación particular. Una vez alcanzada la estabilidad térmica, el espécimen se somete a dicha temperatura durante el tiempo requerido.

En el caso de que sea necesario verificar el funcionamiento del DUT, se deberá realizar un ensayo funcional.

Las condiciones de los especímenes suelen ser de no operación y la velocidad de circulación del aire es, normalmente, alta.

3.1.5.4. Ensayo Bd

Es similar al ensayo Bb, con la diferencia de que se realiza un ensayo para determinar si la cámara cumple los requisitos de velocidad baja de aire o no, si fuese necesario. El procedimiento es el mismo que en Bb a excepción de que, una vez alcanzada la estabilidad térmica, es obligatorio comprobar el correcto funcionamiento del dispositivo en las condiciones del ensayo.

3.1.5.5. Ensayo Be

Al igual que con el ensayo Bd, se realiza un ensayo para determinar si la cámara cumple los requisitos de velocidad baja del flujo de aire si fuese necesario. La única diferencia es que durante todo el ensayo, el DUT debe permanecer en funcionamiento para controlarlo y asegurarse de que es capaz de funcionar correctamente de acuerdo con la especificación indicada.

Se suele usar una velocidad baja de circulación del aire.

3.2. Equipos utilizados

Los equipos necesarios para verificar que las condiciones de los ensayos descritos son adecuadas son principalmente la cámara climática y las sondas de temperatura. Así pues, a continuación se procede a describir los equipos usados en ATN, en concreto la cámara climática VCL 7006 de Vötsch y las sondas de temperatura PT100.

3.2.1 Cámara climática VCL 7006

El modelo VCL 7006 se corresponda con una cámara climática que puede controlar tanto la humedad como la temperatura. Entre sus características técnicas se encuentran:

- Volumen del interior: 64 l.
- Rango de temperaturas: de -70 a 180 ºC.
- Desviación de temperatura en el tiempo: ± 0’3 a ± 1 K.
- Homogeneidad de la temperatura en el espacio: ± 0’5 a ± 2 K.
- Gradientes máximos de temperatura:
    - Calentamiento: 3’5 K/min.
    - Enfriamiento: 2’5 K/min.
- Rango de humedad: de 10 a 95 % de h.r.
- Desviación de humedad en el tiempo: ± 1 a ± 3 % de h.r.
- Peso: 140 kg.
- Potencia máxima: 2’5 kW.

Figura 14: Cámara climática VCL 7006 de Vötsch.

Otros aspectos importantes de la cámara son:

- Interfaz TCP/IP.
- Ventana transparente para la observación del DUT.
- Iluminación del espacio de testeo.
- Puerto de entrada de 50 mm de diámetro.
- Unidad de refrigeración de aire.
- Calibración con dos temperaturas y dos valores de h.r. con certificados.
- Unidad de desmineralización de agua.
- Interfaz IEEE 488.
- Sensor capacitivo de humedad.

En cuanto a los límites operacionales de humedad y temperatura, se pueden observar en la figura 15 cuando se usa para ensayos climáticos – aquellos que tienen en cuenta tanto la humedad como la temperatura –. La zona 1 muestra el intervalo de funcionamiento en condiciones estándar. En cambio, la zona 2 se corresponde a la extensión de los límites gracias a un secador externo.

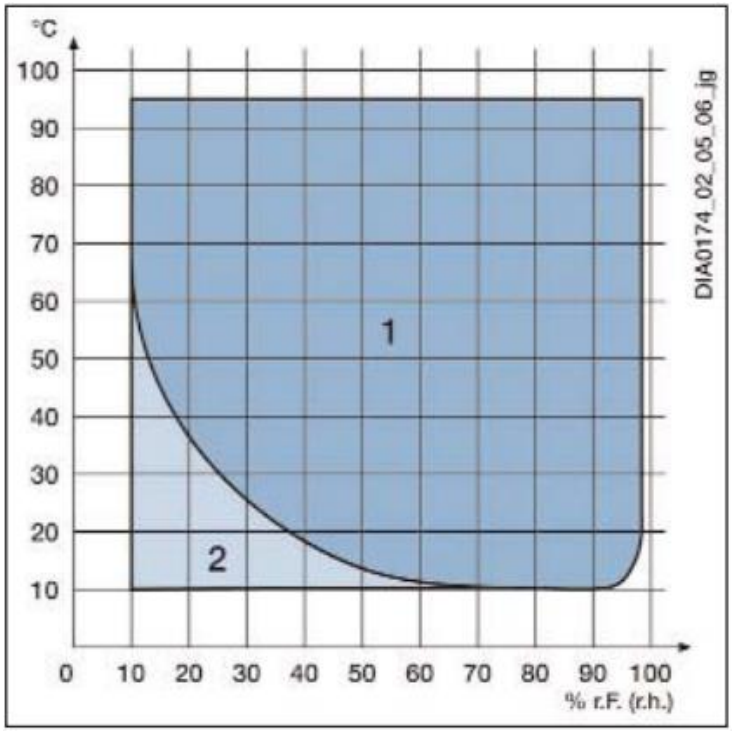

Figura 15: Límites operacionales de temperatura y humedad de la VCL 7006.

En cuanto a la configuración, dispone de un panel táctil o *touchpanel* que tiene como función manejar la cámara. Por tanto, es un dispositivo de mando. El software que se implementa se llama SIMPAC y es común a todas las cámaras de Vötsch, salvando ciertas diferencias entre modelos. El software preparado para computadores convencionales se llama SIMPATI. Gracias a él se pueden configurar y almacenar programas o establecer los valores límites. Dispone incluso de funcionalidades para la representación gráfica de los programas de ensayo. La interfaz se puede observar en la figura 16.

Figura 16: Pantalla de SIMPATI.

3.2.2 Sondas de temperatura PT100

Las sondas PT100 son detectores RTD. Estan fabricadas con platino y tienen una resistencia de 100 Ω a la temperatura de referencia de 0 ºC.

Las sondas que se usarán en los tests son de la familia TLD con referencias internas LE0645.001 hasta LE0645.004. Anualmente son calibradas por el CEM de acuerdo con la norma EIT-90, equivalente a ITS-90. La última calibración efectuada sobre estos instrumentos fue realizada el 8 de septiembre de 2018. En la siguiente subsección se tratará en detalle la calibración tanto de las sondas PT100 como de la cámara climática.

La figura 17 muestra un esquema del interior de la cámara y las posiciones posibles en las que pueden colocarse las sondas. En este caso, las PT100 se distribuirán en las posiciones S1 a S4 de la figura.

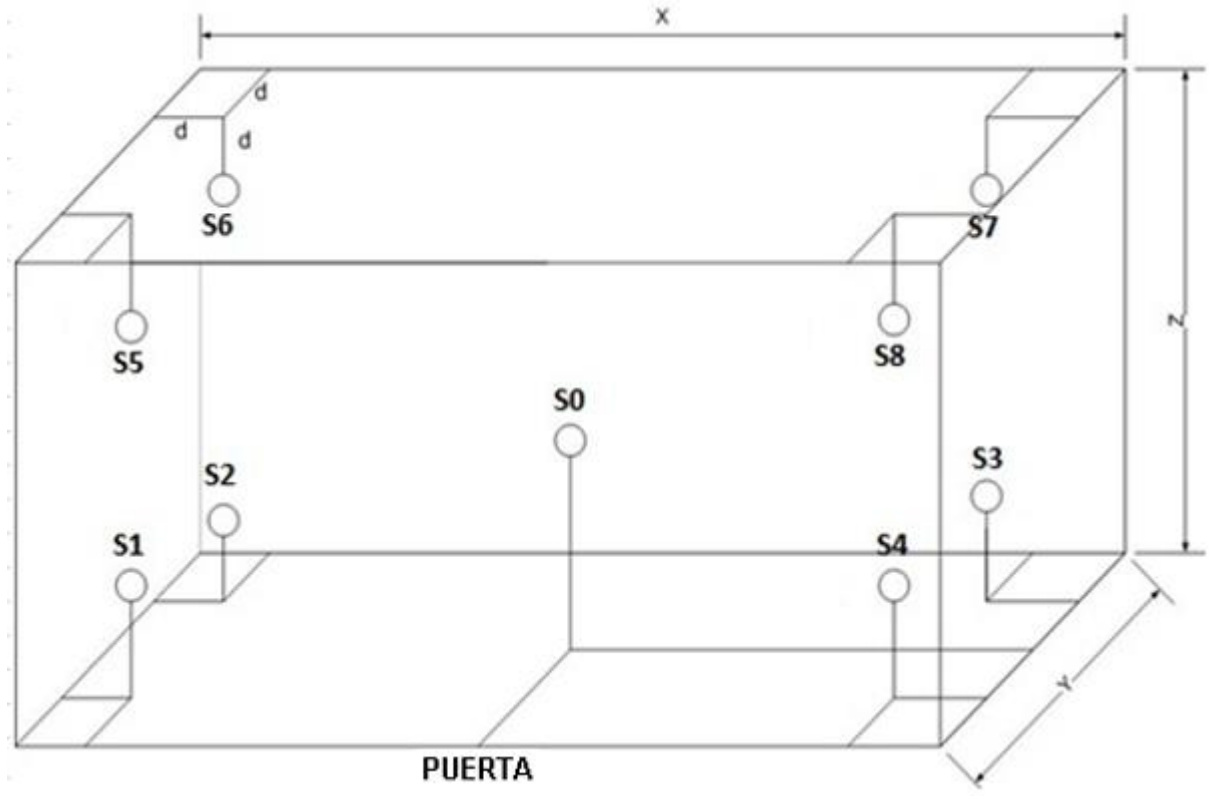

Figura 17: Esquema y distribución espacial de las sondas en la cámara.

Esta distribución tiene como objetivo el análisis del gradiente de temperaturas en el interior de la cámara, comprobando que se cumplen las tolerancias previstas en el ensayo.

3.3. Calibración de los instrumentos

Este epígrafe contiene los estudios de los componentes en cuanto a datos de calibración e incertidumbres. En primer lugar, se establecerán y detallarán las calibraciones de los instrumentos de medida. Estos datos serán necesarios para calcular la incertidumbre expandida y combinada del sistema al completo con el fin de cuantificar el error que se comete y por tanto, determinar si se han cumplido las especificaciones de temperatura del ensayo.

Todos los datos de calibración han sido recopilados a partir de la base de datos de instrumentos de medida de ATN.

Antes de comenzar con los parámetros concretos, es indispensable hacer un repaso a los conceptos básicos sobre calibraciones e incertidumbres. Las definiciones que a continuación se citan se basan en aquellas del documento sobre Vocabulario Internacional de Metrología o VIM, por sus siglas.

- Incertidumbre de medida: según VIM se define como "parámetro no negativo que caracteriza la dispersión de los valores atribuidos a un mensurando, a partir de la información que se utiliza". A su vez, la incertidumbre instrumental es aquella componente de la incertidumbre de medida que procede del instrumento de medida utilizado.
- Estabilidad de un instrumento de medida: es la capacidad del instrumento de conservar invariantes sus características metrológicas a lo largo del tiempo. Según VIM, "la estabilidad puede expresarse por la variación de una propiedad en un intervalo de tiempo determinado".
- Uniformidad: diferencia entre las zonas más caliente o frías que el valor deseado debido al gradiente de temperatura.

3.3.1. Calibración de las sondas

La calibración de las sondas tiene una serie de componentes que se detallan a continuación:

- Temperatura límite inferior: valor inferior del rango de temperatura para el que está calibrada la sonda. Para todos los dispositivos es de -80 ºC.
- Temperatura límite superior: valor superior del rango de temperatura para el que está calibrada la sonda. Para todos los dispositivos es de 125 ºC.
- Resolución: mínima variación detectable por el instrumento. Para todas las PT100 es de 0'01 ºC.
- Linealidad: según la norma IEC 60050, la linealidad se define como la capacidad de un instrumento de medida de proporcionar una indicación con una relación lineal con una magnitud determinada distinta de la magnitud de influencia.
- Incertidumbre en la medida: definido anteriormente en la introducción de esta subsección.
- Deriva: según el CEM, la deriva se define como la variación lenta de una característica metrológica de un instrumento de medida.
- A, B, C: coeficientes del polinomio de mejor ajuste para la corrección de la temperatura medida por las sondas. La ecuación tendría la forma siguiente:

$$T_{corregida} = A + BT + CT^2$$

Ecuación 1: Correción de la temperatura medida por las sondas.

Siendo T la temperatura medida.

En la tabla 2 se pueden consultar los datos de la última calibración de cada una las cuatro sondas usadas en las medidas.

Tabla 2: Valores de calibración de las sondas PT100.

ID del sensor	*Linealidad*	*Incertidumbre*	*Deriva*	*A*	*B*	*C*
LE0645.001	0.14	0.3	0.5	0.1444414	0.99842464	$2.0102 \cdot 10^{-5}$
LE0645.002	0.185	0.3	0.5	0.0378979	0.99780825	$2.1939 \cdot 10^{-5}$
LE0645.003	0.165	0.3	0.5	0.0807074	0.99826139	$2.0166 \cdot 10^{-5}$
LE0645.004	0.14	0.3	0.5	0.0360078	0.99838194	$1.9115 \cdot 10^{-5}$

3.3.2. Calibración de la cámara

Las contribuciones a la incertidumbre de la calibración de la cámara con identificador interno LE0609.000 son las siguientes:

- Límites inferiores y superiores de temperatura: en este caso no existe un único rango de temperatura sino tres. Consúltese la tabla 3 para los valores numéricos.
- Estabilidad de la temperatura.
- Incertidumbre en la estabilidad de la temperatura.
- Uniformidad de la temperatura.
- Incertidumbre en la uniformidad de la temperatura.
- Límite inferior de humedad: al igual que para temperatura, existe límite inferior de humedad en el que está calibrada la cámara. En este caso es igual al 0 % de h.r.
- Límite superior de humedad: el límite superior de h.r. es del 100 %.
- Estabilidad de la humedad.
- Incertidumbre en la estabilidad de la humedad.
- Uniformidad de la humedad.
- Incertidumbre en la uniformidad de la humedad.

Tabla 3: Valores de calibración de la cámara climática LE0609.000

Magnitud	*Límite inferior*	*Límite superior*	*Estabilidad*	*Incertidumbre de la estabilidad*	*Uniformidad*	*Incertidumbre de la uniformidad*
Rango 1 de T	- 40 ºC	24 ºC	0.16	0.2	1.28	0.32
Rango 2 de T	25 ºC	65 ºC	0.08	0.18	1.08	0.27
Rango 3 de T	66 ºC	125 ºC	0.3	0.26	2.35	0.47
Rango de h.r.[1]	0 %	100 %	1.3	1.5	3.3	2.8

[1] Aunque este proyecto sólo trata la medida de temperatura, se ha incluido la calibración de h.r. para concretar la magnitud que puede llegar a tener.

3.4. Cálculo de incertidumbre

En este epígrafe tratará sobre el estudio de las incertidumbres en la medida, teniendo como objetivo la cuatificación de los posibles errores cometidos en la lectura de los sensores de temperatura.

En primer lugar se describirán brevemente los documentos de referencia que se han consultado para llevar a cabo el estudio de la incertidumbre. Posteriormente, se desarrollarán las diferentes ecuaciones necesarias para el cálculo. A continuación, se expondrán las diferentes contribuciones a la incertidumbre que se han considerado para los ensayos. Finalmente, se mostrará el caso práctico de análisis para un ensayo de frío real proporcionado por ATN.

3.4.1. Documentos de referencia para el cálculo de incertidumbres

Los documentos de referencia para está parte del proyecto han sido la guía GUM, la guía VIM, el documento EA-4/02:2013 y la norma UNE-EN 60068-3-11.

La guía GUM o *Guide to the expression of uncertainty in measurement* es un escrito que sirve como referencia para el cálculo de incertidumbres.

La guía VIM o Vocabulario Internacional de Metrología, según la edición en español, es un documento que tiene como objeto ofrecer un diccionario sobre metrología y, por tanto, incertidumbres.

El documento EA-4/02:2013 es una publicación de la Cooperación Europea para la Acreditación. El título es "Evaluación de la incertidumbre de medida en las calibraciones". El propósito de este documento es armonizar la evaluación de la incertidumbre de medida de la ENAC así como sus requisitos generales y especificos sobre la expresión de la incertidumbre.

La norma UNE-EN 60068-3-11 tiene la finalidad de servir de guía en el cálculo de la incertidumbre de las condiciones en cámaras de ensayos climáticos. Contiene tanto definiciones como procedimientos referentes a las incertidumbres de medida.

3.4.2. Obtención de la incertidumbre expandida

De acuerdo con la guía GUM y el documento EA-4/02 M: 2013, la incertidumbre total del sistema o incertidumbre combinada se obtiene a partir de la ecuación 2, considerando la evaluación tipo B de la incertidumbre estándar. Esta ecuación supone que las magnitudes no están correlacionadas.

$$u_c^2(y) = \sum_{i=1}^{N} u_i^2(y)$$

Ecuación 2: Incertidumbre combinada.

Siendo $u_i(y)$, con i de 1 hasta N, la contribución a la incertidumbre típica asociada a la estimación de salida y resultante de la incertidumbre típica asociada a la estimación de entrada x_i, como se muestra a continuación.

$$u_i(y) = c_i \cdot u(x_i)$$

Ecuación 3: Incertidumbre típica asociada a la entrada x_i.

Con c_i como el coeficiente de sensibilidad asociado a la estimación de entrada x_i y $u(x_i)$ la incertidumbre típica calculada a partir de la incertidumbre expandida proporcionada por los certificados de calibración de los patrones. La ecuación 4 muestra esta última relación.

$$u(x_i) = \frac{U(x_i)}{k}$$

Ecuación 4: Incertidumbre típica calculada a partir del certificado de calibración.

Siendo k el factor de cobertura de la distribución que sigue la entrada y $U(x_i)$ la incertidumbre expandida dada en el certificado de calibración de los instrumentos o del patrón.

Según VIM, el factor de cobertura es "el número mayor que uno por el que se multiplica una incertidumbre típica combinada para obtener una incertidumbre expandida". Este factor de cobertura *k* tiene el objetivo de establecer un intervalo correspondiente a un nivel de confianza mayor que si *k* fuese la unidad. Por ejemplo, para disponer de un nivel de confianza del 95'45 % en una distribución normal, el factor de cobertura deberá ser 2. En el caso de que se requiera un nivel de confianza del 99%, *k* será igual a 2'576.

Una vez hallada la incertidumbre combinada de todas las contribuciones, el siguiente paso consistirá en determinar la incertidumbre expandida total del sistema. Para ello, se deberá tener introducir el concepto de grado de libertad y la distribución t-Student.

Tal y como se explica en el apartado G.3. de GUM, para obtener una mejor aproximación que simplemente usando el factor de cobertura *k* de la distribución normal es necesario usar la distribución de la variable $\frac{(y-Y)}{u_c(y)}$, porque, en la práctica, está normalmente disponible *y* a partir de la ecuación 5.

$$y = \sum_{i=1}^{N} c_i x_i$$

Ecuación 5: Estimador de Y.

Siendo x_i el estimador de X_i.

De esta manera, si *z* es una variable aleatoria normal con media poblacional μ_z y desviación estándar σ, y $\bar{z}$ la media aritmética de *n* observaciones independientes z_k de *z* con $s(\bar{z})$ la desviación estándar de $\bar{z}$, entonces la distribución de la variable $\frac{(\bar{z}-\mu_z)}{s(\bar{z})}$ es una t-Student con ν grados de libertad.

$$\nu = n - 1$$

Ecuación 6: Grados de libertad de una t-Student.

En el caso de evaluaciones de tipo B, el número de grados de libertad se puede afirmar que tiende a infinito. De esta manera, los grados de libertad de todas las contribuciones que se expondrán en el siguiente apartado se supondrán con un valor de 1.000.000. Aunque se volverá a recordar más adelante, el número de grados de libertad asociado a la incertidumbre de la desviación de las lecturas será igual al número de medidas menos uno.

Así pues, los grados de libertad tendrán como objetivo calcular la incertidumbre expandida gracias a la distribución t-Student. Sin embargo, ésta no describe la distribución de la variable $\frac{(y-Y)}{u_c(y)}$ si $u_c^2(y)$ es suma de dos o más componentes. La distribución de esa variable puede ser aproximada por una t-Student con un número efectivo de grados de libertad ν_{eff} obtenido a partir de la fórmula de Welch-Satterthwaite.

$$\frac{u_c^4(y)}{\nu_{eff}} = \sum_{i=1}^{N} \frac{u_i^4(y)}{\nu_i}$$

Ecuación 7: Fórmula de Welch-Satterthwaite.

Se despeja fácilmente ν_{eff} a partir de la ecuación 7, teniendo presente la condición siguiente:

$$\nu_{eff} \leq \sum_{i=1}^{N} \nu_i$$

Ecuación 8: Condición de la fórmula de Welch-Satterthwaite.

El último valor que es indispensable calcular es el factor de cobertura total de la distribución t-Student de dos colas de todas las aportaciones de incertidumbre. Para ello, es necesario calcular el valor de la distribución – normalmente se representa con *t* – para los grados de libertad obtenidos gracias a la fórmula de Welch-Satterthwaite y para la probabilidad que se requiera, en este caso del 95 %.

Todos estos cálculos han sido realizados en código en la aplicación.

3.4.3. Fuentes de incertidumbre

A continuación se explicarán las distintas contribuciones a la incertidumbre que se han considerado en la aplicación para la supervisión de los ensayos. Para ello habrá que tener presente los datos de calibración de los instrumentos expuestos anteriormente en el apartado 3.2. de este trabajo.

3.4.3.1. Calibración de las sondas de temperatura

La incertidumbre de calibración de las sondas es la establecida en el certificado de calibración. Se puede estimar que sigue una distribución gaussiana porque es la suma de varios componentes en la cadena de calibración. El nivel de confianza para el cual está determinada es del 95 %, por tanto, el factor de cobertura o divisor necesario para calcular la incertidumbre estándar será 2. El valor que se toma para calcular la incertidumbre del ensayo es el mayor de todas las sondas – aunque en el caso de las sondas previstas en este proyecto, todas esas incertidumbres son iguales a 0'3.

3.4.3.2. Desviación de las lecturas

Es el máximo de la desviación típica muestral de las lecturas de las sondas una vez habiéndoles aplicado la corrección definida por la ecuación 1. La desviación típica muestral o *s* se calcula a partir de todas las lecturas de los instrumentos – véase la ecuación 9 –.

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

Ecuación 9: Desviación típica muestral de las lecturas.

Siendo x_i el valor de la muestra *i*, $\bar{x}$ el promedio de las lecturas y *n* el número de lecturas. Esta desviación típica se calcula para todas las sondas y se toma la máxima.

El divisor que se le aplica es la raíz cuadrada del número de lecturas menos 1.

3.4.3.3. Deriva de las sondas de temperatura

Segun VIM, la deriva se define como "la variación continua o incremental de una indicación a lo largo del tiempo". En el caso de las sondas implicadas en este proyecto, la deriva está determinada en los valores de calibración del instrumento. Estos valores se pueden consultar en la tabla 2.

Normalmente se usan las distintas calibraciones para estimar los límites de este parámetro. Sin embargo, es posible usar los datos del fabricante aunque éste los proporciona en condiciones ideales de utilización. Por este motivo es una buena práctica utilizar un valor mayor.

El divisor que se considera en la deriva es la raíz cuadrada de doce.

3.4.3.4. Uniformidad del ensayo

La uniformidad del ensayo es la máxima diferencia entre la media de temperatura captada por los intrumentos individualmente y la media de la cámara. La media de la cámara se supone como el promedio de la media de las lecturas de los instrumentos en el tramo a evaluar. De esta manera, la incertidumbre en la uniformidad del ensayo se podría escribir como:

$$u_{uniformidad} = \text{máx}\,(\frac{\sum_{i=1}^{N}\bar{x_i}}{N} - \bar{x}_t)$$

Ecuación 10: Incertidumbre en la uniformidad del ensayo.

Para *N* igual al número de sondas, $\overline{x_i}$ la media de las lecturas de la sonda *i* en el tramo estudiado y $\overline{x_t}$ la media de las lecturas de la sonda *t*, para *t* igual a cada una de las sondas.

3.4.3.5. Incertidumbre en la estabilidad de la cámara

Es la incertidumbre estándar que aparece en la tabla 3, los valores de calibración de la cámara. El valor dependerá de la temperatura a la que se encuentre el habitáculo puesto que, en el caso que concierne a este proyecto, la cámara está calibrada para tres rangos de temperatura diferenciados.

3.4.3.6. Incertidumbre en la uniformidad de la cámara

De igual forma que en la incertidumbre en la estabilidad de la cámara, la incertidumbre en la uniformidad de la cámara está recogida en la tabla 3. Su valor dependerá, a su vez, del rango de temperatura a la que se encuentre el habitáculo.

3.4.3.7. Efecto de la carga

En el caso de considerarse especímenes disipantes de calor, se deberá tener en cuenta el efecto que produce en la incertidumbre del sistema. Este efecto se calcula como la suma de los valores absolutos de la diferencia entre la uniformidad de la cámara y la incertidumbre en la uniformidad de la cámara, y la desviación típica de las lecturas y la estabilidad de la cámara. Conceptualmente, la ecuación 11 refleja esta relación.

$$u_{carga} = \left|Uniformidad - u_{uniformidad}\right| - \left|s_{lecturas} - Estabilidad\right|$$

Ecuación 11: Incertidumbre por el efecto de la carga.

3.4.3.8. Efecto de radiación

El efecto de radiación, a su vez, se puede suponer que tiene un valor del 10 % del efecto de la carga anteriormente calculado.

En algunas cámaras, estos efectos pueden ser bastante importantes y requerirán de un estudio propio. Lecturas inesperadamente grandes o que varían mucho de un ensayo a otro pueden ser un indicio de que existe un problema de radiación. En ese caso, deberán realizarse ensayos suplementarios.

3.4.4. Ejemplo práctico: ensayo de frío

Para poner en práctica todos los cálculos y razonamientos que se han explicado en el anterior apartado, se han utilizado datos de un ensayo de frío real del 21 de agosto de 2018.

Tanto la cámara como las sondas son las descritas en este documento, por tanto, se ha realizado el cálculo de incertidumbre con esos valores de calibración.

La figura 18 muestra la gráfica del ensayo de frío donde cada una de las posiciones corresponde a las medidas recogidas por las sondas de temperatura. Como se puede observar, la temperatura máxima de consiga fue de 25 ºC y la mínima de -5 ºC. Así pues, las incertidumbre fueron calculadas en el tramo de temperatura mínima, y se recogen en la tabla 4.

En ATN, el procedimiento a seguir para el cálculo de estos datos ha sido realizado hasta ahora gracias a la herramienta de hoja de cálculo de Microsoft, Excel. De esta manera, el técnico debía copiar los datos del ensayo en una macro que realizase los cálculos. Como era de esperar, la plantilla de Excel debía ser verificada exhaustivamente para evitar errores que pudiesen llevar a una evaluación equivocada de las incertidumbres. Con este proyecto se ha buscado evitar estas tareas para que se hagan automáticamente gracias a la aplicación.

Tabla 4: Cálculo de incertidumbre para un ensayo de frío real.

COMPONENTE	*Incert. Típica*	*Unidad*	*Divisor*	*Coef. Sens.*	$u_i(t)$	v_i	u_i^4/v_i
Calibración de las sondas	0,300	°C	2,000	1,000	0,150	1000000	$5{,}06\cdot10^{-10}$
Desviación de las lecturas	0,062	°C	30,166	1,000	0,002	909	$1{,}91\cdot10^{-14}$
Deriva de las sondas	0,500	°C	3,464	1,000	0,144	1000000	$4{,}34\cdot10^{-10}$
Uniformidad del ensayo	0,164	°C	1,732	1,000	0,094	1000000	$7{,}96\cdot10^{-11}$
Incertidumbre en la estabilidad de la cámara	0,200	°C	2,000	1,000	0,100	1000000	$1{,}00\cdot10^{-10}$
Incertidumbre en la uniformidad de la cámara	1,280	°C	2,000	1,000	0,640	1000000	$1{,}68\cdot10^{-7}$
Efecto de la carga	1,378	°C	3,464	1,000	0,398	1000000	$2{,}51\cdot10^{-8}$
Efecto de radiacíon	0,138	°C	3,464	1,000	0,040	1000000	$2{,}51\cdot10^{-12}$
Incertidumbre combinada					0,748	2057802	
Incertidumbre expandida					1,50	2057802	
		k_{total}	2,01				

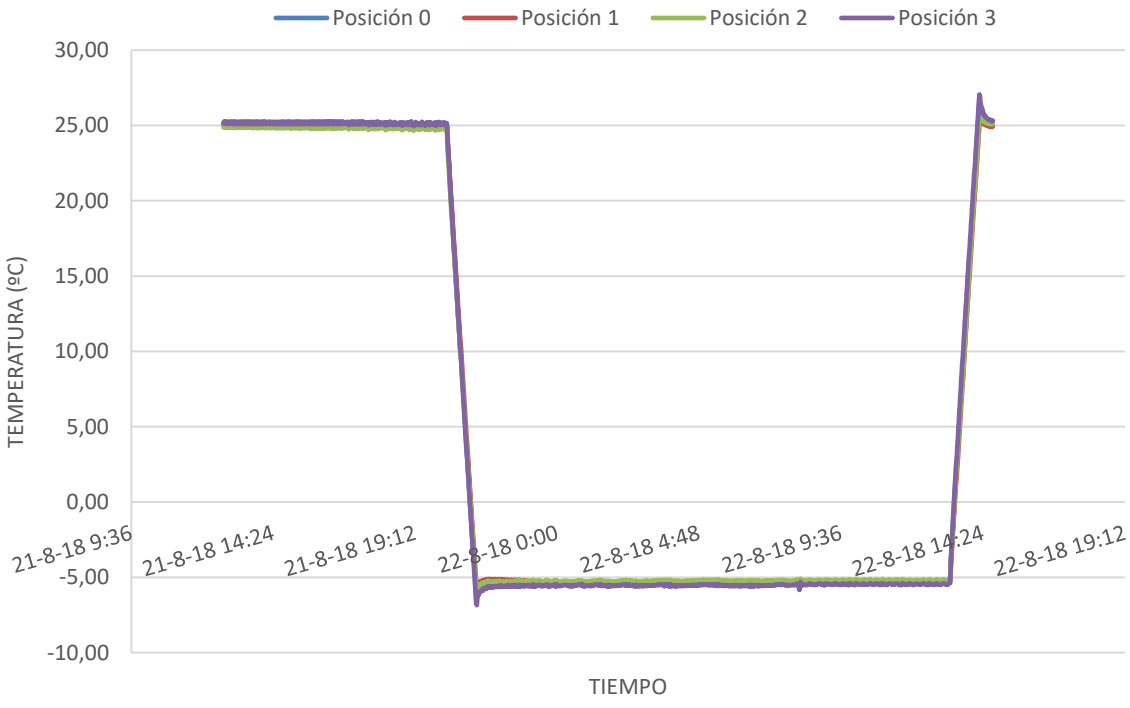

Figura 18: Curva ensayo de frío real.

4. HERRAMIENTAS Y ARQUITECTURA SOFTWARE

Antes de comenzar con la aplicación es indispensable comprender el contexto de las herramientas software que se han usado, en este caso Visual Studio 2019 y Microsoft SQL Server Management Studio 2017. Para ello, se expondrán brevemente las diferencias entre los distintos tipos de bases de datos y el motivo de haber utilizado SQL y no MongoDB, el otro candidato. A continuación, se detallará la arquitectura desarrollada. Finalmente, se expondrán las características del lenguaje C# así como de Windows Forms y .NET. Por último, se concluirá con una descripción exhaustiva de los objetivos de la aplicación desarrollada.

4.1. Base de datos

4.1.1. Conceptos sobre bases de datos

Una base de datos es en esencia una colección de datos propios de un mismo contexto. Esta información debe estar ordenada sistemáticamente para poder recuperarla, analizarla o transmitirla. En cambio, los sistemas gestores de bases de datos tienen por objetivo ofrecer una manera de acceder a los datos almacenados.

Las bases de datos siguen un modelo de datos. Este modelo trata de proponer herramientas conceptuales para especificar la forma de los datos, sus relaciones, la semántica y las restricciones de consistencia. Estos modelos pueden ser clasificados en las siguientes categorías, entre otras:

- **Modelo relacional.**

  Este modelo utiliza un conjunto de tablas para describir tanto los datos como sus relaciones. Las tablas están compuestas por una columna con un nombre único. Además, este modelo se basa en registros de formato fijo de varios tipos. De esta manera, cada tabla posee registros de un cierto tipo. Estos registros están definidos por un número fijo de campos o atributos. Estos atributos conciernen a las columnas de cada tabla.

  El modelo relacional es actualmente el más usado, aunque poco a poco algunos sistemas están siendo migrados a otros modelos más adecuados. El modelo de la base de datos de la aplicación es un modelo relacional basado en lenguaje SQL.

- **Modelo entidad-relación.**

  El modelo entidad-relación o E-R se basa en partir de una situación real en la cual se define entidades y relaciones entre ellas. Estas entidades deben ser distinguibles del resto. Este modelo es ampliamente usado en diseño de bases de datos.

  Las entidades son objetos que no dependen de otros, ni siquiera de los objetos de ese mismo tipo. Están representadas por sus características, también llamados atributos. Por ejemplo, la entidad *Coche* puede tener los siguientes atributos: marca, color, matrícula, modelo, etc.

- **Modelo orientado a objetos.**

  El modelo de datos orientado a objetos agrupa tanto los datos como sus relaciones en estructuras

únicas llamadas objetos. Este modelo se puede entender como una ampliación del modelo E-R junto a ideas como encapsulación o métodos.

- **Modelo de datos semiestructurados.**

  En este modelo, los elementos de datos que sean del mismo tipo pueden tener diferentes grupos de características, a diferencia del resto de modelos explicados anteriormente. Los datos semiestructurados se suelen describir mediante lenguaje de marcas extensible o XML.

4.1.1.1. SQL

El lenguaje SQL ofrece un lenguaje de consultas para el álgebra relacional que se aplica sobre la base de datos. Además de hacer consultas, también es posible utilizarlo para describir estructuras de datos, modificarlos o crear restricciones de seguridad en el sistema.

Entre los distintos componentes de este lenguaje se puede encontrar:

- Lenguaje de definición de datos.
- Lenguaje de manipulación de datos.
- Integridad.
- Definición de vistas.
- Control de transacciones.
- SQL incorporado y dinámico.
- Autorización.

El presente documento no ha de servir como un manual sobre bases de datos, para obtener más información consúltense las referencias relacionadas.

Para la aplicación se ha elegido este tipo de base de datos porque, a pequeña escala, tiene una implementación rápida y eficaz. La creación de tablas y su consulta se puede realizar de manera sencilla y segura tanto a partir de Microsoft SSMS como de Visual Studio. De esta manera, gracias a la infraestructura software de C# proporcionada por ATN, se ha podido implementar el contexto de la base de datos – sus tablas y diagramas – en la aplicación de monitorización desarrollada. Más adelante se detallará su estructura al completo.

4.1.1.2. MongoDB

MongoDB es una base de datos distribuida orientada a documentos de propósito general. A diferencia de las bases de datos relacionales SQL donde la información se guarda en registros, MongoDB almacena los datos en documentos BSON, la representación binaria de JSON. Es la base de datos más popular de las llamadas NoSQL, o *Not only SQL*.

Tiene la particularidad de que los documentos de una misma colección no tienen por qué seguir la misma estructura. Se trata de un concepto similar al de los modelos de datos semiestructurados.

El software dispone de una consola a través de la cual es posible ejecutar los comandos. El lenguaje utilizado para realizar las consultas es JavaScript puesto que la consola está construida con él. Así, muchas de las funciones de JavaScript están disponibles a través de MongoDB. Existe la posibilidad de usar otros lenguajes de programación siempre y cuando los drivers de dichos lenguajes sean instalados previamente. Tienen soporte C#, Java, Node.js, PHP, Python...

Entre las ventajas que aporta MongoDB se encuentran las conocidas como *3V*:

- **Velocidad.**

  En el caso de tener que acceder a una gran cantidad de información en un tiempo mínimo, las bases de datos documentales tienen la capacidad de ser mucho más rápidas que las relacionales por lo general.

- **Volumen.**

  Las bases de datos relacionales funcionan más lentamente en el caso de que las tablas tengan una gran

cantidad de registros. Para solucionar este problema, las tablas se suelen segmentar. Sin embargo, en MongoDB, no existe un problema significativo al tratar con grandes volúmenes de datos.

- **Variabilidad.**

  En las bases de datos relacionales, el diseño está previsto de antemano. De esta manera, el esquema es muy rígido porque no es posible – o muy inconveniente – modificar los campos de los registros de manera dinámica. Esta característica hace más difícil la escalabilidad de la base de datos. En MongoDB, al tratarse de documentos, su estructura puede cambiarse a lo largo del tiempo sin demasiados problemas operativos.

Estas ventajas ya son suficientes para plantearse realizar todo el sistema de gestión de datos a partir de MongoDB. De hecho, en el proyecto *Virtual Lab* de ATN, gran parte de las bases de datos están apoyadas esta arquitectura. Sin embargo, tal y como se expuso anteriormente, debido a la sencillez y al alcance de este proyecto se ha decidido usar una base de datos relacional SQL, totalmente apta para los objetivos de este trabajo.

4.1.2. Microsoft SQL Server Management Studio 2017

Microsoft SSMS es una aplicación software que tiene como función configurar, mantener y administrar todos los componentes de Microsoft SQL Server. La característica principal de SSMS es el llamado *Object Explorer*, que permite buscar, seleccionar y ejecutar acciones sobre cualquiera de los objetos contenidos en el servidor. Desde esta ventana se pueden acceder a las tablas, los diagramas y el resto de componentes del servidor al que está conectado. Esta herramienta se muestra en la figura 19.

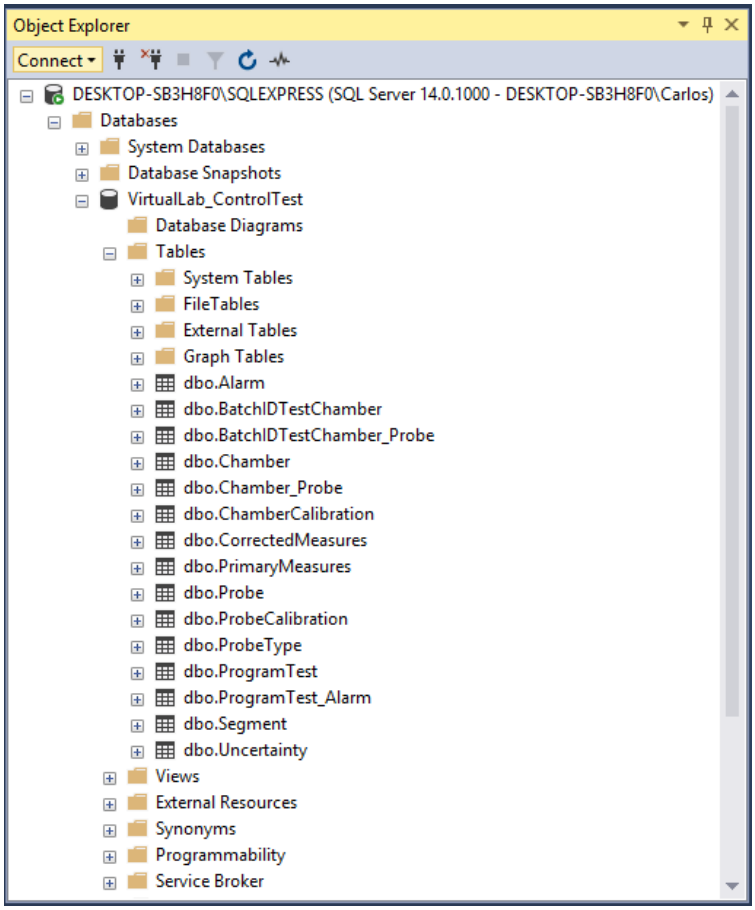

Figura 19: Ventana Object Explorer de Microsoft SSMS.

SSMS ofrece una interfaz muy amigable para el usuario, desde donde hacer consultas, ejecutarlas o crear procedimientos. En la figura 20 se puede observar una consulta de ejemplo a partir de la pestaña *New Query*.

En dicha consulta se crea una tabla de nombre *Clientes* que tiene los campos *IDCliente, Nombre, Apellidos, Direccion* y *Ciudad.*

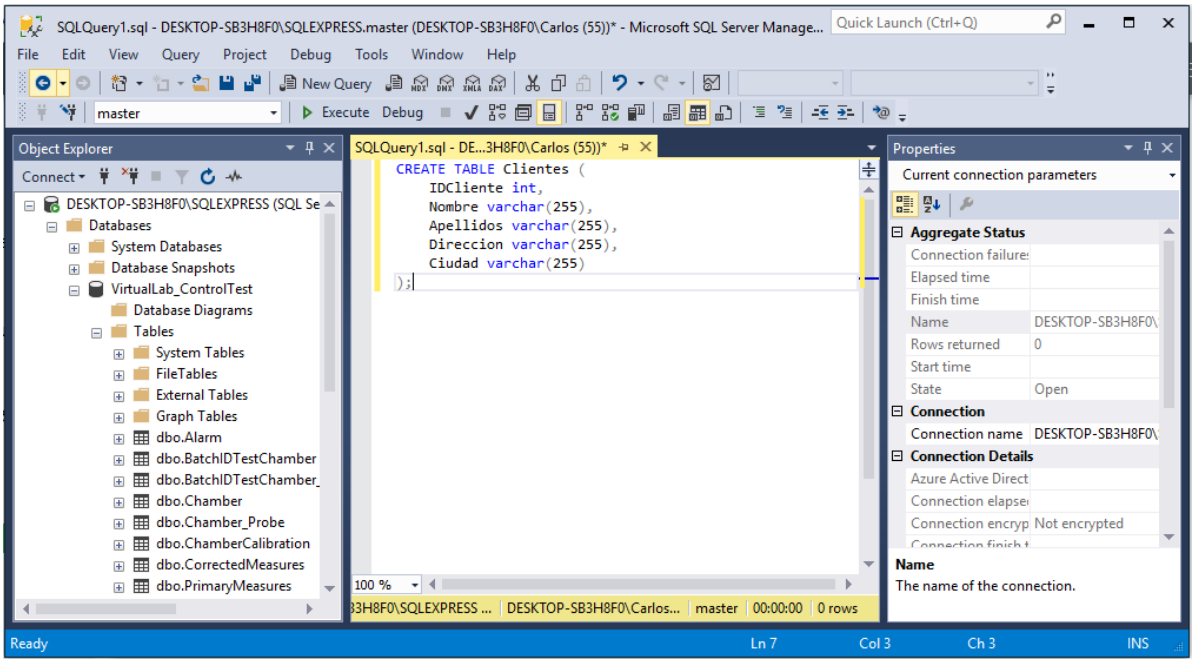

Figura 20: Consulta de ejemplo en Microsoft SSMS.

Además, es posible crear tablas y añadir registros sin necesidad de escribir código SQL si no se desea. En la figura 21 se observa la pestaña de edición de la tabla *PrimaryMeasures*. En ella se pueden modificar los registros que se requieran. Esta tabla está formada por su clave principal *IdPrimaryMeasures*, y el resto de atributos como son *IdTest, ProbeCode, Date, MeasureIndex* y *Measure.* Más adelante se explicará en detalle el diseño completo de la base de datos. También es posible cambiar estos campos en la pestaña de diseño de la tabla – véase la figura 22 –.

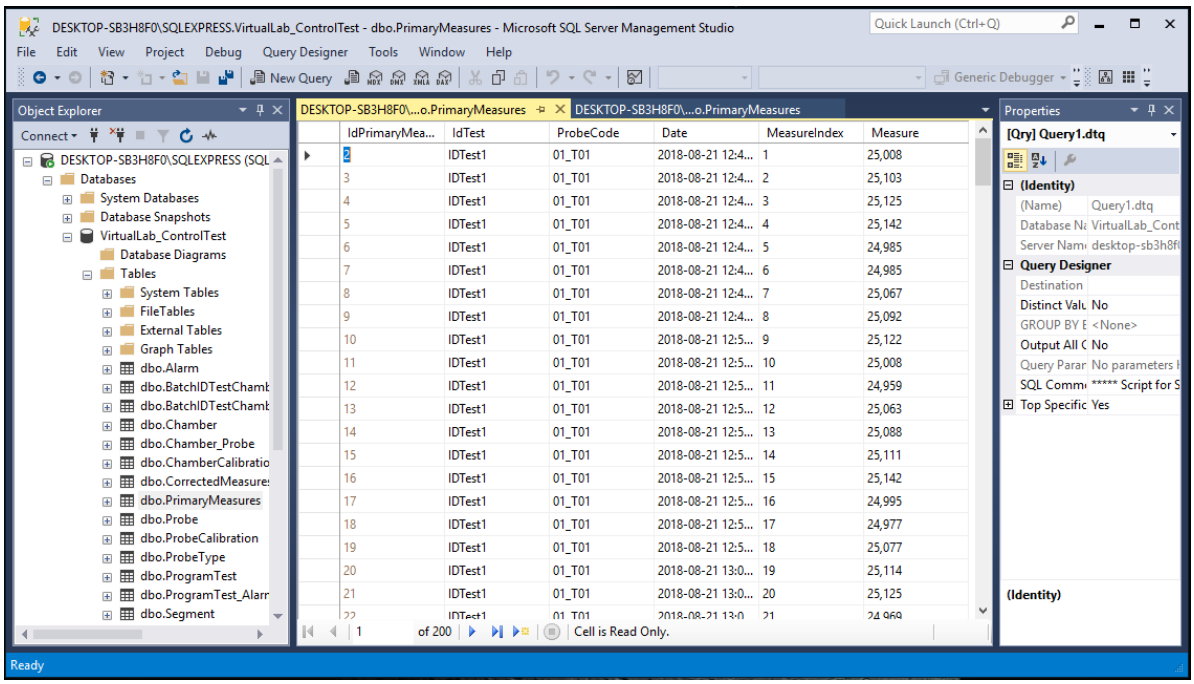

Figura 21: Pestaña de modificación de los registros de la tabla PrimaryMeasures.

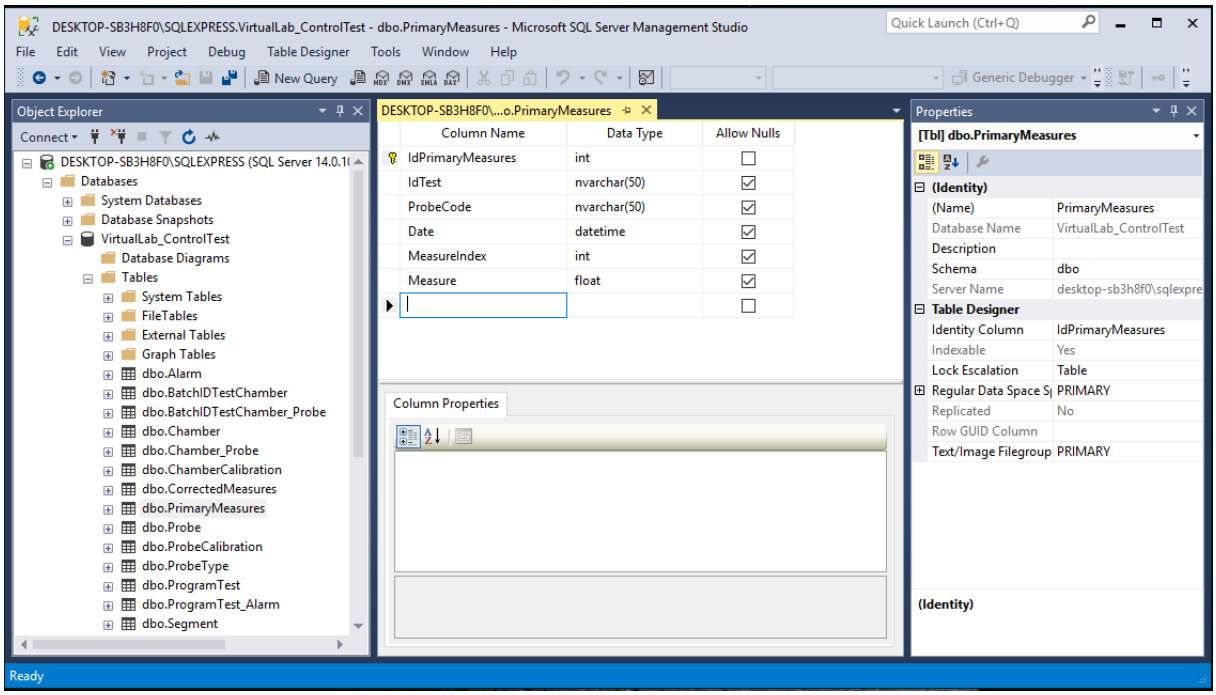

Figura 22: Pestaña de diseño de los campos de la tabla PrimaryMeasures.

Se puede concluir que este software es una herramienta muy sencilla y eficaz para el diseño y mantenimiento de bases de datos, en especial considerando el reducido tamaño de la colección de datos de la aplicación que concierne a este proyecto. Por estos motivos se ha decidido emplear Microsoft SSMS.

4.1.3. Diseño de la base de datos

La base de datos de la aplicación tiene como objetivo reproducir aquella empleada por ATN para almacenar los datos de sus equipos e instrumentos. Además, se han creado las tablas necesarias para poder alcanzar todas las especificaciones que requería la aplicación, por ejemplo, una tabla de alarmas para guardar qué monitorización habría que llevar a cabo, una de tipos de ensayos, etc.

Así pues, a continuación se detallarán las tablas existentes en la base de datos así como su contenido en el caso de que lo tengan. Finalmente, se mostrará un diagrama con las relaciones entre las tablas y sus respectivas claves primarias.

4.1.3.1. Tabla Alarm

La tabla *Alarm* está formada por tres campos:

- *IdAlarm:* clave primaria de la tabla. No tiene utilidad más allá la de distinguir cada registro unívocamente. Todas las tablas poseen un atributo de estas características.
- *Code:* este campo contiene el código de la alarma que se quiere activar. Serán códigos definidos por los informes de ATN.
- *Description:* descripción verbal de la función de la alarma.

Tabla 5: Tabla Alarm.

IdAlarm	*Code*	*Description*
1	A01	Temperatura fuera de rango
2	A02	Humedad fuera de rango

En el caso de este proyecto, los datos que contiene se muestran en la tabla 5. Los códigos *A01* y *A02* indican que es la alarma número 1 y la alarma número 2. Más adelante, en futuros proyectos este código será necesario para una rápida identificación de la alarma, cuando no haya dos sino multitud de tipos de ellas.

4.1.3.2. Tabla BatchIDTestChamber

La tabla *BatchIDTestChamber* relaciona el lote, el ID del test y la cámara con la que se realizó ese ensayo.[2] Por tanto, los campos son:

- *IdBatchIDTestChamber*: clave primaria de esta tabla.
- *Batch*: lote del ensayo.
- *IdTest*: ID del ensayo.
- *IdChamber*: clave primaria de la tabla *Chamber* para relacionar el ID del test y el lote con la cámara.

La tabla 6 muestra los registros existentes en la esta tabla de base de datos.

Tabla 6: Tabla BatchIDTestChamber.

IdBatchIDTestChamber	*Batch*	*IdTest*	*IdChamber*
2	Batch2	IDTest2	2
4	Batch1	IDTest1	3

4.1.3.3. Tabla BatchIDTestChamber_Probe

La tabla *BatchIDTestChamber_Probe* es una tabla que relaciona las claves primarias de la tabla anterior y de la tabla *Probe*. Las tablas con el nombre con el formato *nombre1_nombre2* son de estas características. Tienen la función de relacionar tablas distintas.

Tabla 7: Tabla BatchIDTestChamber_Probe.

IdBatchIDTestChamber_Probe	*IdBatchIDTestChamber*	*IdProbe*
2	2	5
4	2	6
7	4	2
8	4	3
9	4	5
10	4	6

4.1.3.4. Tabla Chamber

Esta tabla guarda registros que identifican a las cámaras y aportan datos cualitativos. Los campos que tiene son:

- *IdChamber*: clave primaria de la tabla.

[2] Todos los ensayos de ATN se pueden identificar a partir de su número de lote – *batch* – y su ID del test, a partir de los cuales se puede obtener automáticamente la cámara – *chamber* – donde se realizó dicho ensayo.

- *Code*: código de la tabla. Están determinados por la nomenclatura propia de ATN.
- *Family*: familia de la cámara. También las define ATN.
- *Description*: descripción cualitativa de la cámara.

Los registros utilizados en la tabla *Chamber* para este proyecto se observan en la tabla 8.

Tabla 8: Tabla Chamber.

IdChamber	*Code*	*Family*	*Description*
1	04_T01	Chamber A (LE1397.000)	Isothermal enviroment
2	04_T02	Chamber B (LE0457.000)	Isothermal enviroment
3	04_T05	Chamber E (LE0609.000)	Isothermal enviroment

4.1.3.5. Tabla Chamber_Probe

Al igual que la tabla *BatchIDTestChamber_Probe, Chamber_Probe* relaciona las claves primarias de *Chamber* y de *Probe* porque, por defecto, las cámaras tienen unas sondas – *probes* en inglés – asociadas. La tabla 9 contiene los registros almacenados en *Chamber_Probe*.

Tabla 9: Tabla Chamber_Probe.

IdChamberProbe	*IdChamber*	*IdProbe*
1	1	2
2	1	3
3	2	5
4	2	6
7	3	2
8	3	3
9	3	5
10	3	6

4.1.3.6. Tabla ChamberCalibration

La tabla *ChamberCalibration* almacena los valores de los parámetros de calibración de las cámaras disponibles – véase la tabla 3 –. Los atributos que la forman son:

- *IdChamberCalibration*: clave primaria de esta tabla.
- *IdChamber*: clave primaria de la tabla *Chamber*.

- *Date*: fecha de la última calibración.
- *NumberOfCertificate*: código del certificado de calibración.
- *TempLow1, TempLow2, TempLow3, TempHigh1, TempHigh2, TempHigh3*: temperaturas límite de los distintos rangos.
- *Stability1, Stability2, Stability3, StabGradient1, StabGradient2, StabGradient3*: estabilidades e incertidumbres en las estabilidades de cada uno de los rangos.
- *Uniformity1, Uniformity2, Uniformity3, UnifGradient1, UnifGradient2, UnifGradient3*: uniformidades e incertidumbres en las uniformidades de cada uno de los rangos.

Debido al tamaño de esta tabla, se ha decidido no incluir en este apartado. Para los valores numéricos, consúltese el apartado 3.2.2. Calibración de la cámara.

4.1.3.7. Tabla CorrectedMeasures

CorrectedMeasures es una tabla que almacena las medidas corregidas a partir del polinomio de mejor ajuste. Está relacionada con la tabla *PrimaryMeasures* pues ésta es la que guarda las medidas primarias o iniciales de los sensores. Tiene los siguientes campos:

- *IdCorrectedMeasures*: clave primaria de esta tabla.
- *IdPrimaryMeasures*: clave primaria de la tabla *PrimaryMeasures*.
- *IdTest*: ID del ensayo en el que se tomó las medición.
- *ProbeCode*: código de la sonda que recogió la medida.
- *Date*: hora y fecha de la medida.
- *MeasureIndex*: índice de la medida. Tiene la función de identificar el orden del conjunto de medidas de un ensayo en concreto.
- *CorrectedTemp*: temperatura corregida si la sonda es de temperatura.
- *CorrectedHum*: humedad corregida si la sonda es de humedad relativa.

La aplicación desarrollada irá introduciendo registros en esta tabla conforme proceda el análisis del ensayo. Por tanto, en el instante inicial, *CorrectedMeasures* está completamente vacía.

4.1.3.8. Tabla PrimaryMeasures

PrimaryMeasures es una tabla que guarda las muestras que van recogiendo los sensores para que luego la aplicación analice y transforme esos datos. Estos registros de las medidas primarias nunca serán modificados para que, en caso de conflicto, se puedan recuperar los datos iniciales tomados desde los instrumentos.

La tabla contiene los siguientes atributos:

- *IdPrimaryMeasures*: clave primaria de esta tabla.
- *IdTest*: ID del ensayo.
- *ProbeCode*: código de la sonda que tomó la medida.
- *Date*: hora y fecha de la muestra.
- *MeasureIndex*: índice de la medida. Deberá ser el mismo que en la tabla *CorrectedMeasures.*
- *Measure*: valor numérico de la medición.

La manera de almacenar los datos será opaca a la aplicación de este proyecto, es decir, no entra dentro del alcance previsto. Esa función la deberá realizar otra herramienta que se conecte al dispositivo de adquisición de datos de la cámara climática.

4.1.3.9. Tabla Probe

La tabla *Probe* es similar a *Chamber* pero en lugar de a las cámaras, hacen referencia a las sondas. Los campos que define a cada sonda son:

- *IdProbe*: clave primaria de la tabla.
- *IdProbeType*: clave primaria de la tabla *ProbeType*.
- *Code*: código de la sonda. Son definidos por ATN.
- *Family*: similar a la familia de las cámaras. La nomenclatura que siguen las define ATN.
- *Description*: descripción cualitativa del instrumento.

El contenido de esta tabla que se ha utilizado se muestra a continuación en la tabla 10.

Tabla 10: Tabla Probe.

IdProbe	*IdProbeType*	*Code*	*Family*	*Description*
2	1	01_T01	TLD (LE0645.001)	Thermometer L.D.
3	1	01_T02	TLD (LE0645.002)	Thermometer L.D.
5	1	01_T03	TLD (LE0645.003)	Thermometer L.D.
6	1	01_T04	TLD (LE0645.004)	Thermometer L.D.

4.1.3.10. Tabla ProbeCalibration

Esta tabla contiene los valores de calibración de las sondas necesarios para el cálculo de incertidumbres, similar a *ChamberCalibration* para las cámaras. Los atributos son:

- *IdProbeCalibration*: clave primaria de esta tabla.
- *IdProbe*: clave primaria de la tabla *Probe*.
- *Date*: fecha de la calibración.
- *CertificateNumber*: código del certificado de calibración.
- *TempLow, TempHigh*: temperaturas límite de las calibraciones.
- *Resolution*: resolución de la sonda.
- *Linearity*: valor de la linealidad del instrumento.
- *Uncertainty*: incertidumbre estándar.
- *Drift*: deriva en las sondas.
- *A0, B0, C0, D0, SR*: valores del polinomio de mejor ajuste para la corrección de la temperatura medida.

Los valores de las calibraciones se pueden encontrar en la tabla 2. Las claves primarias, la fecha y los códigos de los certificados de calibración se observan en la tabla 11.

Tabla 11: Tabla ProbeCalibration.

IdProbeCalibration	*IdProbe*	*Date*	*Certificate*
1	2	2018-08-08	9359
2	3	2018-08-08	9360
4	5	2018-08-08	9361
5	6	2018-08-08	9362

4.1.3.11. Tabla ProbeType

La tabla *ProbeType* tiene la función de indicar cuál es el tipo de sonda que se está usando. Hasta ahora solo habrá dos registros – véase la tabla 12 –, pero más adelante se podrán incluir más tipos de sondas en esta tabla. Los campos que tienen son:

- *IdProbeType*: clave primaria de esta tabla.
- *Type*: tipo de sonda. Si este campo es *T*, la sonda es de temperatura. Si es *H*, la sonda es de humedad relativa.

Tabla 12: Tabla ProbeType.

IdProbeType	*Type*
1	T
2	H

4.1.3.12. Tabla ProgramTest

La tabla *ProgramTest* almacena registros que definen programas de ensayos. Estos programas tendrán ciertas características que los definen como son las tolerancias o el número de ciclos. De esta manera, los atributos de esta tabla son:

- *IdProgramTest*: clave primaria de la tabla.
- *ProgramNameApp*: nombre del programa del ensayo que se mostrará en la aplicación.
- *NumberOfCycles*: número de veces que debería repetirse el programa.
- *MinimunDwellTime*: mínimo periodo de residencia del ensayo.
- *ImageRute*: ruta de la imagen en miniatura que se verá en la aplicación al seleccionar dicho programa.
- *DwellLowerToleranceMaxTemp, DwellUpperToleranceMaxTemp, DwellLowerToleranceMinTemp, DwellUpperToleranceMinTemp*: definen las tolerancias alrededor de las temperaturas más alta y más baja de funcionamiento en el ensayo durante los periodos de residencia. Su unidad es el ºC.
- *RampUpperTolerance, RampLowerTolerance*: tolerancias alrededor del gradiente de temperatura. La unidad es ºC / min.

Un ejemplo de los valores de los campos de un programa de un ensayo de frío aparece en la tabla 13.

Tabla 13: Tabla ProgramTest.

IdProgramTest	*ProgramNameApp*	*NumberOfCycles*	*MinimunDwellTime*
4	Cold	1	54000
ImageRute	***DLTMaxTemp***	***DUTMaxTemp***	***DLTMinTemp***
- Ruta imagen -	5	0	0
DUTMinTemp	***RampUpperTol***	***RampLowerTol***	
5	1	2	

4.1.3.13. Tabla ProgramTest_Alarm

Al igual que *Chamber_Probe* o *BatchIDTestChamber_Probe*, la tabla *ProgramTest_Alarm* relaciona claves primarias. Por tanto, los campos que tiene son *IdProgramTest* e *IdAlarm*, claves primarias de ambas tablas. La tabla 14 muestra los registros guardados para este proyecto.

Tabla 14: Tabla ProgramTest_Alarm.

IdProgramTest_Alarm	*IdProgramTest*	*IdAlarm*
1	1	1
2	1	2
3	2	1
4	2	2
6	4	1
7	4	2

4.1.3.14. Tabla Segment

La tabla *Segment* tiene como objetivo almacenar los diferentes tramos que deberían componer un ciclo de un programa. Un segmento o tramo puede ser de residencia o de rampa. La residencia se define como un grupo de muestras que se encuentran dentro de un rango alrededor de cierto valor de temperatura. La rampa es el tramo intermedio que existe entre dos periodos de residencia.

Esta tabla no es estrictamente necesaria pero, como SIMPATI tiene una interfaz similar, se ha decidido introducir esta funcionalidad.

Los campos de los registros de esta tabla son:

- *IdSegment*: clave primaria de esta tabla.
- *IdProgramTest*: clave primaria de *ProgramTest*.
- *NumberOfSegment*: orden o índice del segmento.
- *Time*: instante de tiempo en segundos en el que comienza ese tramo del programa.

- *Temperature*: temperatura en ºC al comienzo del tramo.
- *Humidity*: h.r. en % al comienzo del segmento.

En la tabla 15 aparecen los diferentes registros para el programa expuesto en 4.1.3.12., mientras que la figura 23 muestra el perfil de dicho ciclo de programa.

Tabla 15: Tabla Segment para IdProgramTest igual a 4.

IdSegment	*IdProgramTest*	*NumberOfSegment*	*Time*	*Temperature*	*Humidity*
13	4	1	NULL	25	NULL
14	4	2	457	25	NULL
15	4	3	557	-5	NULL
16	4	4	1466	-5	NULL
17	4	5	1564	25	NULL

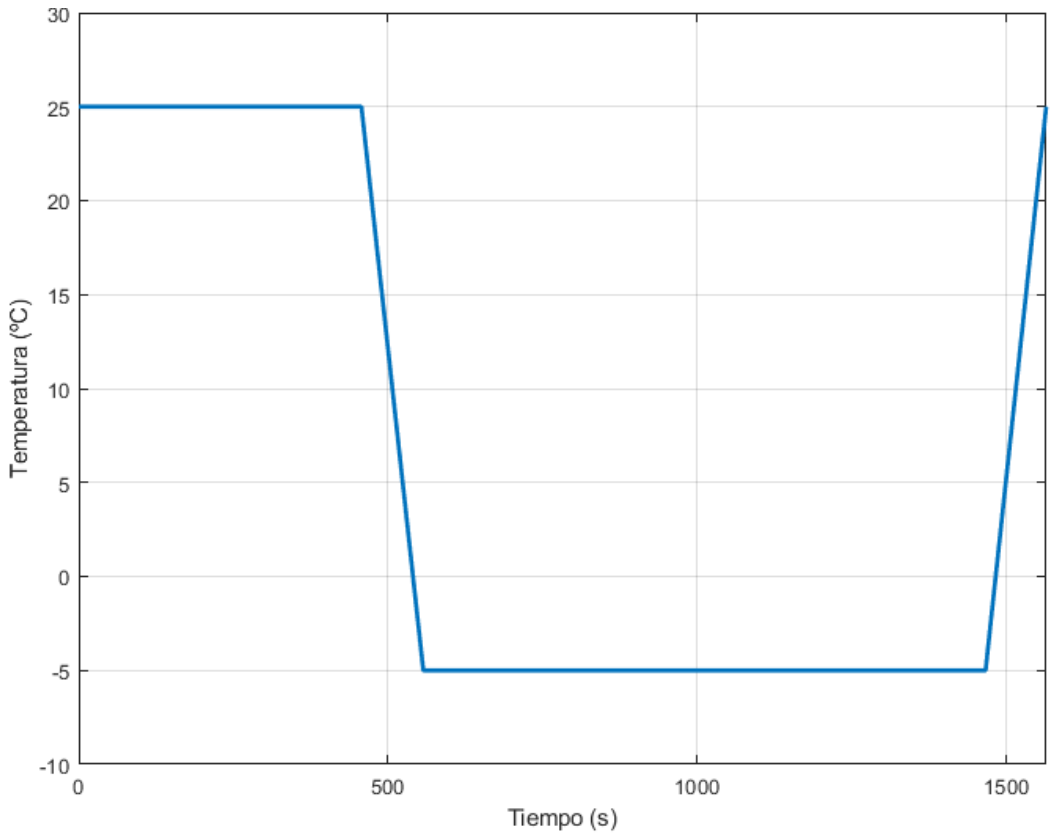

Figura 23: Perfil del ensayo de IdProgramTest igual a 4.

4.1.3.15. Tabla Uncertainty

La tabla *Uncertainty*, al contrario que la anterior, es completada por la aplicación. Guarda los tramos que ha detectado que ha realizado el ensayo y los va almacenando con los atributos siguientes:

- *IdUncertainty*: clave primaria de la tabla.
- *IdTest*: ID del ensayo para poder relacionar el resto de datos.
- *SectionDescription*: descripción cualitativa del tramo como pueden ser *"Short dwell period"* – tiempo de residencia demasiado pequeño –, *"Ramp period"* o rampa, *Normal dwell period* o tiempo de residencia correcto, entre otros.
- *NumberOfSection*: es el índice del tramo para, una vez se realiza el análisis completo, ver el orden que ha seguido el ensayo.
- *ReferenceTemperature*: temperatura media en el caso de que haya sido un periodo de residencia correcto.
- *ExpandedUncertaintyTemp*: incertidumbre expandida en ese tramo. Solo se guarda en el caso de que sea un periodo de residencia correcto.
- *ReferenceHumidity*: similar a *ReferenceTemperature* pero para la humedad relativa.
- *ExpandedUncertaintyHum*: similar a *ExpandedUncertaintyTemp* pero para humedad relativa.

Más adelante se expondrá un ejemplo de un ensayo de frío real y se mostrarán los registros generados para esta tabla.

4.2. IDE y soporte software

La aplicación de monitorización y supervisión se ha desarrollado en *C#* con el soporte de .NET Framework por Microsoft, todo ello con el IDE llamado Visual Studio 2019. A continuación, se describirán tanto la plataforma .NET como el lenguaje C# y Visual Studio y los motivos por los que se han elegido estas herramientas.

4.2.1. .NET Framework

.NET Framework es una plataforma software desarrollada por Microsoft, que actualmente es *open-source*. Incluye una gran librería de clases llamada *Framework Class Library,* o *FCL* por sus siglas, que provee soporte para el uso de varios lenguajes de programación, entre ellos C# o Python.

Además, los programas que usan esta plataforma se ejecutan en un entorno software llamado *Common Language Runtime*, el cual proporciona servicios como seguridad, gestión de memoria y tratamiento de excepciones.

La FCL es muy amplia, llegando a disponer de servicios de interfaz de usuario, acceso a datos, conectividad con bases de datos, criptografía, desarrollo de aplicaciones web o comunicaciones en red.

Gracias a estas características, se puede concluir que es una plataforma que engloba todos los aspectos recogidos en el alcance de este proyecto, desde la base de datos al cálculo numérico para las incertidumbres. Además, es evidente la escalabilidad que aporta gracias a su soporte para varios lenguajes, entre otras propiedades.

4.2.2. Lenguaje C#

C# es un lenguaje de propósito general diseñado específicamente para .NET. De esta manera, aunque es posible programar aplicaciones .NET en otros lenguajes, C# es mucho más sencillo y óptimo porque no requiere elementos heredados para funcionar. Por esta razón, C# es usualmente conocido como el lenguaje nativo de .NET.

Las características principales de este lenguaje son:

- **Sencillez.**

  Muchos elementos que son necesarios en otros lenguajes pero no en .NET son eliminados. Por ejemplo, el código es autocontenido evitando ficheros adicionales al código fuente. Otro caso es el tamaño fijo de los tipos de datos básicos, facilitando la portabilidad.

- **Orientación a objetos.**

  Como era de esperar, C# soporta las peculiaridades de la programación orientada a objeto como pueden ser la encapsulación, la herencia y el polimorfismo.

- **Gestión automática de memoria.**

- **Seguridad de tipos.**

  El lenguaje incluye formas para asegurar que el acceso a tipos de datos no falle. Por ejemplo, solo se admiten conversiones entre tipos compatibles, no se permite usar variables no inicializadas, etc.

- **Sistema de tipos unificado.**

- **Extensibilidad de tipos básicos.**

- **Extensibilidad de operadores.**

La versatilidad que proporciona C# es el motivo principal por el cual se ha elegido este lenguaje en el presente proyecto, así como en los trabajos futuros que acometerá ATN al respecto de *Virtual Lab.*

4.2.3. Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo para .NET Framework. Este IDE es compatible con mutitud de lenguajes de programación, tal y como se comento anteriormente. Además, es posible añadirle las funcionalidades que aporta Windows Azure.

Entre sus propiedades principales se encuentran:

- **Desarrollo rápido de aplicaciones.**

  Existen herramientas de diseño de elementos gráficos que permiten crear interfaces de usuario rápidamente. Si fuera necesario, también es posible acceder al código gracias a las vistas del editor.

- **Acceso a datos y servidores.**

  Proporciona un *Explorador de servidores* que permite iniciar sesión en los servidores conectados y explorar su bases de datos, crearlas o modificarlas.

- **Funciones de depuración de código.**

  Dispone de un depurador de código para ejecutarlo paso a paso localmente o de forma remota, haciendo pausas en puntos de interrupción o siguiendo rutas de ejecución.

- **Manejo de errores.**

  Ofrece una ventana llamada *Lista de errores*, que muestra advertencias, errores o mensajes que se producen al escribir el código.

- **Ayuda y documentación.**

  Gracias a Microsoft IntelliSense, se puede acceder a la ayuda con documentación y ejemplos de los fragmentos de código deseados.

Al ser la herramienta natural para usar en aplicaciones .NET con C#, Visual Studio 2019 ha sido el IDE utilizado en este trabajo.

5 APLICACIÓN DESARROLLADA

Este capítulo tratará sobre la aplicación desarrollada en C#, desde la arquitectura que soporta el contexto de la base de datos a los formularios gráficos que constituyen la interfaz con el usuario. En primer lugar, se detallarán los objetivos concretos de la aplicación. Luego, se realizará una descripción de NuGet y los paquetes que se han utilizado. A continuación, se explicarán las cuatro partes de las que consta la arquitectura del proyecto, tres de las cuales son las necesarias para la conexión con las bases de datos. Finalmente, se detallará el código de los formularios y el cálculo de incertidumbres.

5.1. Objetivos de la aplicación

La solución software deberá implementar una serie de formularios o ventanas que sirvan como interfaz para el usuario. En concreto, una vez iniciada la aplicación, se deberá abrir una ventana en la que se puedan seleccionar:

- ID del ensayo.
- Lote del ensayo.
- Cámara utilizada.
- Sondas de la cámara.
- Programa del ensayo.
- Número de ciclos que tiene el ensayo.
- Alarmas a monitorizar.

Además, en este menú principal existirá un botón que inicie la monitorización una vez se hayan seleccionado todos los parámetros. También deberá aparecer una imagen que muestre la curva ensayo al seleccionar el programa para que, en caso de duda, el usuario pueda comprobar visualmente que ése es el perfil que espera.

Una vez iniciado el ensayo con todos los parámetros correctamente, la aplicación tratará de buscar qué datos de la base de datos coinciden con las especificaciones definidas en el menú principal y arrancará otra ventana en la que se dé comienzo a la monitorización. Dicha ventana mostrará si el ensayo está en vivo o no – el ensayo puede haberse producido en un tiempo anterior y por tanto ya habría finalizado o puede estar en proceso en ese momento –, mensajes de interés para el operario, el progreso en los cálculos y las alarmas que se han activado. Contará también con botones para volver a la pantalla anterior, ya sea una vez terminado el ensayo o durante el análisis. En este último caso, los datos que ya hayan sido tratados se guardarán para continuar más adelante por donde se pausó la aplicación.

Si fuese necesario, debe haber más formularios para mostrar errores en la introducción de parámetros o para mostrar más datos como los segmentos de un programa.

La aplicación tendrá como entrada los parámetros de identificación del ensayo y sus muestras primarias. Una vez los datos sean analizados y tratados – la temperatura corregida, los tramos, las incertidumbres... –, estos

serán enviados y guardados en la base de datos en la tabla correspondiente.

5.2. Paquetes de NuGet

NuGet es una plataforma en las que los desarrolladores pueden crear, compartir y consumir código. A estos códigos públicos se los llama paquetes. NuGet, por tanto, es el método que usa Microsoft para el intercambio de código así como para acceder, alojar y crear dichos paquetes. La forma de distribuir NuGet es como una extensión de Visual Studio. A partir de la versión de 2012, NuGet está preinstalado por defecto.

Estos paquetes actúan como librerías que resuelven problemas de código. Los paquetes que se han instalado para la aplicación de monitorización se detallan en los siguientes apartados.

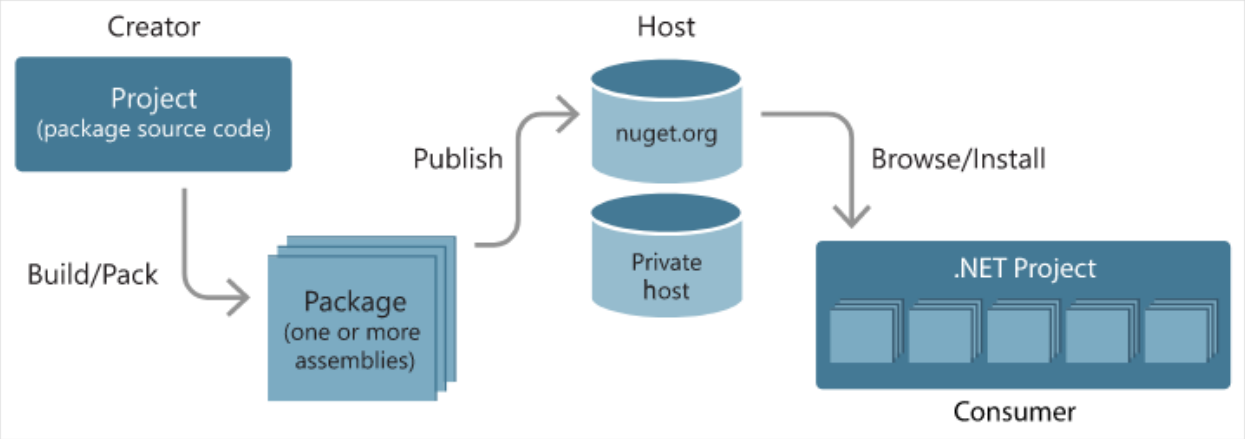

Figura 24: Desarrollo de un paquete NuGet (fuente: https://docs.microsoft.com/en-us/nuget/what-is-nuget)

En la figura 24 se observa como existen tres agentes durante el desarrollo de un paquete de NuGet: el creador del paquete, su *host* y el consumidor.

5.2.1. AutoMapper

AutoMapper es un paquete que tiene como objetivo sustituir el código que es necesario para mapear clases diferentes. Éste es un problema muy recurrente en la programación orientada a objeto. De esta manera, configurando correctamente el paquete y utilizando las clases y métodos disponibles, el mapeo entre objetos se puede conseguir de forma sencilla y rápida. Más adelante se explicará por qué es necesario usar esta librería en el proyecto.

5.2.2. EntityFramework

EntityFramework es un paquete desarrollado por Microsoft para mejorar el acceso y el tratamiento de datos. Gracias a él, las tablas de las bases de datos se pueden modelar como objetos y propiedades específicas del dominio del proyecto, trabajando a un nivel más elevado de abstracción, resolviendo parte del problema del mapeo objeto-relacional o ORM.

El mapeo objeto-relacional es un mecanismo utilizado en informática para establecer una relación entre la base de datos y una base de datos orientada a objetos virtual. Este mapeo permite el uso de características propias de la programación orientada a objetos como son la herencia o el polimorfismo.

5.2.3. Ninject

Ninject tiene un objetivo similar a Automapper pero resolviendo el problema de inyección de dependencias que puede surgir en las aplicaciones. Este problema surge en muchas ocasiones en momentos posteriores a la realización de un proyecto, cuando se está ampliando o modificando una parte de él.

La inyección de dependencias consiste en independizar partes del software haciendo que se comuniquen únicamente a través de una interfaz. Este mecanismo permite que los módulos del programa no tengan por qué tener conocimiento total sobre las funciones de los submódulos que utilizan, lo que generaría una dependencia fuerte, sobre todo en vistas a ampliar o modificar el proyecto.

En la inyección de dependencia están involucrados:

- Las interfaces que determinan la función de los servicios.
- Los servicios que implementan las interfaces a inyectar.
- Los clientes que hacen uso de los servicios.
- El inyector, cuya función principal es la de construir e inyectar los servicios a los clientes.

5.3. Solución propuesta

La solución software se ha denomidado *VirtualLab_TestBinding_v2*. Consta de cuatro proyectos:

- ***VirtualLab_TestBinding_v2.***

  Implementa el cálculo de incertidumbres, los formularios, el mapeo, los *data readers* y la inyección de dependencia.

- ***VirtualLab_TestBinding_v2.Data.***

  Contiene las clases que representan las tablas de la base de datos. Suelen denominarse repositorios. Necesita una infraestructura de código para funcionar. Estos archivos han sido provistos por ATN.

- ***VirtualLab_TestBinding_v2.Model.***

  Define el modelo de la base de datos.

- ***VirtualLab_TestBinding_v2.Service.***

  Determina los servicios de la base de datos. Por ejemplo, contiene métodos como *GetAll,* el cual realiza una consulta a todos los registros de la tabla requerida.

VirtualLab_TestBinding_v2.Data, VirtualLab_TestBinding_v2.Model y *VirtualLab_TestBinding_v2.Service* son módulos que han sido definidos y dirigidos por ATN y en los cuales no se ha entrado en detalle en este trabajo. Así pues, *VirtualLab_TestBinding_v2* es realmente el módulo que se ha creado íntegramente para realizar este proyecto.

A continuación, se explicará en detalle cada uno de los submódulos que componen *VirtualLab_TestBinding_v2*. Los anexos del final de este documento contienen el código completo de los archivos no proporcionados por ATN.

5.3.1. View Models

Con el fin de no modificar de forma descontrolada las tablas que recoja la aplicación de la base de datos, se ha decidido crear un *View Model* para cada una de ellas. Un *View Model* es igual que el modelo de la tabla pero en otra instancia, en el contexto de los formularios y la aplicación y no en el del modelo de la base de datos. De esta manera, en el caso de que un objeto de tipo *View Model* sea modificado, no se verá cambiada la tabla a la que hace referencia en la base de datos.

Para que los cambios que se producen en un *View Model* sean efectivos en el servidor, el *View Model* debe estar mapeado con la clase de la tabla a que hace referencia y, además, debe ejecutarse un *commit*. La figura 25 muestra las distintas clases creadas de *View Models* para representar todas las tablas de la base de datos.

Para usar cualquiera de estos *View Models,* es necesario:

1) Declarar la instancia de la tabla correspondiente de la base de datos.
2) Declarar la instancia de *View Model* equivalente.

3) Mapear ambas instancias entre sí.

El código siguiente es un ejemplo de cómo se inicializan la tabla *Chamber* y su *View Model*, *VMChamber*, para luego rescatar todos los registros de esa tabla en la base de datos y finalmente realizar el mapeo.

```
1. List<Chamber> listChamber;
2. List<VMChamber> listVMChamber;
3. listChamber = chamberService.GetAll().ToList();
4. listVMChamber = mapper.Map<List<Chamber>, List<VMChamber>>(listChamber);
```

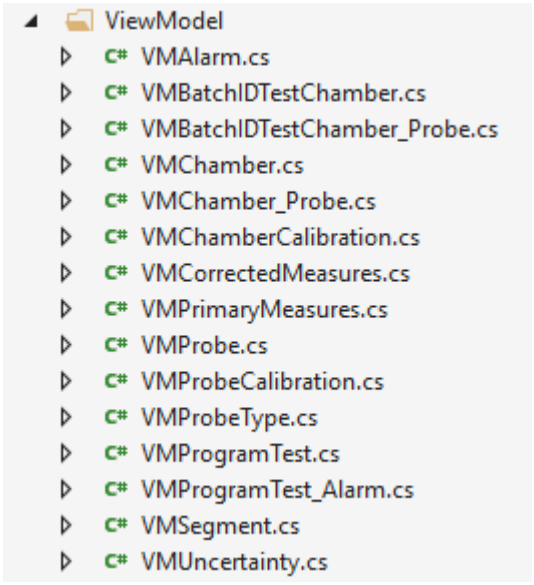

Figura 25: View Models del proyecto.

5.3.2. Mapping

El mapeo se realiza, por claridad, en tres archivos distintos: uno de configuración de AutoMapper y dos más de relaciones entre los *View Models* y las entidades de la base de datos – véase la figura 26 –.

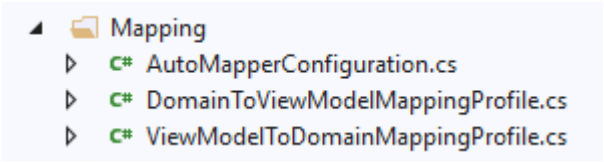

Figura 26: Archivos para el mapeo.

El de configuración de AutoMapper fue proporcionado por ATN. Los mapeos entre el dominio – de la base de datos – y los *View Model* se realizan gracias a los comandos de AutoMapper. Se muestra el código para mapear de *ViewModelToDomainMappingProfile.cs* de la tabla *ProbeCalibration*:

```
1. CreateMap<VMProbeCalibration, ProbeCalibration>();
```

En el caso de *DomainToViewModelMappingProfile.cs*:

```
1. CreateMap<ProbeCalibration, VMProbeCalibration>();
```

Es necesario realizar ambos mapeos porque no son equivalentes ni conmutativos. Así pues, ya se podrían realizar transformaciones entre *ProbeCalibration* y su *view model* asociado, *VMProbeCalibration,* y viceversa.

5.3.3. Infrastucture

Esta carpeta contiene los archivos necesarios para realizar correctamente la inyección de dependencia de las interfaces y las instancias correspondientes. Las dependencias que hay que inyectar son las referentes a los repositorios del proyecto *Data* y sus interfaces, los servicios de la base de datos y sus interfaces, la *UnitOfWork* encargada de realizar el acceso a la base de datos, el AutoMapper y el contexto de la base de datos llamado *DbFactory*. Éste último debe ser del tipo *singleton*, es decir, sólo puede existir un contexto de base de datos para evitar errores.

El repositorio y la unidad de trabajo o *unit of work* tiene como objetivo crear una capa de abstracción entre el acceso a datos en la base de datos y la aplicación software, por eso son necesarias en esta solución.

El siguiente código de ejemplo enseña los comandos necesarios para los repositorios y los servicios de *Probe*, así como de la *UnitOfWork,* el contexto de la base de datos *DbFactory* y el AutoMapper, como se expuso anteriormente.

```
1. Bind(typeof(IUnitOfWork)).To(typeof(UnitOfWork));
2.
3. Bind(typeof(IDbFactory)).To(typeof(DbFactory)).InSingletonScope();
4.
5. IMapper d = AutoMapperConfiguration.Configure();
6. Bind(typeof(IMapper)).ToConstant(d);
7.
8. Bind(typeof(IProbeRepository)).To(typeof(ProbeRepository));
9. Bind(typeof(IProbeService)).To(typeof(ProbeService));
```

5.3.4. Data Readers

Los *Data Readers* son los módulos que transmiten los datos de las muestras primarias al resto de la aplicación. Es, por tanto, donde se produce la recogida de los datos de la tabla *PrimaryMeasures*. Los *Data Readers* constan de sus interfaces, sus *data readers* y sus tipos de datos propios llamados *Data.* El *lector de datos* que concierne a este trabajo es el de temperatura. A continuación se detallarán sus características.

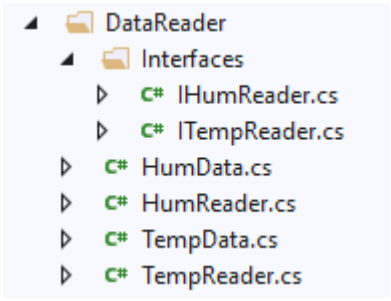

Figura 27: Archivos de DataReader.

5.3.4.1. ITempReader

ITempReader es la interfaz del lector de temperaturas. Al ser una interfaz de C#, solo nombra los métodos que han de implementarse en la instancia de la clase. La interfaz del *data reader* de temperatura es la siguiente – se ha obviado la cabecera –:

```
1. namespace VirtualLab_TestBinding_v2.DataReader.Interfaces
2. {
3.     interface ITempReader
4.     {
5.         TempData Get(int listIndex);
6.         bool TestFinished();
7.         int GetNumberOfSamples();
8.     }
9. }
```

5.3.4.2. TempReader

En *TempReader* se implementa la instancia de *ITempReader* que se encarga de obtener los registros. En primer lugar, una vez se inicializa por primera vez la clase, el código ejecuta el constructor, el cual recoge el ID del ensayo y el ID de la sonda para poder identificar los datos en la tabla. En este caso:

```
1. public TempReader(string idTest, int idProbe)
2. {
3.     iDTest = idTest;
4.     iDProbe = idProbe;
5.     InitializeDataReader();
6. }
```

Donde *InitializeDataReader* es:

```
1. private void InitializeDataReader()
2. {
3.     IUnitOfWork unitOfWork = CompositionRoot.Resolve<IUnitOfWork>();
4.     IMapper mapper = CompositionRoot.Resolve<IMapper>();
5.
6.     IPrimaryMeasuresService primaryMeasuresService =
       CompositionRoot.Resolve<IPrimaryMeasuresService>();
7.
8.     IProbeService probeService = CompositionRoot.Resolve<IProbeService>();
9.
10.    List<Probe> listProbe = probeService.GetAll().ToList();
11.    Probe probeCode = listProbe.FirstOrDefault(u => u.IdProbe == iDProbe);
12.
13.    // listPrimaryMeasures is the list of registers that match with the IDTest and
       the ProbeCode.
14.    List<PrimaryMeasures> listPrimaryMeasures =
       primaryMeasuresService.GetAll().ToList();
15.    listPrimaryMeasures = listPrimaryMeasures.Where(u => u.IdTest == iDTest &&
       probeCode.Code == u.ProbeCode).ToList();
16.
17.    listVMPrimaryMeasures = mapper.Map<List<PrimaryMeasures>,
       List<VMPrimaryMeasures>>(listPrimaryMeasures);
18. }
```

InitializeDataReader inicializa todos los objetos que serán necesarios para tomar los datos de *PrimaryMeasures.* Primero se declaran la unidad de trabajo *unitOfWork*, a continuación el mapeo y los servicios de *PrimaryMeasures* y de *Probe.* A continuación, se crea una lista con todos los registros de la tabla *Probe* y se selecciona aquél en el cual coincide su atributo *IdProbe* con el entero *iDProbe* pasado a través del constructor. En otras palabras, se selecciona la sonda deseada. Entonces, se declara una lista de objetos *PrimaryMeasures* y se le añaden todos los registros de la tabla de la base de datos. Después, se filtran mediante su ID del ensayo y su ID de la sonda, obteniendo así todas las medidas que la sonda correspondiente ha recogido. Finalmente, se mapea esa lista de medidas primarias a una lista pero en el contexto de la aplicación, es decir, a un *view model* del tipo *VMPrimaryMeasures.*

Una vez ejecutado el constructor, solo queda invocar alguno de sus métodos en el momento oportuno. El método *Get* tiene dos implementaciones: si toma el índice de la muestra que se quiere recoger o si, en lugar del índice, toma como argumento la fecha y hora.

A pesar de que no se ha comprobado el funcionamiento con un ensayo real *online* debido a que el resto de la infraestructura que soporta este proyecto no ha sido desarrollada, se ha decidido probar mediante código si la supervisión funciona correctamente tanto en modo *Offline* como en modo *Online.* Para ello, al inicio del método *Get*, se establece que si el índice que se pasa como argumento es mayor que un número que será la muestra a partir de la cuál es test se considera *online,* entonces la muestra no se actualizará hasta pasado un tiempo. El objetivo es simular que las muestras llegan cada cierto tiempo a la base de datos como si estuviese la cámara funcionando en ese momento. En el apartado 5.4. se observará este comportamiento mediante los datos de un ensayo de frío real. Para ver el código, consúltese el Anexo.

Los dos métodos restantes son *GetNumberOfSamples,* que devuelve el número de muestras de esa sonda en

ese ensayo que existen en la base de datos, y *TestFinished* que arroja un boleano dependiendo de si está el test finalizado o si no lo está. Estos métodos se han realizado con vistas a la simulación, una vez se quiera implementar el *data reader* en una situación real, estos métodos deberán ser modificados con las nuevas especificaciones.

```
        public int GetNumberOfSamples()
        {
            return index;
        }

        public bool TestFinished()
        {
            bool aux = false;
            if (index >= listVMPrimaryMeasures.Count - 1)
            {
                aux = true;
            }
            return aux;

        }
```

5.3.4.3. TempData

TempData es la clase que implementa el objeto que devuelven los *data readers*. Consta de las cuatro propiedades que se muestran a continuación:

```
1. public class TempData
2. {
3.     public DateTime Date { get; set; }
4.     public double Temp { get; set; }
5.     public int Order { get; set; }
6.     public int IdPrimaryMeasures { get; internal set; }
7. }
```

Date es la hora y fecha de la muestra, *Temp* el valor numérico, *Order* el índice de la medida e *IdPrimaryMeasures* la clave primaria de la muestra original de la tabla *PrimaryMeasures.*

5.3.5. Formularios

La solución dispone de seis formularios que tienen como objetivo guiar al usuario eligiendo el ensayo que se quiere analizar y mostrarle las diferentes alarmas y mensajes de interés que se vayan produciendo.

5.3.5.1. Menú principal

Una vez iniciada la aplicación, el formulario llamado *Form1* es la ventana principal. En él se puede elegir qué alarmas se quieren activar, el programa del ensayo correspondiente, el número de ciclos de ese programa que debe realizar el ensayo, el ID del ensayo, el lote, la cámara y las sondas implicadas.

También aparecerá el logo de ATN en la parte superior izquierda del menú, mientras que en el centro se encontrará una imagen de la curva del ensayo.

Como valor añadido, se han implementado botones que seleccionen o descarten todas las opciones en las cajas con listas – las alarmas o las sondas –. Además, como cada ensayo tiene un identificador y éste, a su vez, un lote y una cámaras asociados, estos parámetros se rellenarán automáticamente. En el caso de que el usuario no crea que esos valores son los adecuados, los podrá modificar sin ningún problema. También las sondas serán elegidas de forma automática al tener relación con la cámara del ensayo.

Existe además la posibilidad de comprobar los segmentos definidos en el programa del ensayo. Estos segmentos son similares a los ya vistos en SIMPATI, facilitando de esta manera la labor de los técnicos de laboratorio.

Finalmente, una vez introducidos todos los datos requeridos, se pasará a la ventana del ensayo en curso al pulsar el botón *"Start monitoring"*. Si por el contrario, hay algunos campos que no han sido rellenados y se

intenta avanzar al siguiente formulario, aparecerá una ventana de error informando acerca de este hecho.

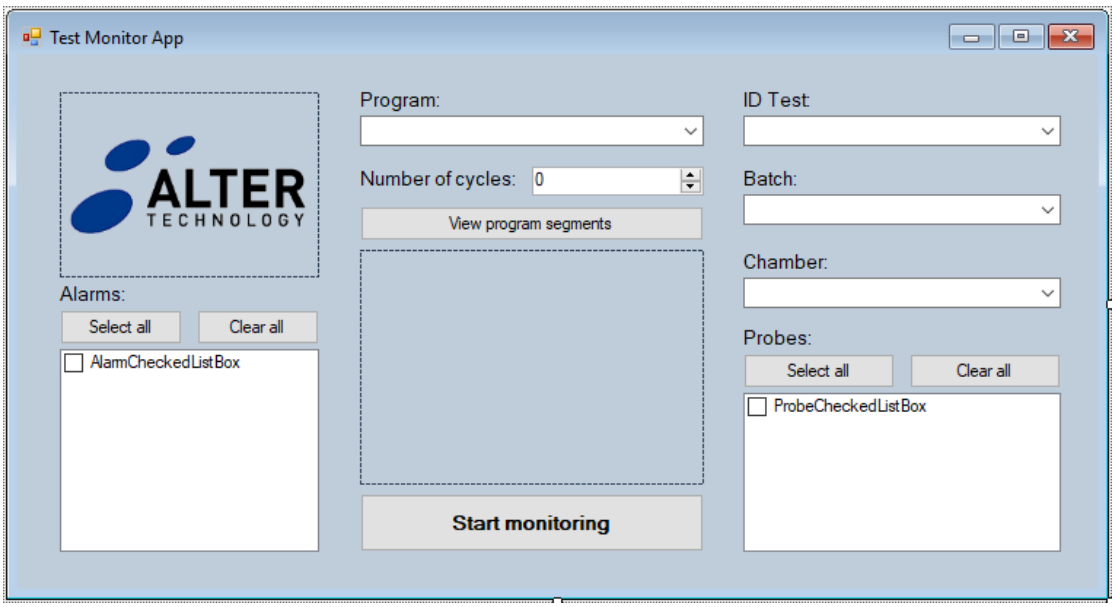

Figura 28: Menú principal de la solución.

Para confeccionar todas estas funcionalidades, es ineludible acceder a la base de datos. Para ello, el constructor de *Form1* ejecuta dos métodos: *InitializeComponent* e *InitializeForm.* El primero de ellos lo crea el propio IDE y sirve para inicializar los elementos del formulario. El segundo accede a las tablas de la base de datos para que puedan disponerse en las distintas pestañas y cajas de listas del formulario. Para consultar el código completo, véase el Anexo al final de este documento.

Además del constructor, se declaran los campos – variables internas de esa clase –. Entre ellos destacan las listas de los registros de las tablas, variables enteras auxiliares, la *unit of work*, el mapeo y las listas de objetos de las *checked list boxes*, es decir, de las cajas de listas de las sondas y las alarmas. Estos objetos son del tipo *ObjCheckedListBox,* cuyas propiedades se definen internamente como:

```
1. internal class ObjCheckedListBox
2. {
3.     public int Index { get; set; }
4.     public string Text { get; set; }
5.     public int ID { get; set; }
6. }
```

También se ha creado la clase interna *ObjComboBox* para los selectores del ID del ensayo, los lotes, las cámaras y el programa, y tiene la forma:

```
1. internal class ObjComboBox
2. {
3.     public string Text { get; set; }
4.     public int Value { get; set; }
5. }
```

Como se introdujo anteriormente, muchos de los parámetros se rellenan automáticamente. Para que esta característica funcione, es indispensable crear métodos que detecten que se ha cambiado el valor del elemento correspondiente y actúen en consecuencia. Estos son los denominados métodos activados por eventos. Visual Studio 2019 proporciona una interfaz interactiva en la que crear métodos asociados a los eventos deseados. La figura 29 muestra como el método *IDTestBox_SelectedIndexChanged* se ejecuta cuando el valor del elemento, en este caso *IDTestBox* – donde se selecciona el ID del ensayo –, cambia.

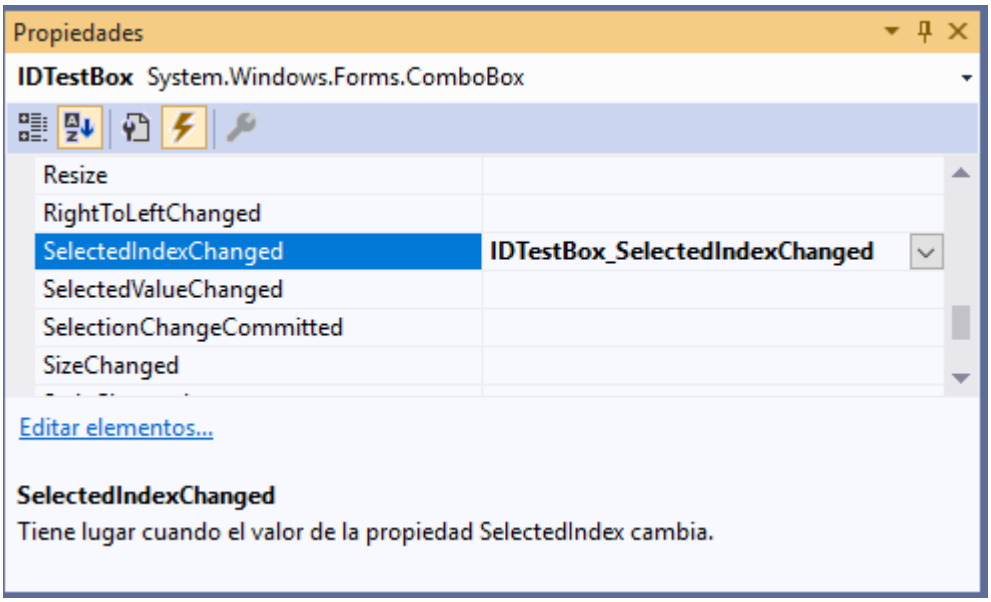

Figura 29: Método asociado al evento SelectedIndexChanged.

Los eventos que dan lugar a completar automáticamente ciertos parámetros son rellenar el ID del ensayo, la cámara o el programa. Si se escribe el ID del ensayo, el lote o *batch*, la cámara y las sondas se autocompletan. En el caso de elegir la cámara, se seleccionan las sondas. Si se escoge el programa, las alarmas son rellenadas.

Debido a la similitud de estos métodos únicamente se explicará *IDTestBox_SelectedIndexChanged.* Para ver el resto – *ProgramTestBox_SelectedIndexChanged* y *ChamberBox_SelectedIndexChanged* –, consúltese el Anexo correspondiente.

```
1. private void IDTestBox_SelectedIndexChanged(object sender, EventArgs e)
2. {
3.     // Get what IDTest is selecting to execute data readers later.
4.     if ((IDTestBox.SelectedItem as ObjComboBox).Text != null)
5.     {
6.     // 1) Get what IDTest is selected. Then take related IDBatch and IdChamber from
       database and put it onto the form.
7.     iDBatchIDTestChamberSelected = (IDTestBox.SelectedItem as ObjComboBox).Value;
8.     var registerSelectedAux1 = listVMBatchIDTestChamber.FirstOrDefault(u =>
       u.IdBatchIDTestChamber == iDBatchIDTestChamberSelected);
9.     // BatchBox modified.
10.    BatchBox.SelectedIndex = BatchBox.FindString(registerSelectedAux1.Batch);
11.    // ChamberBox modified.
12.    iDChamberSelected = (int)registerSelectedAux1.IdChamber;
13.    var registerSelectedAux2 = listVMChamber.FirstOrDefault(u => u.IdChamber ==
       iDChamberSelected);
14.
15.    // 2) Fill probes from chamber selected.
16.    ChamberBox.SelectedIndex = ChamberBox.FindString(registerSelectedAux2.Code);
17.    var registerSelectedAux3 = listVMChamber_Probe.Where( u => (int)u.IdChamber ==
       iDChamberSelected);
18.    // Clear list box.
19.    for (int i = 0; i < ProbeCheckedListBox.Items.Count ; i++)
20.    {
21.           ProbeCheckedListBox.SetItemChecked(i, false);
22.    }
23.    foreach (VMChamber_Probe register in registerSelectedAux3)
24.    {
25.        var registerSelectedAux4 = itemsProbeBox.FirstOrDefault(u => u.ID ==
       register.IdProbe);
26.    ProbeCheckedListBox.SetItemChecked(registerSelectedAux4.Index,true);
27.     }
28. }
```

Al comienzo del código anterior, se ha ejecutado un condicional con el objetivo de que, si la ventana del *ID Test* está vacía, siquiera se ejecute el resto del método. En el caso de que sí se haya seleccionado correctamente alguna opción del ID del ensayo, se busca que registro de la tabla *BatchIDTestChamber* tiene el mismo *IdBatchIDTestChamber* que la opción seleccionada. De esta manera, se obtienen el lote y la cámara asociados a ese registro y se modifican sus valores.

El siguiente paso es obtener las sondas que tiene esa cámara. Para ello, gracias al atributo *IdChamber*, se localiza los registros de la tabla *Chamber_Probe* que tienen ese mismo valor para dicho atributo. Finalmente, se limpia la lista de la caja de las sondas y se les pone un *tick* a las correspondientes que se han hallado anteriormente.

En cuanto al botón que abre la ventana para ver los segmentos del programa, el código se puede comprobar a continuación:

```
1.  private void ViewSegmentsButton_Click(object sender, EventArgs e)
2.  /* This displays segments tab. */
3.  {
4.      string programNameApp;
5.      if (iDProgramTestSelected != -1)
6.      {
7.            programNameApp = (listVMProgramTest.FirstOrDefault(u =>
              (int)u.IdProgramTest == iDProgramTestSelected) as
              VMProgramTest).ProgramNameApp;
8.      }
9.      else programNameApp = "Program not selected";
10.     SegmentsForm segmentsForm = new SegmentsForm (iDProgramTestSelected,
        programNameApp);
11.     segmentsForm.ShowDialog();
12. }
```

Al igual que con los métodos ya explicados, *ViewSegmentsButton_Click* está referido al evento *Click* del botón. Como su nombre indica, se ejecuta una vez se clickea en él. En primer lugar, comprueba que se ha seleccionado un programa válido. Luego, guarda en *programNameApp* el nombre de dicho programa. En caso contrario, si no es válido, *programNameApp* vale la cadena de caracteres "Program not selected". Finalmente, se crea un formulario del tipo *SegmentsForm* y se arranca. En el apartado 5.3.5.2. se explicará en detalle este formulario.

Aunque el botón que reza *"Start monitoring*" se activa de igual manera que la ventana de los segmentos con el evento *Click*, trata una funcionalidad diferente. La utilidad del método *TestButton_Click* consiste en transmitir los datos de supervisión que se han seleccionado en el menú principal al siguiente formulario, a la ventana de monitorización. Para ello, se inicializa una instancia de la ventana de monitorización – *testInProgressForm* – con la propiedad *StartMonitoring* igual a *false.*

```
1. TestInProgressForm testInProgressForm = new TestInProgressForm
2. {
3.     StartMonitoring = false
4. };
```

A continuación, se van tomando los datos de las tablas *ChamberCalibration*, *ProgramTest*, *ProbeCalibration, Alarm, Segment* y *BatchIDTestChamber* y se asignan valores a las propiedades de *testInProgressForm.* Éste es el mecanismo de paso de información entre formularios que se ha decidido implementar. Por último, se arranca la nueva ventana con el método *ShowDialog*.

En el caso de que al pulsar *"Start monitoring"* no todos los campos hayan sido rellenados, se despliega un cuadro de diálogo llamado *errorForm*. El punto 5.3.5.3. esclarece esta nueva ventana.

5.3.5.2. Ventana de los segmentos del programa

El formulario que muestra los segmentos del programa posee un constructor que tiene como argumentos *n*, un entero, y *pname*, una cadena de caracteres. El programa seleccionado será el valor de *n* y el nombre del programa es la cadena *pname*. Posteriormente, se inicializa el resto de la ventana con el método *InitializeSegmentsGridView.*

```
1. public SegmentsForm(int n,string pname)
2. {
3.     InitializeComponent();
4.     iDProgramTestSelected = n;
5.     programAppName = pname;
6.     InitializeSegmentsGridView();
7. }
```

En la figura 30 se puede observar esta ventana con el ejemplo del ensayo de frío.

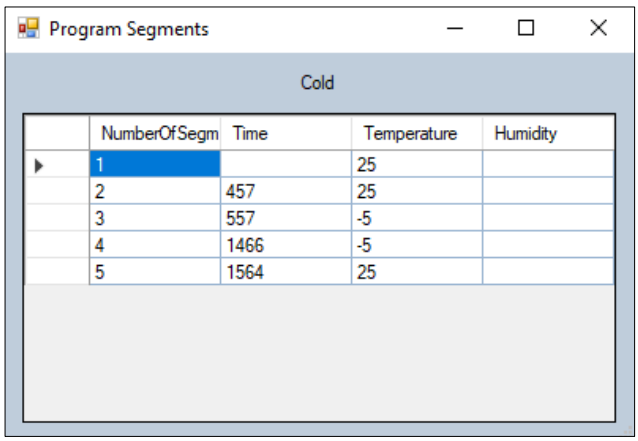

Figura 30: Formulario de los segmentos del programa.

El método *InitializeSegmentsGridView* realiza ciertas funciones. Inicialmente, escribe el nombre del programa en la caja que se observa en la parte superior. Luego, crea la lista con los segmentos que se han seleccionado y se añaden al elemento de tabla disponible.

```
1.  private void InitializeSegmentsGridView()
2.  {
3.       ISegmentService segmentService = CompositionRoot.Resolve<ISegmentService>();
4.       // Write program name in ProgramNameTextBox
5.       ProgramNameTextBox.AppendText(programAppName);
6.       // Fill SegmentGridView
7.       listSegment = segmentService.GetAll().ToList();
8.       listVMSegment = mapper.Map<List<Segment>, List<VMSegment>>(listSegment);
9.       listVMSegment = listVMSegment.FindAll(u=>u.IdProgramTest ==
         iDProgramTestSelected);
10.      List<VMSegmentOnGrid> listVMSegmentOnGrid = new List<VMSegmentOnGrid>();
11.      foreach (VMSegment register in listVMSegment)
12.      {
13.          listVMSegmentOnGrid.Add(new VMSegmentOnGrid { NumberOfSegment =
             register.NumberOfSegment, Time = register.Time, Temperature =
             register.Temperature, Humidity = register.Humidity });
14.      }
15.      BindingList<VMSegmentOnGrid> bindingListVMSegmentsOnGrid = new
         BindingList<VMSegmentOnGrid>();
16.      for (int i = 0; i < listVMSegmentOnGrid.Count; i++)
17.      {
18.           bindingListVMSegmentsOnGrid.Add(listVMSegmentOnGrid[i]);
19.      }
20.      bindingSourceSegments.DataSource = bindingListVMSegmentsOnGrid;
21.      SegmentsGridView.DataSource = bindingSourceSegments;
22.      Controls.Add(SegmentsGridView);
23. }
```

5.3.5.3. Mensaje de error

En el apartado 5.3.5.1. ya se aclaró que el cuadro de diálogo del error sólo aparecería en el caso de que no se rellenasen todos los parámetros necesarios. Si esto sucede, la ventana que aparece es la de la figura 31.

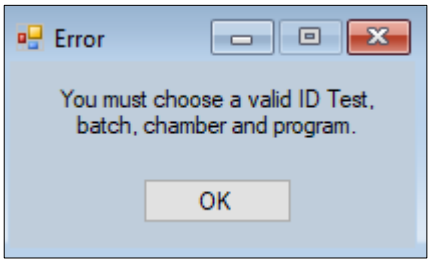

Figura 31: Cuadro de error.

5.3.5.4. Ventana de monitorización

En este formulario es donde realmente se produce el análisis del ensayo y el volcado de los resultados procedentes del cálculo de incertidumbres a la base de datos. Como se explicó anteriormente, al introducir correctamente los parámetros y pulsar el botón de *"Start monitoring"* del menú principal, se abre la ventana de la figura 32.

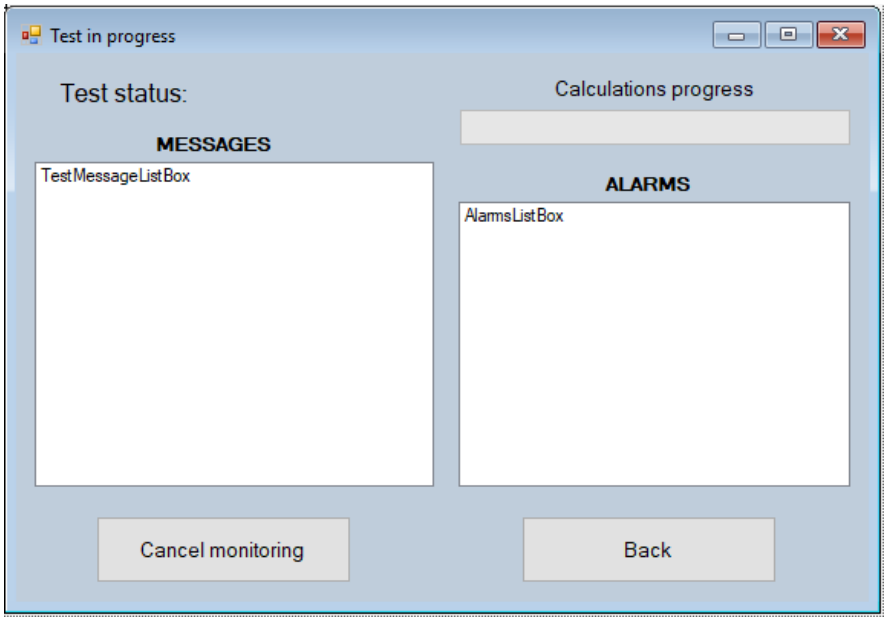

Figura 32: Ventana de monitorización.

El mecanismo de traspaso de información entre formularios consiste en el uso de propiedades, como se aclaró en uno de los apartados anteriores. De esta manera, la clase *TestInProgressForm* contiene las propiedades referentes a los parámetros del test. Además, posee campos auxiliares para el cálculo y para guardar las muestras corregidas con una lista de *VMCorrectedMeasures*.

También se ha creado una clase interna llamada *UncertaintyObject* con el objetivo de guardar los datos de los cálculos de incertidumbre de cada tramo. Las propiedades de este objeto se pueden ver a continuación:

```
1. internal class UncertaintyObject
2. {    public decimal StandardUncertainty { get; set; }
3.      public decimal Divisor { get; set; }
4.      public decimal SensityCoefficient { get; set; }
5.      public decimal Uncertainty { get; set; }
6.      public decimal DegreesOfFreedom { get; set; }
7.      public decimal TotalUncertainty { get; set; }
8. }
```

El primer paso en la ejecución es arrancar el constructor. Este constructor inicializa el componente – el formulario –, deshabilita el botón "Back", establece el *background worker* y lo reanuda.

```
1. public TestInProgressForm()
2. {
3.     InitializeComponent();
4.     BackButton.Enabled = false;
5.     InitializeBackgroundWorker();
6.     backgroundWorkerCalculations.RunWorkerAsync();
7. }
```

La inicialización del *background worker* se puede encontrar en el Anexo adjunto con el método *InitializeBackgroundWorker*.

En esta clase también se introduce el concepto de *BackgroundWorker*. Un *background worker* es un objeto que permite realizar operaciones en segundo plano de forma asíncrona. Dispone de varios métodos propios asociados a eventos como son *DoWork, RunWorkerCompleted* y *ProgressChanged.*

DoWork es el proceso que realiza en segundo plano. En este caso, aquí se produce el acceso a la base de datos a partir de los *data readers*, el cálculo de incertidumbres y la subida de resultados al servidor. Es la parte más importante de toda la aplicación pues consigue los objetivos propuestos principales.

La primera parte de este método es una declaración y asignación inicial de las variables y objetos necesarios como pueden ser las listas de muestras o los *data readers*. También se comprueba al principio si se ha producido una petición de cancelación. En ese caso, se resetearía el *unit of work* con el fin de que si existe alguna consulta a medias con el servidor, se resetee. Al final de esta primera parte se comprueba si el análisis para este ensayo ya se ha realizado. Para ello, se recupera la tabla *CorrectedMeasures* y se comprueba si hay algún registro que tenga los datos del test y la sonda en concreto. Si es así, se reanuda el análisis a partir de la última muestra leída.

En segundo lugar, ya comienza el bucle del cálculo de incertidumbres. Se comprueba si el ensayo ha finalizado con el método *IsTestFinished* para las sondas y los *data readers* correspondientes. Si no ha terminado, se toman muestras primarias de cada uno de los *data readers* y se añaden a la consulta que se realizará sobre la base de datos más adelante. Luego, se verifica que las muestras recogidas no son iguales a las anteriores. Si es así, se prosigue calculando en qué fase se encuentra dichas medidas – periodo de residencia o rampa –. Si el periodo actual es rampa y el anterior fue de residencia, se activa el *flag calculateUncertainty* para que, en el siguiente paso, se realice todo el cálculo de incertidumbres.

Una vez realizado el cálculo si era necesario, se añade un nuevo registro a la tabla *Uncertainty* con los datos del tramo recién analizado. En el caso de que el periodo de residencia haya sido menor del mínimo, no se calculan las incertidumbres sino que se activa un mensaje de alarma por pantalla. Si es una rampa, también se añaden a la base de datos pero con *"Ramp period"* como atributo *SectionDescription*.

Las subidas a la base de datos se producen siempre una vez se ha detectado la finalización de un periodo, ya sea una rampa o una residencia. Se ha decidido así para que no se sature de pequeñas consultas sino que se realicen menos pero de mayor tamaño. Gracias a este procedimiento, se ha mejorado considerablemente el rendimiento.

Además, el estado del ensayo – *online* u *offline* – se actualiza por pantalla.

Este bucle de supervisión finaliza una vez se detecta que el análisis del ensayo al completo ha terminado. Así, se muestra por pantalla un mensaje son la fecha a la que se ha terminado, se resetea el *background worker* y, para acabar, se reactiva el botón *"Back"* para volver al menú principal.

Además de *DoWork,* se encuentra *ProgressChanged*. Este método se utiliza para la comunicación con el resto del formulario. Se recuerda que *DoWork* es un proceso aparte del formulario que lo ha creado y no es posible la comunicación entre ambas entidades sin *ProgressChanged*.

Para invocar *ProgressChanged*, es obligatorio ejecutar el método *ReportProgress* disponible para los objetos del tipo *BackgroundWorker*. Por ejemplo, con una instancia del objeto *BackgroundWorker,* se puede enviar un entero – que puede ser o no un porcentaje – y un objeto genérico.

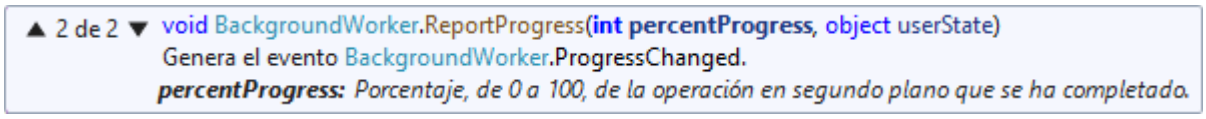

Figura 33: Argumentos del método ReportProgress.

Una vez en *ProgressChanged*, el segundo argumento que es del tipo *ProgressChangedEventArgs* es el que guarda tanto el entero como el objeto genérico de *ReportProgress*. Así, se ha decidido que el entero servirá para indicar qué tipo de mensaje es y el objeto genérico será la cadena de caracteres a escribir gracias a la estructura *switch*. El siguiente cuadro muestra el método *BackgroundWorkerCalculations_ProgressChanged*:

```
1.  private void BackgroundWorkerCalculations_ProgressChanged(object sender,
       ProgressChangedEventArgs e)
2.  {
3.     BackgroundWorker worker = sender as BackgroundWorker;
4.     switch (e.ProgressPercentage)
5.     {
6.            case 1: // Message list box
7.                TestMessageListBox.Items.Add(e.UserState);
8.                visibleItems = TestMessageListBox.ClientSize.Height /
                  TestMessageListBox.ItemHeight;
9.                TestMessageListBox.TopIndex = Math.Max(TestMessageListBox.Items.Count-
                  visibleItems + 1, 0);
10.           break;
11.           case 2: // Alarm list box
12.               AlarmsListBox.Items.Add(e.UserState);
13.               visibleItems = AlarmsListBox.ClientSize.Height /
                  AlarmsListBox.ItemHeight;
14.               AlarmsListBox.TopIndex = Math.Max(AlarmsListBox.Items.Count -
                  visibleItems + 1, 0);
15.           break;
16.           case 3: // Cancel button
17.               CancelMonitoringButton.Enabled = false;
18.               BackButton.Enabled = true;
19.               if (cancelMonitoringForm != null) cancelMonitoringForm.Close();
20.           break;
21.           case 4: // Progress Bar
22.               testProgressBar.Value = (int)Math.Truncate((e.UserState as
                  List<double>)[0]*100/(e.UserState as List<double>)[1]);
23.           break;
24.           case 5: // Online – offline test
25.               OnOffTestTextBox.Text = e.UserState as string;
26.           break;
27.           default:
28.           break;
29.    }
30. }
```

El último método que hace referencia a eventos del *background worker* es *RunWorkerCompleted,* que arranca cuando las instrucciones en *DoWork* llegan a su fin. Sin embargo, si se observa el final del código de *DoWork* en el caso de esta aplicación, al ser un bucle *while* que nunca actualiza su condición de entrada, siempre estará ahí. Se ha creado el método *RunWorkerCompleted* para evitar errores no controlados, aunque estrictamente no es indispensable.

Definido el *background worker*, se encuentra *CancelMonitoringButton_Click.* Como su nombre indica, se ejecuta cuando se pulsa el botón *"Cancel monitoring"*. El método crea un objeto del tipo *CancelMonitoringForm* y abre dicho formulario. Gracias al paso de datos a través de la propiedad *MonitoringCanceled*, la ventana de monitorización se cierra si el usuario ha decidido cancelar el análisis o sigue ejecutándose en caso contrario. En el apartado 5.3.5.5. se verá más profundamente la ventana surgida al clickear *"Cancel monitoring"*.

El otro botón que también consta de un método propio es el de *"Back"*. Al iniciarse el formulario, está

deshabilitado. Solo se activa una vez se ha realizado todo el análisis del ensayo programado. Por tanto, si se pulsa, cierra la ventana de monitorización y vuelve al menú principal.

El método *IsTestFinished* comprueba si ha finalizado el análisis del ensayo con argumentos la lista de enteros que guarda el ID de las sondas seleccionadas, la de *data readers* y la del número de muestras de cada sonda. Para ello, aumenta un contador auxiliar si se ha llegado al final del número de muestras recogidas y si el *data reader* devuelve un resultado positivo cuando se ejecuta su método *TestFinished* – véase 5.3.4.2. para consultar los métodos del *data reader* –.

Para comprobar si el test está online, se ha creado el método *IsTestOnline*. Posee los mismos argumentos que *IsTestFinished* y un planteamiento similar, salvo que ahora no se tienen en cuenta el número de muestras recogidas por cada sonda sino solamente si *TestFinished* da siempre un resultado positivo – *true* –.

Un problema a resolver es el hecho de que los *data readers* puedan tomar dos veces la misma muestra. Para detectar si ha sido así, el método *AllSamplesAreTheSame* devuelve *true*, si no se recupera *false.* El razonamiento es muy parecido a los dos últimos métodos descritos pero con la lista de boleanos *sameLastSample.*

Para comprobar las alarmas, también existen dos sobrecargas – dos series de argumentos distintos – para el método *CheckAlarms*. Una de ellas sirve para los instantes después del cálculo de la incertidumbre en un tramo mientras que la otra las comprueba en cualquier momento. La alarma que se desea comprobar debe ser introducida mediante su código como cadena de caracteres. A continuación, se muestra este método para los rangos de temperatura una vez calculada la incertidumbre del tramo.

```
1.  private void CheckAlarms(string code, List<decimal> listAverTemp, decimal
       expandedUncertainty, int? numberOfSection, object sender)
2.  /* Check alarms after calculating expanded uncertainty. */
3.  {
4.     BackgroundWorker worker = sender as BackgroundWorker;
5.
6.     // Case where alarms will be checked
7.     switch (code)
8.     {
9.        case "A01": // Temperature out of range
10.           foreach (decimal temp in listAverTemp)
11.           {
12.                 if ((temp + expandedUncertainty < maxTemp +
                    (decimal)VMProgramTest.DwellLowerToleranceMaxTemp &&
                    temp + expandedUncertainty > maxTemp -
                    (decimal)VMProgramTest.DwellUpperToleranceMaxTemp) ||
                    (temp + expandedUncertainty < minTemp +
                    (decimal)VMProgramTest.DwellLowerToleranceMaxTemp &&
                    temp + expandedUncertainty > minTemp -
                    (decimal)VMProgramTest.DwellUpperToleranceMaxTemp))
13.                 {
14.                 }
15.                 else worker.ReportProgress(2, "> Average temperature " +
                    temp.ToString("0.00") + " +- uncertainty " +
                    expandedUncertainty.ToString("0.00") + " out of range at section " +
                    numberOfSection + ".");
16.           }
17.       break;
18.       case "A02": // Humidity out of range
19.
20.       break;
21.       default:
22.       break;
23.    }
24. }
```

Si se implementasen más alarmas, éstas deberían crear un nuevo caso en el *switch*.

Por último, *AddToDBContext* toma como argumentos el mapeador, la lista de sondas seleccionadas, etc, y añade a la consulta la muestra corregida de *VMCorrectedMeasures.* Para ello, inserta el objeto recién creado

vMCorrectedMeasures al contexto de la base de datos gracias al *unit of work.*

```
1.  private void AddToDBContext(IUnitOfWork unitOfWork, IMapper mapper, int[]
       iDProbesSelected, int i, TempData tempSampleCorrected)
2.  /* Add a VMCorrectedMeasure to DbContext*/
3.  {
4.     VMCorrectedMeasures vMCorrectedMeasures = new VMCorrectedMeasures
5.     {
6.            IdPrimaryMeasures = tempSampleCorrected.IdPrimaryMeasures,
7.            IdTest = IDTestSelected,
8.            ProbeCode = ListVMProbe.First(u => u.IdProbe == iDProbesSelected[i]).Code,
9.            Date = tempSampleCorrected.Date,
10.           MeasureIndex = tempSampleCorrected.Order,
11.           CorrectedTemp = tempSampleCorrected.Temp,
12.           CorrectedHum = null
13.    };
14.
15.    unitOfWork.DbContext.CorrectedMeasures.Persist(mapper).InsertOrUpdate
       (vMCorrectedMeasures);
16.    listVMCorrectedMeasures.Add(vMCorrectedMeasures);
17. }
```

5.3.5.5. Ventana de cancelación del análisis

Si aún no ha terminado el análisis pero se quiere abortar, basta con pulsar el botón *"Cancel monitoring"*. Luego, se abrirá un cuandro de diálogo como el de la figura 32.

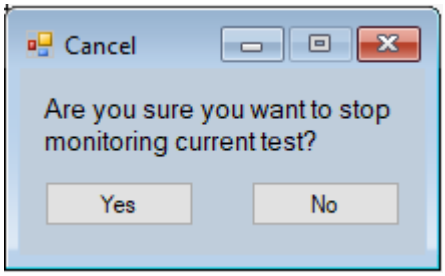

Figura 34: Ventana de cancelación del análisis.

Como era de esperar, en el caso de pulsar *"Yes"*, se volverá al menú principal. Si se pulsa *"No"*, se cierra esta ventana y prosigue el análisis. El resultado de la elección se pasa a *testInProgressForm* mediante la propiedad *MonitoringCanceled.* El código se muestra a continuación.

```
1.  namespace VirtualLab_TestBinding_v2.Forms
2.  {
3.     public partial class CancelMonitoringForm : Form
4.     {
5.            TestInProgressForm testInProgressForm;
6.
7.            public CancelMonitoringForm(TestInProgressForm _testInProgressForm)
8.            {
9.                   InitializeComponent();
10.                  testInProgressForm = _testInProgressForm;
11.           }
12.
13.           private void YesButton_Click(object sender, EventArgs e)
14.           {
15.                  testInProgressForm.MonitoringCanceled = true;
16.                  Close();
17.           }
18.
19.           private void NoButton_Click(object sender, EventArgs e)
20.           {
```

```
21.                 testInProgressForm.MonitoringCanceled = false;
22.                 Close();
23.             }
24.     }
25. }
```

5.4. Funcionamiento de la aplicación

El funcionamiento se ha comprobado con los datos de las medidas primarias de un ensayo de frío. Este ensayo fue analizado gracias a un archivo *Excel* de forma manual y servirá para comprobar los valores numéricos de las incertidumbres calculadas mediante la aplicación software de monitorización.

Las muestras del ensayo siguen la gráfica de *Excel* de la figura 18, en la página 32. Se ha programado para que sea una simulación de un ensayo en vivo que ha recogido en ese momento 1540 muestras – valor escogido aleatoriamente –. A partir de ese índice, las muestras serán entregadas por los *data readers* cada cierto tiempo, haciendo parecer que se producen las lecturas de manera realista. Las tablas *CorrectedMeasures* y *Uncertainty* comienzan vacías.

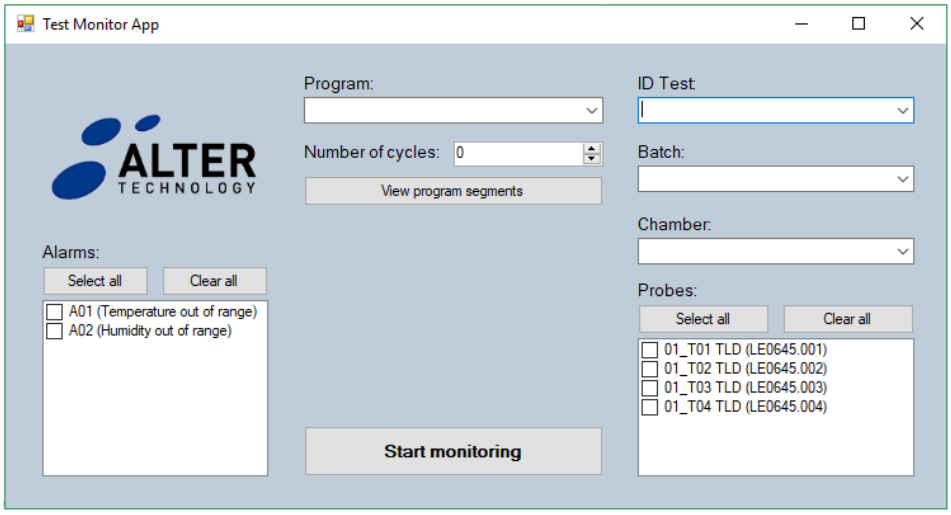

Figura 35: Menú principal.

Al iniciar la aplicación, se ve el menú principal de la figura 35. En él, no hay nada seleccionado. Tampoco está la imagen del ensayo puesto que no se ha elegido ninguno. Si se elige un ID del ensayo se completan todos los campos de la columna de la derecha. A su vez, si se selecciona el programa, se carga la imagen del mismo y se rellenan las alarmas – véase la figura 36 –.

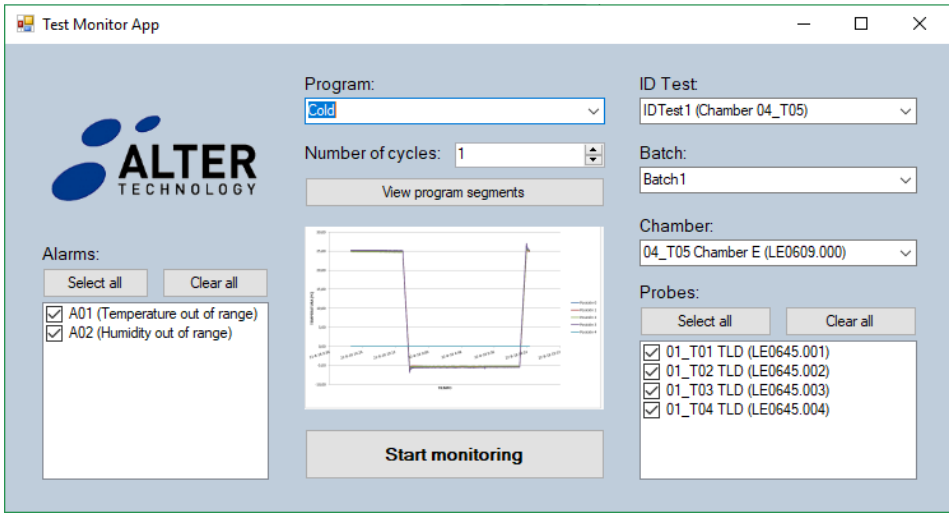

Figura 36: Menú principal para el ensayo de frío.

Al pulsar *"Start monitoring"*, se abre la ventana de la supervisión del ensayo. Además, se suceden los mensajes de inicio, ya sea de los *data readers* como de los cálculos. Nótese que el botón *"Back"* está deshabilitado, al contrario que *"Cancel monitoring"*.

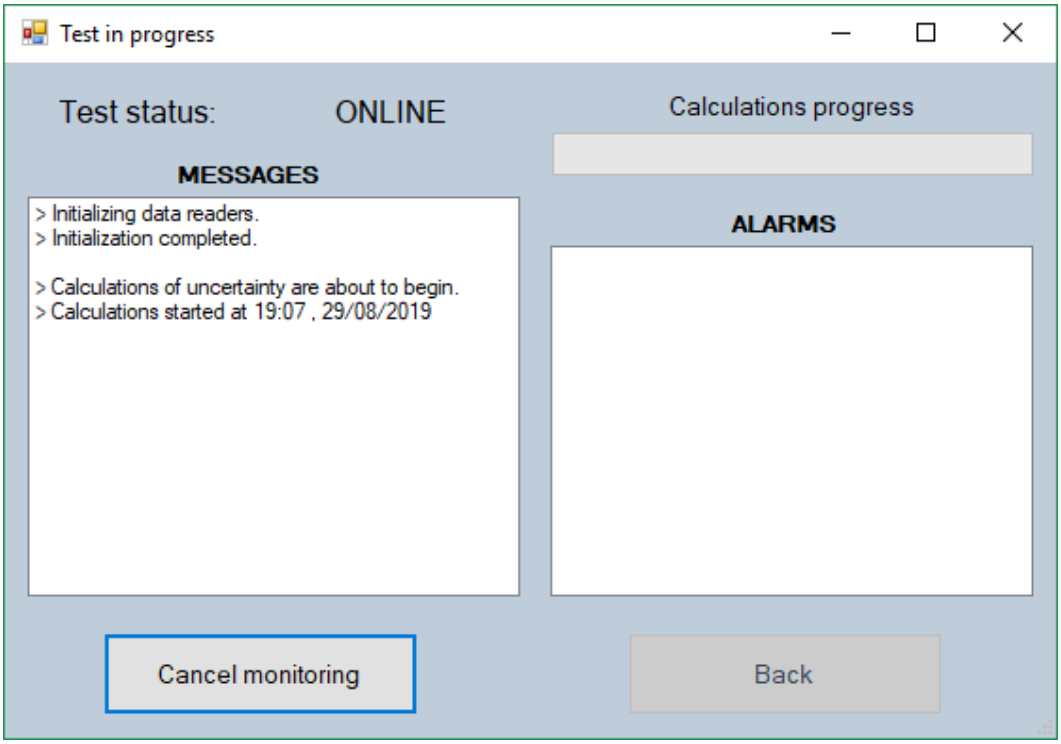

Figura 37: Ventana de monitorización al inicio del análisis.

Así pues, el test sigue su curso, actualizando la tabla *Uncertainty* con los datos de los tramos analizado. Por ejemplo, una vez se pasan las dos primeras secciones – una residencia y una rampa –, aparecen los registros de la figura 38 en la base de datos. También son almacenadas las muestras corregidas de temperatura en la tabla *CorrectedMeasures* – véase la figura 39 –.

DESKTOP-SB3H8F0\S...- dbo.Uncertainty

	IdUncertainty	IdTest	SectionDescrip...	NumberOfSec...	ReferenceTem...	ExpandedUnce...	ReferenceHu...	ExpandedUnce...
▶	35	IDTest1	Short dwell peri...	0	*NULL*	*NULL*	*NULL*	*NULL*
	36	IDTest1	Ramp period	1	*NULL*	*NULL*	*NULL*	*NULL*

Figura 38: Datos de los dos primeros tramos analizados.

Si se deja continuar el test, se van completando los tramos. En la figura 40, se observa como han terminado los tres primeros tramos. El primero de ellos ha sido de residencia pero demasiado corto, por este motivo no se ha calculado su incertidumbre. El segundo ha sido la rampa que se ve desde los 25 a los -5 ºC. El tercero es el periodo de residencia en torno a -5 ºC. En este caso si se ha calculado su residencia y se ha comprobado que está dentro del rango – no ha habido ninguna alarma al respecto –.

La figura 41, en cambio, es la ventana de monitorización al acabar el análisis. El estado del test es *offline* porque se ha detectado que ha terminado el ensayo. Además, aparecen dos mensajes diciendo *"Click back button to return to main menu"*. Esto se debe a que, una vez se termina la supervisión, ese mensaje aparece cada cierto intervalo de tiempo. Además, se ha deshabilitado el botón *"Cancel monitoring"* y se ha activado el botón *"Back"*.

Los registros finales de la tabla *Uncertainty* son los de la figura 42. Se comprueba que son cuatro tramos, dos de ellos de residencia y otras dos rampas.

DESKTOP-SB3H8F0\...orrectedMeasures

IdCorrectedM...	IdPrimaryMea...	IdTest	ProbeCode	Date	MeasureIndex	TestSection	CorrectedTemp
77926	2	IDTest1	01_T01	2018-08-21 12:4...	1	NULL	25,12561658920...
77927	4884376	IDTest1	01_T02	2018-08-21 12:4...	1	NULL	25,03477449427...
77928	4885938	IDTest1	01_T03	2018-08-21 12:4...	1	NULL	24,87397533746...
77929	4887500	IDTest1	01_T04	2018-08-21 12:4...	1	NULL	25,12342665425...
77930	3	IDTest1	01_T01	2018-08-21 12:4...	2	NULL	25,22056262648...
77931	4884377	IDTest1	01_T02	2018-08-21 12:4...	2	NULL	25,11069154291...
77932	4885939	IDTest1	01_T03	2018-08-21 12:4...	2	NULL	24,98689226893...
77933	4887501	IDTest1	01_T04	2018-08-21 12:4...	2	NULL	25,25434080061...
77934	4	IDTest1	01_T01	2018-08-21 12:4...	3	NULL	25,24255018159...
77935	4884378	IDTest1	01_T02	2018-08-21 12:4...	3	NULL	25,13066975573...
77936	4885940	IDTest1	01_T03	2018-08-21 12:4...	3	NULL	25,01487176520...
77937	4887502	IDTest1	01_T04	2018-08-21 12:4...	3	NULL	25,25933753675...
77938	5	IDTest1	01_T01	2018-08-21 12:4...	4	NULL	25,25954057841...
77939	4884379	IDTest1	01_T02	2018-08-21 12:4...	4	NULL	25,17162514688...
77940	4885941	IDTest1	01_T03	2018-08-21 12:4...	4	NULL	25,06683377072...
77941	4887503	IDTest1	01_T04	2018-08-21 12:4...	4	NULL	25,27432775091...
77942	6	IDTest1	01_T01	2018-08-21 12:4...	5	NULL	25,10262970842...
77943	4884380	IDTest1	01_T02	2018-08-21 12:4...	5	NULL	25,04676137984...
77944	4885942	IDTest1	01_T03	2018-08-21 12:4...	5	NULL	24,87097754983...
77945	4887504	IDTest1	01_T04	2018-08-21 12:4...	5	NULL	25,12842336535...
77946	7	IDTest1	01_T01	2018-08-21 12:4...	6	NULL	25,10262970842...
77947	4884381	IDTest1	01_T02	2018-08-21 12:4...	6	NULL	25,02078980243...
77948	4885943	IDTest1	01_T03	2018-08-21 12:4...	6	NULL	24,86198418914...
77949	4887505	IDTest1	01_T04	2018-08-21 12:4...	6	NULL	25,09844311308...

Figura 39: Tabla CorrectedMeasures de las sondas para el ensayo de frío.

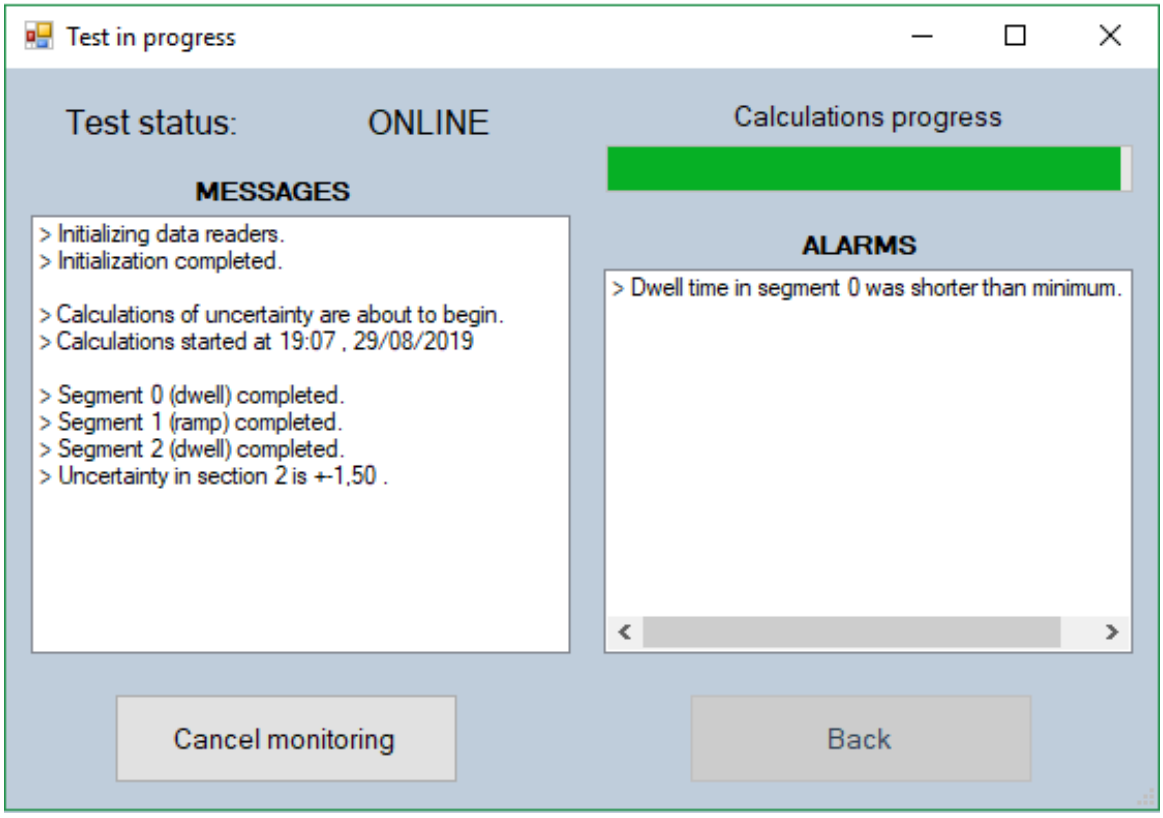

Figura 40: Ventana de monitorización al finalizar el tercer segmento.

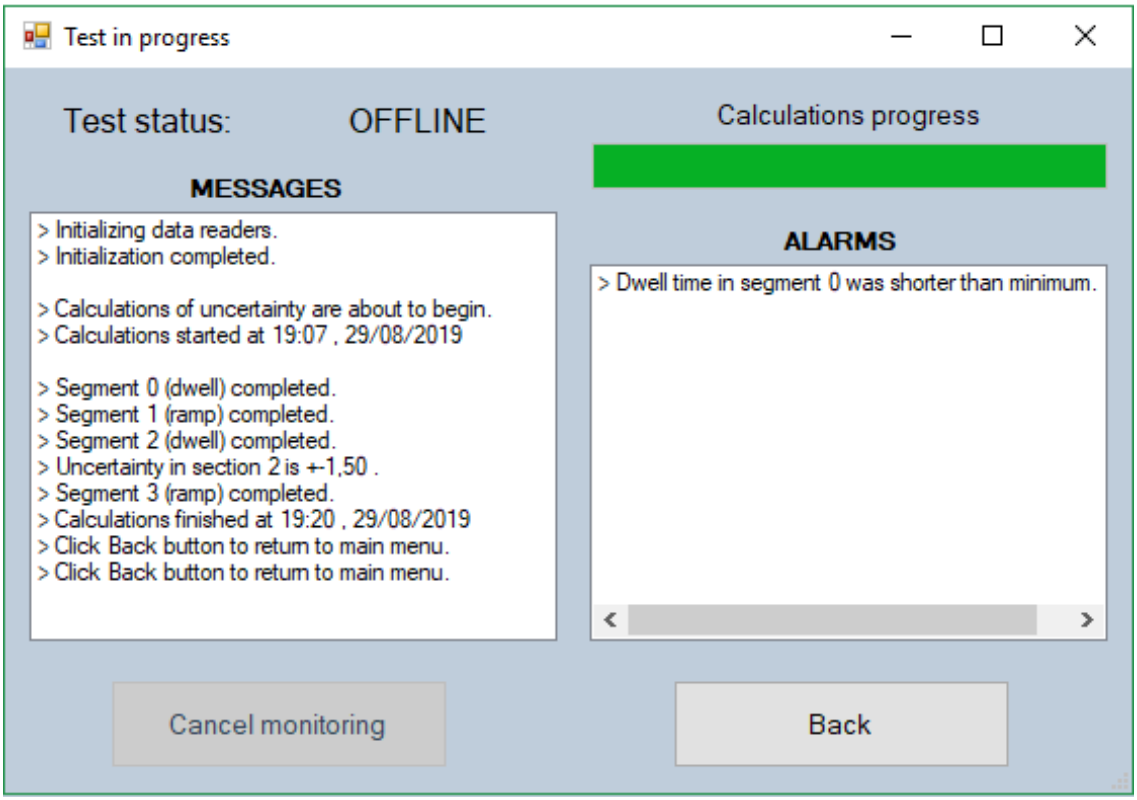

Figura 41: Ventana de monitorización al completar el análisis.

DESKTOP-SB3H8F0\S...- dbo.Uncertainty | DESKTOP-SB3H8F0\...orrectedMeasures

	IdUncertainty	IdTest	SectionDescrip...	NumberOfSec...	ReferenceTem...	ExpandedUnce...	ReferenceHu...	ExpandedUnce...
▶	35	IDTest1	Short dwell peri...	0	NULL	NULL	NULL	NULL
	36	IDTest1	Ramp period	1	NULL	NULL	NULL	NULL
	37	IDTest1	Normal dwell p...	2	-5,39511932367...	1,502745770586...	NULL	NULL
	38	IDTest1	Ramp period	3	NULL	NULL	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 42: Registros al finalizar el análisis.

6 CONCLUSIONES Y LÍNEAS FUTURAS

Este último capítulo recopila los objetivos cumplidos en el alcance, las conclusiones que se han desarrollado al respecto y las líneas futuras que surgen a partir del trabajo realizado. Es imprescindible tener el cuenta antes de concluir el contexto en el que se encuentra este proyecto: el de la automatización como forma de mejorar el rendimiento y los servicios prestados.

6.1. Conclusiones

El objetivo principal ha sido construir una aplicación que pudiese supervisar de manera remota y en vivo un ensayo climático en las cámaras propiedad de ATN. Como demuestra este documento, se ha conseguido en su mayor parte. Sin embargo, hay que mencionar que la aplicación ha sido probada en condiciones de simulación de ensayo, no en un ensayo real *online.* Como ya se aclaró en un capítulo anterior, la razón para ello es que, aunque este trabajo forme parte de un proyecto mayor, la infraestructura necesaria para probar la aplicación en condiciones reales no estaba desarrollada. Así pues, se ha implementado una base de datos propia con valores reales para comprobar que el funcionamiento es correcto, concluyendo así que el sistema está dotado de todo lo necesario para utilizarse en situaciones realistas.

La interfaz resulta sencilla para el usuario, contando con mensajes de error y ventanas intermedias que aclaran su manejo. El hecho de que todo esté recogido en pantalla ayuda a que el usuario pueda comprobar en cualquier momento que el proceso marcha convenientemente.

Para valores iguales de las muestras – los del ensayo de frío de ejemplo –, se ha comprobado que el resultado de la incertidumbre expandida es el mismo tanto para la macro de Excel que se usa para el cálculo manual como para la aplicación software.

En conclusión, el alcance propuesto inicialmente ha sido cumplido en su mayor parte, solo sería necesario realizar una prueba sobre una cámara real con la infraestructura adecuada para verificar completamente el comportamiento del sistema.

6.2. Líneas futuras

Aunque las bases ya están planteadas, las posibles ampliaciones de este proyecto son a todas luces innumerables.

En primer lugar, la base de datos SQL podría ser migrada a una gestionada por MongoDB, pues es más escalable y fácil de mantener cuando el tamaño del conjunto de datos se hace enorme.

Además, resultaría de ayuda que se guardase un registro de los mensajes de la aplicación, ya sea en un archivo de texto plano o en otro formato, para su recuperación en el caso de que fuese necesario.

En el futuro, podría ser de utilidad que, además de almacenar los datos en la nube, se generase un informe automáticamente con las características y el análisis completo del ensayo al finalizar. Esta funcionalidad ya está siendo implementada por ATN en otros procesos.

Una mejora recomendable también sería la inclusión de una gráfica en la ventana de monitorización del ensayo, donde el técnico observaría las medidas que va recogiendo el software, así como el rango de la

incertidumbre de cada tramo.

No puede faltar la implementación de otros tipos de sondas, ya sean de h.r., radiación, etc.

En definitiva, este proyecto abre un abanico de posibilidades de mejora que, sin duda alguna, gracias a estar enfocado en torno a *Virtual Lab*, permitirá un incremento sustancial del rendimiento en los ensayos que se realizan en Alter Technology TÜV NORD.

REFERENCIAS

[1] S. Addelman, G. E. P. Box, W. G. Hunter y J. S. Hunter, «Statistics for Experimenters,» *Technometrics,* 1979.

[2] J. C. F. G. I. M. Jcgm, «Evaluation of measurement data — Guide to the expression of uncertainty in measurement,» *International Organization for Standardization Geneva ISBN,* 2008.

[3] T. J. Quinn, «The international temperature scale of 1990 (ITS-90),» *Physica Scripta,* 1990.

[4] A. L. Silberschatz, B. Sudarshan, «Fundamentos de bases de datos», 2002.

[5] WMO, «International Meteorological Vocabulary,» *WMO Technical Publication No. 182,* p. 784, 1992.

[6] E. C.-o. f. Accreditation, «EA 4 / 02 Expression of the Uncertainty of Measurement in Calibration,» *Measurement,* 1999.

[7] A. Troelsen y P. Japikse, Pro C# 7, 2017.

[8] R. Vystavěl, C# Programming for Absolute Beginners, 2017.

[9] Asociación Española de Normalización y Certificación, «Ensayos ambientales. Parte 2-1: Ensayos. Ensayo A: Frío. UNE-EN 60068-2-1,» 2007.

[10] Asociación Española de Normalización y Certificación, «Ensayos ambientales. Parte 2-2: Ensayos. Ensayo B: Calor seco,» 2008.

[11] Asociación Española de Normalización y Certificación, «Ensayos ambientales. Parte 2: Ensayos. Ensayo Z/AD: Ensayo cíclico compuesto de temperatura y humedad,» 2000.

[12] Asociación Española de Normalización y Certificación, «Ensayos ambientales. Parte 2-30: Ensayos. Ensayo Db: Ensayo cíclico de calor húmedo (ciclo de 12 h + 12 h),» 2006.

[13] Asociación Española de Normalización y Certificación, «Ensayos ambientales. Parte 2-14: Ensayos. Ensayo N: Variación de la temperatura.,» 2011.

[14] Asociación Española de Normalización y Certificación, «Ensayos ambientales. Parte 3-11: Documentación de acompañamiento y guía. Cálculo de la incertidumbre de las condiciones en cámaras de ensayos climáticos,» 2008.

[15] Asociación Española de Normalización y Certificación, «Ensayos ambientales. Parte 3-1: Información básica. Ensayos de frío y calor seco.,» 2012.

ANEXO

Data Readers

ITempReader

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.
7.  namespace VirtualLab_TestBinding_v2.DataReader.Interfaces
8.  {
9.    interface ITempReader
10.   {
11.       TempData Get(int listIndex);
12.       bool TestFinished();
13.       int GetNumberOfSamples();
14.   }
15. }
```

TempData

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.
7.  namespace VirtualLab_TestBinding_v2.DataReader
8.  {
9.      public class TempData
10.     {
11.         public DateTime Date { get; set; }
12.         public double Temp { get; set; }
13.         public int Order { get; set; }
14.         public int IdPrimaryMeasures { get; internal set; }
15.     }
16. }
```

TempReader

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.  using VirtualLab_TestBinding_v2.DataReader.Interfaces;
7.  using AutoMapper;
8.  using AutoMapper.EntityFramework;
9.  using System.ComponentModel;
10. using System.Data;
11. using System.Drawing;
```

```
12. using System.Windows.Forms;
13. using VirtualLab_TestBinding_v2.Data.Infrastructure;
14. using VirtualLab_TestBinding_v2.Data.Repositories;
15. using VirtualLab_TestBinding_v2.DataReader;
16. using VirtualLab_TestBinding_v2.Domain.Models;
17. using VirtualLab_TestBinding_v2.Infrastructure;
18. using VirtualLab_TestBinding_v2.Services;
19. using VirtualLab_TestBinding_v2.ViewModel;
20. using System.Threading;
21. using VirtualLab_TestBinding_v2.Forms;
22.
23. namespace VirtualLab_TestBinding_v2.DataReader
24. {
25. public class TempReader : ITempReader
26. {
27.     private readonly string iDTest;
28.     private readonly int iDProbe;
29.     private List<VMPrimaryMeasures> listVMPrimaryMeasures = new List<VMPrimaryMeasures>();
30.     private DateTime date = DateTime.Now.AddMilliseconds(-10000); // To make sure that it works
31.     private int index = 1540;
32.
33.     public TempReader(string idTest, int idProbe)
34.     {
35.         iDTest = idTest;
36.         iDProbe = idProbe;
37.         InitializeDataReader();
38.     }
39.
40.     private void InitializeDataReader()
41.     {
42.         IUnitOfWork unitOfWork = CompositionRoot.Resolve<IUnitOfWork>();
43.         IMapper mapper = CompositionRoot.Resolve<IMapper>();
44.         IPrimaryMeasuresService primaryMeasuresService =
            CompositionRoot.Resolve<IPrimaryMeasuresService>();
45.         IProbeService probeService = CompositionRoot.Resolve<IProbeService>();
46.
47.         List<Probe> listProbe = probeService.GetAll().ToList();
48.         Probe probeCode = listProbe.FirstOrDefault(u => u.IdProbe == iDProbe);
49.
50.         // listPrimaryMeasures is the list of registers that match with the IDTest and the ProbeCode.
51.         List<PrimaryMeasures> listPrimaryMeasures = primaryMeasuresService.GetAll().ToList();
52.         listPrimaryMeasures = listPrimaryMeasures.Where(u => u.IdTest == iDTest && probeCode.Code ==
            u.ProbeCode).ToList();
53.
54.         listVMPrimaryMeasures = mapper.Map<List<PrimaryMeasures>,
            List<VMPrimaryMeasures>>(listPrimaryMeasures);
55.     }
56.
57.     public TempData Get(int listIndex)
58.     {
59.         // Online simulation
60.         if (listIndex >= 1540)
61.         {
62.             if((DateTime.Now - date).TotalMilliseconds > 1000)
63.             {
64.                 date = DateTime.Now;
65.                 index = listIndex;
66.             }
67.             else
68.             {
69.                 listIndex = index;
70.             }
71.         }
72.         if (listIndex >= listVMPrimaryMeasures.Count - 1)
73.         {
74.             listIndex = listVMPrimaryMeasures.Count - 1;
75.         }
76.         TempData tempData = new TempData
77.         {
78.             Order = (int)listVMPrimaryMeasures[listIndex].MeasureIndex,
79.             Date = (DateTime)listVMPrimaryMeasures[listIndex].Date,
80.             Temp = (double)listVMPrimaryMeasures[listIndex].Measure,
81.             IdPrimaryMeasures = listVMPrimaryMeasures[listIndex].IdPrimaryMeasures
82.         };
83.
```

```
84.          return tempData;
85.      }
86.
87.      public TempData Get(DateTime date)
88.      {
89.          TempData tempData = new TempData();
90.          List<VMPrimaryMeasures> aux = listVMPrimaryMeasures.Where(u => ((DateTime)u.Date -
             date).TotalSeconds >= 0).ToList();
91.          if (aux != null)
92.          {
93.              int listIndex = (int)aux[0].MeasureIndex - 1;
94.              tempData.Order = (int)listVMPrimaryMeasures[listIndex].MeasureIndex;
95.              tempData.Date = (DateTime)listVMPrimaryMeasures[listIndex].Date;
96.              tempData.Temp = (double)listVMPrimaryMeasures[listIndex].Measure;
97.              tempData.IdPrimaryMeasures = listVMPrimaryMeasures[listIndex].IdPrimaryMeasures;
98.          }
99.          else
100.         {
101.             tempData.Order = (int)listVMPrimaryMeasures[listVMPrimaryMeasures.Count - 1]
                 .MeasureIndex;
102.             tempData.Date = (DateTime)listVMPrimaryMeasures[listVMPrimaryMeasures.Count - 1].Date;
103.             tempData.Temp = (double)listVMPrimaryMeasures[listVMPrimaryMeasures.Count - 1].Measure;
104.             tempData.IdPrimaryMeasures = listVMPrimaryMeasures[listVMPrimaryMeasures.Count -
                 1].IdPrimaryMeasures;
105.         }
106.
107.         return tempData;
108.     }
109.
110.     public int GetNumberOfSamples()
111.     {
112.          return index;
113.     }
114.
115.     public bool TestFinished()
116.     {
117.         bool aux = false;
118.         if (index >= listVMPrimaryMeasures.Count - 1)
119.         {
120.             aux = true;
121.         }
122.         return aux;
123.     }
124. }
125. }
```

Formularios

Form1

```
1.   using AutoMapper;
2.   using AutoMapper.EntityFramework;
3.   using System;
4.   using System.Collections.Generic;
5.   using System.ComponentModel;
6.   using System.Data;
7.   using System.Drawing;
8.   using System.Linq;
9.   using System.Text;
10.  using System.Threading.Tasks;
11.  using System.Windows.Forms;
12.  using VirtualLab_TestBinding_v2.Data.Infrastructure;
13.  using VirtualLab_TestBinding_v2.Data.Repositories;
14.  using VirtualLab_TestBinding_v2.DataReader;
15.  using VirtualLab_TestBinding_v2.DataReader.Interfaces;
16.  using VirtualLab_TestBinding_v2.Domain.Models;
17.  using VirtualLab_TestBinding_v2.Infrastructure;
18.  using VirtualLab_TestBinding_v2.Services;
19.  using VirtualLab_TestBinding_v2.ViewModel;
20.  using System.Threading;
```

```
21.  using VirtualLab_TestBinding_v2.Forms;
22.
23.  namespace VirtualLab_TestBinding_v2
24.  {
25.     public partial class Form1 : Form
26.     {
27.         #region Fields
28.         // View Model fields
29.         private int aux; // Auxiliary variable to iterate
30.         private int iDBatchIDTestChamberSelected;
31.         private int iDProgramTestSelected = -1;
32.         private int iDChamberSelected = -1;
33.         private int numberOfCyclesToComplete = 0;
34.         List<BatchIDTestChamber> listBatchIDTestChamber;
35.         List<VMBatchIDTestChamber> listVMBatchIDTestChamber;
36.         List<Chamber> listChamber;
37.         List<VMChamber> listVMChamber;
38.         List<Alarm> listAlarm;
39.         List<VMAlarm> listVMAlarm;
40.         List<ProgramTest> listProgramTest;
41.         List<VMProgramTest> listVMProgramTest;
42.         List<ProgramTest_Alarm> listProgramTest_Alarm;
43.         List<VMProgramTest_Alarm> listVMProgramTest_Alarm;
44.         List<Probe> listProbe;
45.         List<VMProbe> listVMProbe;
46.         List<Chamber_Probe> listChamber_Probe;
47.         List<VMChamber_Probe> listVMChamber_Probe;
48.         List<ChamberCalibration> listChamberCalibration;
49.         List<VMChamberCalibration> listVMChamberCalibration;
50.         List<ProbeCalibration> listProbeCalibration;
51.         List<VMProbeCalibration> listVMProbeCalibration;
52.
53.         List<ObjCheckedListBox> itemsAlarmBox = new List<ObjCheckedListBox>();
54.         List<ObjCheckedListBox> itemsProbeBox = new List<ObjCheckedListBox>();
55.         #endregion
56.
57.         internal class ObjComboBox
58.         {
59.             public string Text { get; set; }
60.             public int Value { get; set; }
61.         }
62.
63.         internal class ObjCheckedListBox
64.         {
65.             public int Index { get; set; }
66.             public string Text { get; set; }
67.             public int ID { get; set; }
68.         }
69.
70.         #region Dependency injection and mapper
71.         // Initialize UnitOfWork
72.         IUnitOfWork unitOfWork = CompositionRoot.Resolve<IUnitOfWork>();
73.         // Initialize Mapper
74.         IMapper mapper = CompositionRoot.Resolve<IMapper>();
75.         #endregion
76.
77.         public Form1()
78.         {
79.             InitializeComponent();
80.             InitializeForm();
81.         }
82.
83.         public void InitializeForm()
84.         /* It initializes the user window with the data catched from the database. */
85.         {
86.             IAlarmService alarmService = CompositionRoot.Resolve<IAlarmService>();
87.             IBatchIDTestChamberService batchIDTestChamberService =
                CompositionRoot.Resolve<IBatchIDTestChamberService>();
88.             IChamberService chamberService = CompositionRoot.Resolve<IChamberService>();
89.             IChamber_ProbeService chamber_ProbeService =
                CompositionRoot.Resolve<IChamber_ProbeService>();
90.             IProbeService probeService = CompositionRoot.Resolve<IProbeService>();
91.             IProgramTest_AlarmService programTest_AlarmService =
                CompositionRoot.Resolve<IProgramTest_AlarmService>();
92.             IProgramTestService programTestService = CompositionRoot.Resolve<IProgramTestService>();
```

```
93.
94.             #region Chamber
95.             listChamber = chamberService.GetAll().ToList();
96.             listVMChamber = mapper.Map<List<Chamber>, List<VMChamber>>(listChamber);
97.             ChamberBox.DisplayMember = "Text";
98.             ChamberBox.ValueMember = "Value";
99.             foreach (VMChamber register in listVMChamber)
100.            {
101.                    ChamberBox.Items.Add(new ObjComboBox{ Text = register.Code + " " +
                        register.Family, Value = register.IdChamber });
102.            }
103.            #endregion
104.
105.            #region IDTest and Batch
106.            // 1) Get IDTest and Batch field of every register in database.
107.            listBatchIDTestChamber = batchIDTestChamberService.GetAll().ToList();
108.            // 2) Mapping
109.            listVMBatchIDTestChamber = mapper.Map<List<BatchIDTestChamber>,
                List<VMBatchIDTestChamber>>(listBatchIDTestChamber);
110.            // 3) Get IDTest and Batch fields and add them to IDTestBox and to BatchBox.
111.            IDTestBox.DisplayMember = "Text";
112.            IDTestBox.ValueMember = "Value";
113.            BatchBox.DisplayMember = "Text";
114.            BatchBox.ValueMember = "Value";
115.            foreach (VMBatchIDTestChamber register in listVMBatchIDTestChamber)
116.            {
117.                    // There is only one table for Batches and IDTests so the primary key is the same
                        for both of them.
118.                    IDTestBox.Items.Add(new ObjComboBox { Text = register.IdTest + " (Chamber " +
                        listVMChamber.First(u => u.IdChamber == register.IdChamber).Code + ")", Value =
                        register.IdBatchIDTestChamber });
119.                    BatchBox.Items.Add(new ObjComboBox { Text = register.Batch /*+ " (Chamber " +
                        listVMChamber.First(u => u.IdChamber == register.IdChamber).Code + ")"*/, Value =
                        register.IdBatchIDTestChamber });
120.            }
121.            #endregion
122.
123.            #region Program Test
124.            listProgramTest = programTestService.GetAll().ToList();
125.            listVMProgramTest = mapper.Map<List<ProgramTest>, List<VMProgramTest>>(listProgramTest);
126.            ProgramTestBox.DisplayMember = "Text";
127.            ProgramTestBox.ValueMember = "Value";
128.            foreach (VMProgramTest register in listVMProgramTest)
129.            {
130.                    ProgramTestBox.Items.Add(new ObjComboBox{ Text = register.ProgramNameApp, Value =
                        register.IdProgramTest });
131.            }
132.            #endregion
133.
134.            #region Alarm
135.            listAlarm = alarmService.GetAll().ToList();
136.            listVMAlarm = mapper.Map<List<Alarm>, List<VMAlarm>>(listAlarm);
137.            aux = 0;
138.            foreach (VMAlarm register in listVMAlarm)
139.            {
140.                    AlarmCheckedListBox.Items.Insert(aux, register.Code + " (" + register.Description
                        + ")");
141.                    itemsAlarmBox.Add(new ObjCheckedListBox { Index = aux, Text = register.Code + "
                        (" + register.Description + ")", ID = register.IdAlarm });
142.                            aux++;
143.            }
144.            aux = 0;
145.            #endregion
146.
147.            #region Probe
148.            listProbe = probeService.GetAll().ToList();
149.            listVMProbe = mapper.Map<List<Probe>, List<VMProbe>>(listProbe);
150.            aux = 0;
151.            foreach (VMProbe register in listVMProbe)
152.            {
153.                    ProbeCheckedListBox.Items.Insert(aux,register.Code + " " +
                        register.Family);
154.                    itemsProbeBox.Add(new ObjCheckedListBox { Index = aux, Text =
                        register.Code + " " + register.Family, ID = register.IdProbe});
155.                    aux++;
```

```
156.            }
157.            aux = 0;
158.            #endregion
159.
160.            #region Rest of tables
170.            listChamber_Probe = chamber_ProbeService.GetAll().ToList();
180.            listVMChamber_Probe = mapper.Map<List<Chamber_Probe>,
                List<VMChamber_Probe>>(listChamber_Probe);
181.            listProgramTest_Alarm = programTest_AlarmService.GetAll().ToList();
182.            listVMProgramTest_Alarm = mapper.Map<List<ProgramTest_Alarm>,
                List<VMProgramTest_Alarm>>(listProgramTest_Alarm);
183.            #endregion
184.
185.        }
186.
187.        private void IDTestBox_SelectedIndexChanged(object sender, EventArgs e)
188.        /* It detects what IDTest is selected and fill the rest of
189.        * the form (except ProgramTestBox and AlarmCheckListBox). */
190.        {
191.            // Get what IDTest is selecting to execute data readers later.
192.            if ((IDTestBox.SelectedItem as ObjComboBox).Text != null)
193.            {
194.                    // 1) Get what IDTest is selected. Then take related IDBatch and IdChamber from
                        database and put it onto the form.
195.                    iDBatchIDTestChamberSelected = (IDTestBox.SelectedItem as ObjComboBox).Value;
196.                    var registerSelectedAux1 = listVMBatchIDTestChamber.FirstOrDefault(u =>
                        u.IdBatchIDTestChamber == iDBatchIDTestChamberSelected);
197.                    // BatchBox modified.
198.                    BatchBox.SelectedIndex = BatchBox.FindString(registerSelectedAux1.Batch);
199.                    // ChamberBox modified.
200.                    iDChamberSelected = (int)registerSelectedAux1.IdChamber;
201.                    var registerSelectedAux2 = listVMChamber.FirstOrDefault(u => u.IdChamber ==
                        iDChamberSelected);
202.
203.                    // 2) Fill probes from chamber selected.
204.                    ChamberBox.SelectedIndex = ChamberBox.FindString(registerSelectedAux2.Code);
205.                    var registerSelectedAux3 = listVMChamber_Probe.Where( u => (int)u.IdChamber ==
                        iDChamberSelected);
206.                    // Clear list box.
207.                    for (int i = 0; i < ProbeCheckedListBox.Items.Count ; i++)
208.                    {
209.                            ProbeCheckedListBox.SetItemChecked(i, false);
210.                    }
211.                    foreach (VMChamber_Probe register in registerSelectedAux3)
212.                    {
213.                            var registerSelectedAux4 = itemsProbeBox.FirstOrDefault(u => u.ID ==
                            register.IdProbe);
214.                            ProbeCheckedListBox.SetItemChecked(registerSelectedAux4.Index,true);
215.                    }
216.            }
217.        }
218.
219.        private void NumberOfCyclesNumericUpDown_ValueChanged(object sender, EventArgs e)
220.        {
221.            numberOfCyclesToComplete = (int)NumberOfCyclesNumericUpDown.Value;
222.        }
223.
224.        private void ProbesSelectAllButton_Click(object sender, EventArgs e)
225.        {
226.            for (int i = 0; i < ProbeCheckedListBox.Items.Count; i++)
227.            {
228.                    ProbeCheckedListBox.SetItemChecked(i, true);
229.            }
230.        }
231.
232.        private void ProbesClearAllButton_Click(object sender, EventArgs e)
233.        {
234.            for (int i = 0; i < ProbeCheckedListBox.Items.Count; i++)
235.            {
236.                    ProbeCheckedListBox.SetItemChecked(i, false);
237.            }
238.        }
239.
240.        private void AlarmsSelectAllButton_Click(object sender, EventArgs e)
241.        {
```

```
242.            for (int i = 0; i < AlarmCheckedListBox.Items.Count; i++)
243.            {
244.                    AlarmCheckedListBox.SetItemChecked(i, true);
245.            }
246.        }
247.
248.        private void AlarmsClearAllButton_Click(object sender, EventArgs e)
249.        {
250.            for (int i = 0; i < AlarmCheckedListBox.Items.Count; i++)
251.            {
252.                    AlarmCheckedListBox.SetItemChecked(i, false);
253.            }
254.        }
255.
256.        private void ProgramTestBox_SelectedIndexChanged(object sender, EventArgs e)
257.        {
258.            // Fill alarms from program test selected.
259.            iDProgramTestSelected = (ProgramTestBox.SelectedItem as ObjComboBox).Value;
260.            var registerSelectedAux1 = listVMProgramTest_Alarm.Where(u => (int)u.IdProgramTest ==
                iDProgramTestSelected); // It will display an error if the result is null. Check this.
261.            // Clear list box.
262.            for (int i = 0; i < AlarmCheckedListBox.Items.Count; i++)
263.            {
264.                    AlarmCheckedListBox.SetItemChecked(i, false);
265.            }
266.            foreach (VMProgramTest_Alarm register in registerSelectedAux1)
267.            {
268.                    var registerSelectedAux2 = itemsAlarmBox.FirstOrDefault(u => u.ID ==
                        register.IdAlarm);
269.                    AlarmCheckedListBox.SetItemChecked(registerSelectedAux2.Index, true);
270.            }
271.            var registerSelectedAux3 = listVMProgramTest.FirstOrDefault(u => (int)u.IdProgramTest ==
                iDProgramTestSelected);
272.            NumberOfCyclesNumericUpDown.Value = (decimal)registerSelectedAux3.NumberOfCycles;
273.            ProgramPicture.Image = null;
274.            if (registerSelectedAux3.ImageRute != null)
275.            {
276.                    string imagePath = registerSelectedAux3.ImageRute;
277.                    Image image = Image.FromFile(@imagePath);
278.                    ProgramPicture.Image = image;
279.            }
280.        }
281.
282.        private void ViewSegmentsButton_Click(object sender, EventArgs e)
283.        /* This displays segments tab. */
284.        {
285.            string programNameApp;
286.            if (iDProgramTestSelected != -1)
287.            {
288.                    programNameApp = (listVMProgramTest.FirstOrDefault(u => (int)u.IdProgramTest ==
                        iDProgramTestSelected) as VMProgramTest).ProgramNameApp;
289.            }
290.            else programNameApp = "Program not selected";
291.            SegmentsForm segmentsForm = new SegmentsForm(iDProgramTestSelected,programNameApp);
292.            segmentsForm.ShowDialog();
293.        }
294.
295.        private void ChamberBox_SelectedIndexChanged(object sender, EventArgs e)
296.        {
297.            // Fill alarms from program test selected.
298.            iDChamberSelected = (ChamberBox.SelectedItem as ObjComboBox).Value;
299.            if (iDChamberSelected != -1)
300.            {
301.                    var registerSelectedAux1 = listVMChamber_Probe.Where(u => (int)u.IdChamber ==
                        iDChamberSelected);
302.                    // Clear list box.
303.                    for (int i = 0; i < ProbeCheckedListBox.Items.Count; i++)
304.                    {
305.                            ProbeCheckedListBox.SetItemChecked(i, false);
306.                    }
307.                    foreach (VMChamber_Probe register in registerSelectedAux1)
308.                    {
309.                            var registerSelectedAux2 = itemsProbeBox.FirstOrDefault(u => u.ID ==
                                register.IdProbe);
310.                            ProbeCheckedListBox.SetItemChecked(registerSelectedAux2.Index, true);
```

```
311.                    }
312.                }
313.            }
314.
315.            private void TestButton_Click(object sender, EventArgs e)
316.            /* First of all,this runs the WIP dialog, passes all
317.            * data needed to calculate uncertainty and save
318.            * avalaible data from the test. */
319.            {
320.                // Check if every combobox is filled.
321.                if (IDTestBox.Text != "" && BatchBox.Text != "" && ChamberBox.Text != "" &&
                    ProgramTestBox.Text != "")
322.                {
323.                        // Create TestInProgressForm
324.                        TestInProgressForm testInProgressForm = new TestInProgressForm
325.                        {
326.                                StartMonitoring = false
327.                        };
328.
329.                        #region Update data for monitoring
330.
331.                        ISegmentService segmentService = CompositionRoot.Resolve<ISegmentService>();
332.                        IChamberCalibrationService chamberCalibrationService =
                            CompositionRoot.Resolve<IChamberCalibrationService>();
333.                        IProbeCalibrationService probeCalibrationService =
                            CompositionRoot.Resolve<IProbeCalibrationService>();
334.
335.                        // Chamber Calibration
336.                        iDChamberSelected = (ChamberBox.SelectedItem as ObjComboBox).Value;
337.                        listChamberCalibration = chamberCalibrationService.GetAll().ToList();
338.                        listVMChamberCalibration = mapper.Map<List<ChamberCalibration>,
                            List<VMChamberCalibration>>(listChamberCalibration);
339.                        testInProgressForm.VMChamberCalibration = listVMChamberCalibration.First(u =>
                            u.IdChamber == iDChamberSelected);
340.
341.                        // Program Test
342.                        iDProgramTestSelected = (ProgramTestBox.SelectedItem as ObjComboBox).Value;
343.                        testInProgressForm.VMProgramTest = listVMProgramTest.First(u => u.IdProgramTest
                            == iDProgramTestSelected);
344.
345.                        // Probe Calibration
346.                        listProbeCalibration = probeCalibrationService.GetAll().ToList();
347.                        listVMProbeCalibration = mapper.Map<List<ProbeCalibration>,
                            List<VMProbeCalibration>>(listProbeCalibration);
348.                        aux = 0;
349.                        List<VMProbeCalibration> ListVMProbeCalibrationSelected = new
                            List<VMProbeCalibration>();
350.                                List<VMProbe> ListVMProbeSelected = new List<VMProbe>();
351.                        for (int i = 0; i < ProbeCheckedListBox.Items.Count; i++)
352.                        {
353.                                // Check if that item is selected (true)
354.                                if(ProbeCheckedListBox.GetItemChecked(i) == true)
355.                                {
356.                                var iDProbeSelectedAux = itemsProbeBox[i].ID;
357.                                var vMProbeCalibrationAux = listVMProbeCalibration.First(u => u.IdProbe
                                    == iDProbeSelectedAux);
358.                                var vMProbeAux = listVMProbe.First(u => u.IdProbe == iDProbeSelectedAux);
359.                                ListVMProbeCalibrationSelected.Add(vMProbeCalibrationAux);
360.                                ListVMProbeSelected.Add(vMProbeAux);
361.                                aux++;
362.                        }
363.                        // aux = 0 means that there is not any probe selected.
364.                }
365.                testInProgressForm.ListVMProbeCalibration = ListVMProbeCalibrationSelected;
366.                testInProgressForm.ListVMProbe = ListVMProbeSelected;
367.
368.                // Alarm
369.                List<VMAlarm> ListVMAlarmSelected = new List<VMAlarm>();
370.                for (int i = 0; i < AlarmCheckedListBox.Items.Count; i++)
371.                {
372.                        // Check if that item is selected (true)
373.                        if (AlarmCheckedListBox.GetItemChecked(i) == true)
374.                        {
375.                                var iDAlarmSelectedAux = itemsAlarmBox[i].ID;
376.                                var vMAlarmAux = listVMAlarm.First(u => u.IdAlarm == iDAlarmSelectedAux);
```

```
                            ListVMAlarmSelected.Add(vMAlarmAux);
                            aux++;
                    }
                    // aux = 0 means that there is not any probe selected.
            }
            testInProgressForm.ListVMAlarm = ListVMAlarmSelected;

            // Segment
            List<Segment> listSegment = segmentService.GetAll().ToList();
            List<VMSegment> listVMSegment = mapper.Map<List<Segment>, List<VMSegment>>(listSegment);
            testInProgressForm.ListVMSegment = listVMSegment.FindAll(u => u.IdProgramTest ==
            iDProgramTestSelected);

            // ID Test
            testInProgressForm.IDTestSelected = listVMBatchIDTestChamber.First(u => u.IdChamber ==
            iDChamberSelected).IdTest;

            testInProgressForm.StartMonitoring = true;

            #endregion

            testInProgressForm.ShowDialog();
            }
            else // If not all fields are filled.
            {
                    ErrorForm errorForm = new ErrorForm();
                    errorForm.ShowDialog();
            }
        }
    }
}
```

TestInProgress

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using VirtualLab_TestBinding_v2.ViewModel;
using VirtualLab_TestBinding_v2.DataReader;
using VirtualLab_TestBinding_v2.DataReader.Interfaces;
using AutoMapper;
using AutoMapper.EntityFramework;
using VirtualLab_TestBinding_v2.Data.Infrastructure;
using VirtualLab_TestBinding_v2.Data.Repositories;
using VirtualLab_TestBinding_v2.Domain.Models;
using VirtualLab_TestBinding_v2.Infrastructure;
using VirtualLab_TestBinding_v2.Services;
using System.Threading;
using VirtualLab_TestBinding_v2.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace VirtualLab_TestBinding_v2.Forms
{
    public partial class TestInProgressForm : Form
    {
        #region Properties
        public VMChamberCalibration VMChamberCalibration { get; set; }
        public VMProgramTest VMProgramTest { get; set; }
        public List<VMProbeCalibration> ListVMProbeCalibration { get; set; }
        public List<VMAlarm> ListVMAlarm { get; set; }
        public List<VMSegment> ListVMSegment { get; set; }
        public List<VMProbe> ListVMProbe { get; set; }
        public int NumberOfCompletedCycles { get; set; }
        public string IDTestSelected { get; set; }
        public bool MonitoringCanceled { get; set; }
        public bool StartMonitoring { get; set; }
        private CancelMonitoringForm cancelMonitoringForm;
```

```
        #endregion

        #region Fields
        private BackgroundWorker backgroundWorkerCalculations = new BackgroundWorker();
        private bool testFinished = false;
        private int visibleItems;
        private decimal expandedUncertainty;
        decimal maxTemp = -500;
        decimal minTemp = 500;
        private List<VMCorrectedMeasures> listVMCorrectedMeasures = new List<VMCorrectedMeasures>();
        private int mostSampledProbe = 0;
        #endregion

        internal class UncertaintyObject
        {
            // Inc.Típ. Unid Dist. Div. C.Sens. ui(t)   vi ui4/vi
            public decimal StandardUncertainty { get; set; }
            public decimal Divisor { get; set; }
            public decimal SensityCoefficient { get; set; }
            public decimal Uncertainty { get; set; }
            public decimal DegreesOfFreedom { get; set; }
            public decimal TotalUncertainty { get; set; }
        }

        public TestInProgressForm()
        {
            InitializeComponent();
            BackButton.Enabled = false;
            InitializeBackgroundWorker();
            backgroundWorkerCalculations.RunWorkerAsync();
        }

        private void InitializeBackgroundWorker()
        {
            backgroundWorkerCalculations.WorkerSupportsCancellation = true;
            backgroundWorkerCalculations.WorkerReportsProgress = true;
            backgroundWorkerCalculations.DoWork += BackgroundWorker_DoWork;
            backgroundWorkerCalculations.RunWorkerCompleted += BackgroundWorker_RunWorkerCompleted;
            backgroundWorkerCalculations.ProgressChanged += BackgroundWorkerCalculations_ProgressChanged;
        }

        private void BackgroundWorkerCalculations_ProgressChanged(object sender, ProgressChangedEventArgs
        e)
        {
            BackgroundWorker worker = sender as BackgroundWorker;

            switch (e.ProgressPercentage)
            {
                case 1: // Message list box
                    TestMessageListBox.Items.Add(e.UserState);
                    visibleItems = TestMessageListBox.ClientSize.Height / TestMessageListBox.ItemHeight;
                    TestMessageListBox.TopIndex = Math.Max(TestMessageListBox.Items.Count - visibleItems
                        + 1, 0);
                    break;
                case 2: // Alarm list box
                    AlarmsListBox.Items.Add(e.UserState);
                    visibleItems = AlarmsListBox.ClientSize.Height / AlarmsListBox.ItemHeight;
                    AlarmsListBox.TopIndex = Math.Max(AlarmsListBox.Items.Count - visibleItems + 1, 0);
                    break;
                case 3: // Cancel button
                    CancelMonitoringButton.Enabled = false;
                    BackButton.Enabled = true;
                    if (cancelMonitoringForm != null) cancelMonitoringForm.Close();
                    break;
                case 4: // Progress Bar
                    testProgressBar.Value = (int)Math.Truncate((e.UserState as
                        List<double>)[0]*100/(e.UserState as List<double>)[1]);
                    break;
                case 5: // Online - offline test
                    OnOffTestTextBox.Text = e.UserState as string;
                    break;
                default:
                    break;
            }
        }
```

```
        private void BackgroundWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
        {
            backgroundWorkerCalculations.CancelAsync();
            Close();
        }

        private void CancelMonitoringButton_Click(object sender, EventArgs e)
        {
            cancelMonitoringForm = new CancelMonitoringForm(this);
            cancelMonitoringForm.ShowDialog();
            // Check if user has clicked Yes on cancel window.
            if (MonitoringCanceled == true)
            {
                backgroundWorkerCalculations.CancelAsync();
                // Close this dialog
                Close();
            }
        }

        private void WaitMilliseconds(int delay)
        /* Delay of the number of milliseconds passed as argument. */
        {
            DateTime time = new DateTime();
            time = DateTime.Now;
            while ((DateTime.Now - time).TotalMilliseconds < delay) { }
        }

        private void BackButton_Click(object sender, EventArgs e)
        {
            if (testFinished == true)
            {
                backgroundWorkerCalculations.CancelAsync();
                Close();
            }
        }

        private void CheckAlarms(string code, List<decimal> listAverTemp, decimal expandedUncertainty,
        int? numberOfSection, object sender)
        /* Check alarms after calculating expanded uncertainty. */
        {
            BackgroundWorker worker = sender as BackgroundWorker;

            // Case where alarms will be checked
            switch (code)
            {
                case "A01": // Temperature out of range
                    foreach (decimal temp in listAverTemp)
                    {
                        if ((temp + expandedUncertainty < maxTemp +
                        (decimal)VMProgramTest.DwellLowerToleranceMaxTemp &&
                        temp + expandedUncertainty > maxTemp -
                        (decimal)VMProgramTest.DwellUpperToleranceMaxTemp) ||
                        (temp + expandedUncertainty < minTemp +
                        (decimal)VMProgramTest.DwellLowerToleranceMaxTemp &&
                        temp + expandedUncertainty > minTemp -
                        (decimal)VMProgramTest.DwellUpperToleranceMaxTemp))
                        {
                        }
                        else worker.ReportProgress(2, "> Average temperature " + temp.ToString("0.00") +
                        " +- uncertainty " + expandedUncertainty.ToString("0.00") + " out of range at
                        section " + numberOfSection + ".");
                    }
                    break;
                case "A02": // Humidity out of range

                    break;
                default:
                    break;
            }
        }

        private void CheckAlarms(string code, decimal averageTemp, object sender)
        /* Check alarms after getting any sample. */
        {
```

```
            BackgroundWorker worker = sender as BackgroundWorker;

            switch (code)
            {
                case "A01": // Temperature out of range
                    if(averageTemp >= maxTemp + (decimal)VMProgramTest.DwellLowerToleranceMaxTemp ||
                        averageTemp <= minTemp - (decimal)VMProgramTest.DwellUpperToleranceMinTemp)
                    {
                        worker.ReportProgress(2, "> Average temperature of " +
                        averageTemp.ToString("0.000") + " out of range." );
                    }
                    break;
                case "A02": // Humidity out of range

                    break;
                default:
                    break;
            }

        }


private void AddToDBContext(IUnitOfWork unitOfWork, IMapper mapper, int[] iDProbesSelected, int i,
TempData tempSampleCorrected)
        /* Add a VMCorrectedMeasure to DbContext*/
        {
            VMCorrectedMeasures vMCorrectedMeasures = new VMCorrectedMeasures
            {
                IdPrimaryMeasures = tempSampleCorrected.IdPrimaryMeasures,
                IdTest = IDTestSelected,
                ProbeCode = ListVMProbe.First(u => u.IdProbe == iDProbesSelected[i]).Code,
                Date = tempSampleCorrected.Date,
                MeasureIndex = tempSampleCorrected.Order,
                CorrectedTemp = tempSampleCorrected.Temp,
                CorrectedHum = null
            };
            unitOfWork.DbContext.CorrectedMeasures.Persist(mapper).InsertOrUpdate(vMCorrectedMeasures);
            listVMCorrectedMeasures.Add(vMCorrectedMeasures);
        }

        private void IsTestFinished(int[] iDProbesSelected, List<TempReader> tempReaders, int[]
        numberOfSamples)
        /* It evaluates if test has finished and changes testFinished as needed. */
        {
            int aux = 0;
            for (int i = 0; i < iDProbesSelected.Length; i++)
            {
                // If number of samples is bigger or equal than total samples and that probe has
                finished.
                if (numberOfSamples[i] >= tempReaders[i].GetNumberOfSamples() &&
                tempReaders[i].TestFinished() == true)
                    aux++;
            }

            if (aux >= iDProbesSelected.Length) testFinished = true;
            else testFinished = false;
        }

        private bool IsTestOnline(int[] iDProbesSelected, List<TempReader> tempReaders, int[]
        numberOfSamples)
        /* It evaluates if test has finished and changes testFinished as needed. */
        {
            int aux = 0;
            bool aux1 = false;
            for (int i = 0; i < iDProbesSelected.Length; i++)
            {
                // If number of samples is bigger or equal than total samples and that probe has
                finished.
                if (tempReaders[i].TestFinished() == true)
                    aux++;
            }

            if (aux >= iDProbesSelected.Length) aux1 = false;
            else aux1 = true;
```

```
            return aux1;
        }

        private static bool AllSamplesAreTheSame(bool[] sameLastSample)
        /* It checks all data readers has read the same sample. */
        {
            int k = 0;
            bool result = false;
            foreach (bool lastSample in sameLastSample)
            {
                if (lastSample == true)
                {
                    k++;
                }
            }
            if (k >= sameLastSample.Length) result = true;
            else result = false;

            return result;
        }

        private void BackgroundWorker_DoWork(object sender, DoWorkEventArgs e)
        /* BackgroundWorker's goal is to calculate uncertainties and to monitor tests.*/
        {
            // Wait until initialization is done.
            while (StartMonitoring == false) { }

            #region 1) Initialization of samples, data readers and important variables.
            IUnitOfWork unitOfWork = CompositionRoot.Resolve<IUnitOfWork>();
            IMapper mapper = CompositionRoot.Resolve<IMapper>();

            if (backgroundWorkerCalculations.CancellationPending == true)
            {
                unitOfWork.Reset();
                e.Cancel = true;
                return;
            }

            ICorrectedMeasuresService correctedMeasuresService =
                CompositionRoot.Resolve<ICorrectedMeasuresService>();
            IUncertaintyService uncertaintyService = CompositionRoot.Resolve<IUncertaintyService>();

            BackgroundWorker worker = sender as BackgroundWorker;

            worker.ReportProgress(1, "> Initializing data readers.");

            // Initialize which probes will be read.
            int[] iDProbesSelected = new int[ListVMProbeCalibration.Count];
            // Auxiliary lists
            List<List<TempData>> probeSamples = new List<List<TempData>>(); // To save probe samples
            List<List<TempData>> probeSamplesCorrected = new List<List<TempData>>(); // To save corrected
                probe samples
            List<TempData> lastSampleRead = new List<TempData>();
            int i = 0;
            foreach (VMProbeCalibration register in ListVMProbeCalibration)
            {
                iDProbesSelected[i] = (int)register.IdProbe;
                probeSamples.Add(new List<TempData>());
                probeSamplesCorrected.Add(new List<TempData>());
                lastSampleRead.Add(new TempData());
                i++;

            }


            // Declare datareaders to get samples
            List<TempReader> tempReaders = new List<TempReader>();
            for (i = 0; i < iDProbesSelected.Length; i++)
            {
                tempReaders.Add(new TempReader(IDTestSelected, iDProbesSelected[i]));
            }

            // Maximum and minimum temperatures
            foreach (VMSegment segment in ListVMSegment)
            {
```

```
            if ((decimal)segment.Temperature > maxTemp) { maxTemp = (decimal)segment.Temperature; }
            if ((decimal)segment.Temperature < minTemp) { minTemp = (decimal)segment.Temperature; }
        }

        // Temperature ranges and other variables
        decimal topBeginDwellPeriod = maxTemp - (Math.Abs(maxTemp - minTemp) * 0.02m); // 2% (98%)
        decimal bottomBeginDwellPeriod = minTemp + (Math.Abs(maxTemp - minTemp) * 0.02m); // 2% (98%)
        bool lastPeriod = false; // Last period (dwell = true, ramp = false)
        bool currentPeriod = false; // This shows if test is in dwell period (true) or if is in ramp
            period (false)
        bool[] sameLastSample = new bool[ListVMProbeCalibration.Count]; // It determines wheter last
            and current samples of probe X are equal.
        int[] numberOfSamples = new int[ListVMProbeCalibration.Count]; // Number of read samples of
            each probe.
        bool calculateUncertainty = false;
        int? numberOfSection = 0;
        List<decimal> listAverageTemp = new List<decimal>();

        #region Which corrected measures are already in database?
        // Pick up which corrected measures have been saved. Then it initializes numberOfSamples,
            sameLastSample, lastSampleRead, numberOfSection, currentPeriod, lastPeriod
        List<CorrectedMeasures> listCorrectedMeasures = correctedMeasuresService.GetAll().ToList();
        if (listCorrectedMeasures.Count != 0)
        {
            List<VMCorrectedMeasures> listVMCorrectedMeasures = mapper.Map<List<CorrectedMeasures>,
            List<VMCorrectedMeasures>>(listCorrectedMeasures);
            for (int j = 0; j < iDProbesSelected.Length; j++)
            {
                List<VMCorrectedMeasures> listVMCorrectedMeasuresAux =
                    listVMCorrectedMeasures.FindAll(u => u.ProbeCode == ListVMProbe.First(k =>
                    k.IdProbe == iDProbesSelected[j]).Code);
                if (listVMCorrectedMeasuresAux.Count != 0)
                {
                    numberOfSamples[j] = (int)listVMCorrectedMeasuresAux.Max(u => u.MeasureIndex);
                    sameLastSample[j] = false;
                    lastSampleRead[j] = new TempData
                    {
                        Date = (DateTime)listVMCorrectedMeasuresAux.First(k=>k.MeasureIndex ==
                            listVMCorrectedMeasuresAux.Max(u => u.MeasureIndex)).Date,
                        Temp = (double)listVMCorrectedMeasuresAux.First(k => k.MeasureIndex ==
                            listVMCorrectedMeasuresAux.Max(u => u.MeasureIndex)).CorrectedTemp,
                        Order = numberOfSamples[j] - 1,
                        IdPrimaryMeasures = (int)listVMCorrectedMeasuresAux.First(k => k.MeasureIndex
                            == listVMCorrectedMeasuresAux.Max(u => u.MeasureIndex)).IdPrimaryMeasures
                    };
                }
                else
                {
                    numberOfSamples[j] = 0;
                    sameLastSample[j] = false;
                }
            }

            DateTime lastDate = new DateTime();
            lastDate = lastSampleRead[0].Date;
            int h = 0;

            // Update lastSampleRead for every probe.
            foreach (TempData measureOfProbe in lastSampleRead)
            {
                if((measureOfProbe.Date-lastDate).TotalSeconds > 0)
                {
                    lastDate = measureOfProbe.Date;
                }
                h++;
            }
            h = 0;
            foreach (TempReader reader in tempReaders)
            {
                lastSampleRead[h] = reader.Get(lastDate);
                numberOfSamples[h] = lastSampleRead[h].Order;
                h++;
            }

            List<Uncertainty> listUncertainty = uncertaintyService.GetAll().ToList();
```

```
            List<VMUncertainty> listVMUncertainty = mapper.Map<List<Uncertainty>,
            List<VMUncertainty>>(listUncertainty);
            listVMUncertainty = listVMUncertainty.FindAll(u => u.IdTest == IDTestSelected);
            numberOfSection = listVMUncertainty.Max(u => u.NumberOfSection) + 1;
            int availableProbes = 0; // Indicates how many probes have been read.
            decimal averageTemp = 0; // Average temperature of this measurement.
            for (i = 0; i < tempReaders.Count; i++)
            {
                if (sameLastSample[i] == false && numberOfSamples[i] != 0)
                {
                    availableProbes++;
                    averageTemp = averageTemp + (decimal)lastSampleRead[i].Temp;
                }
            }
            averageTemp = averageTemp / availableProbes;
            currentPeriod = averageTemp > topBeginDwellPeriod || averageTemp < bottomBeginDwellPeriod
            ? true : false;
        }
        #endregion

        #endregion


        #region 2) Get primary data from database and calculate their uncertainty.

        worker.ReportProgress(1, "> Initialization completed.");
        worker.ReportProgress(1, "");
        worker.ReportProgress(1, "> Calculations of uncertainty are about to begin.");
        worker.ReportProgress(1, "> Calculations started at " + DateTime.Now.ToShortTimeString() + "
            , " + DateTime.Now.ToShortDateString());
        worker.ReportProgress(1, "");

        i = 0;
        IsTestFinished(iDProbesSelected, tempReaders, numberOfSamples);
        while (testFinished == false)
        {
            if (backgroundWorkerCalculations.CancellationPending == true)
            {
                unitOfWork.Reset();
                e.Cancel = true;
                return;
            }

            #region 2.1) Get samples and 2.2) Correction of temperature using polinomial coeficients.
            i = 0;
            foreach (TempReader tempReader in tempReaders)
            {
                if (numberOfSamples[i] != 0)
                {
                    TempData tempSample = new TempData();

                    // Save primary sample at probeSamples[i]
                    tempSample = tempReader.Get(numberOfSamples[i]);

                    if (tempSample.Date != lastSampleRead[i].Date)
                    {
                        // In probe i, put a sample at samplesCount of that probe
                        probeSamples[i].Add(tempSample);

                        // Calculate correction of last probe sample.
                        TempData tempSampleCorrected = new TempData
                        {
                            Date = tempSample.Date,
                            Temp = (double)ListVMProbeCalibration[i].A0 +
                            (double)ListVMProbeCalibration[i].B0 * tempSample.Temp +
                            (double)ListVMProbeCalibration[i].C0 * Math.Pow(tempSample.Temp, 2) +
                            (double)ListVMProbeCalibration[i].D0 * Math.Pow(tempSample.Temp, 3),
                            Order = tempSample.Order,
                            IdPrimaryMeasures = tempSample.IdPrimaryMeasures
                        };
                        probeSamplesCorrected[i].Add(tempSampleCorrected);

                        AddToDBContext(unitOfWork, mapper, iDProbesSelected, i, tempSampleCorrected);

                        lastSampleRead[i] = tempSampleCorrected;
```

```
                    sameLastSample[i] = false;
                    numberOfSamples[i]++;
                }
                else
                {
                    sameLastSample[i] = true;
                }
            }
            else
            {
                TempData tempSample = new TempData();

                // Save primary sample at probeSamples[i]
                tempSample = tempReader.Get(numberOfSamples[i]);

                // In probe i, put a sample at samplesCount of that probe
                probeSamples[i].Add(tempSample);

                // Calculate correction of last probe sample.
                TempData tempSampleCorrected = new TempData
                {
                    Date = tempSample.Date,
                    Temp = (double)ListVMProbeCalibration[i].A0 +
                        (double)ListVMProbeCalibration[i].B0 * tempSample.Temp +
                    (double)ListVMProbeCalibration[i].C0 * Math.Pow(tempSample.Temp, 2) +
                    (double)ListVMProbeCalibration[i].D0 * Math.Pow(tempSample.Temp, 3),
                    Order = tempSample.Order,
                    IdPrimaryMeasures = tempSample.IdPrimaryMeasures
                };
                probeSamplesCorrected[i].Add(tempSampleCorrected);

                AddToDBContext(unitOfWork, mapper, iDProbesSelected, i, tempSampleCorrected);

                lastSampleRead[i] = tempSampleCorrected;
                sameLastSample[i] = false;
                numberOfSamples[i]++;
            }
            i++;
        }
        #endregion


        if (AllSamplesAreTheSame(sameLastSample) != true)
        {

            #region 2.3) Get which section is running

            #region Average temp and alarm checking.
            int availableProbes = 0; // Indicates how many probes have been read.
            decimal averageTemp = 0; // Average temperature of this measurement.
            for (i = 0; i < tempReaders.Count; i++)
            {
                if (sameLastSample[i] == false)
                {
                    availableProbes++;
                    averageTemp = averageTemp +
                    (decimal)probeSamplesCorrected[i][probeSamplesCorrected[i].Count - 1].Temp;
                }
            }
            averageTemp = averageTemp / availableProbes;
            listAverageTemp.Add(averageTemp);
            foreach (VMAlarm alarm in ListVMAlarm)
            {
                CheckAlarms(alarm.Code, averageTemp, worker);
            }
            #endregion

            // If average temperature is inside the range of dwell period, we can consider that
                test is in dwell period.
            lastPeriod = currentPeriod;
            currentPeriod = averageTemp > topBeginDwellPeriod || averageTemp <
                bottomBeginDwellPeriod ? true : false;

            // Pick the maximum time which has passed in the biggest list of
```

```
                ProbeSamplesCorrected.
                    int maxSamples = 0;
                    mostSampledProbe = -1;
                    for (i = 0; i < probeSamplesCorrected.Count; i++)
                    {
                        if (probeSamplesCorrected[i].Count > maxSamples)
                        {
                            maxSamples = probeSamplesCorrected[i].Count;
                            mostSampledProbe = i;
                        }
                    }

                    // It is neccesary if the first of all periods is a dwell one.
                    if(numberOfSamples[mostSampledProbe] == 1)
                    {
                        lastPeriod = currentPeriod;
                    }

                    if (currentPeriod != lastPeriod)
                    {
                        numberOfSection++;
                        unitOfWork.Commit();
                    }

                    calculateUncertainty = lastPeriod == true && currentPeriod == false ? true : false;

                    #endregion

                    #region 2.4) After a dwell period, calculate uncertainty and 2.5) Check alarms.

                    // Check if test has finished
                    IsTestFinished(iDProbesSelected, tempReaders, numberOfSamples);

                    if (calculateUncertainty == true || (testFinished == true && currentPeriod == true))
                    {
                        // If minimun dwell time has not passed, send an error/a message.
                        // If last period was shorter than 10% of the time of minimum dwell period,
                        consider it as ramp.
                        if (VMProgramTest.MinimunDwellTime * 0.1 >=
                        Math.Abs((probeSamplesCorrected[mostSampledProbe][maxSamples - 1].Date -
                        probeSamplesCorrected[mostSampledProbe][0].Date).TotalSeconds))
                        {
                            // Add this segment to last ramp period
                            numberOfSection = numberOfSection--;
                        }
                        else if (VMProgramTest.MinimunDwellTime >=
                        Math.Abs((probeSamplesCorrected[mostSampledProbe][maxSamples - 1].Date -
                        probeSamplesCorrected[mostSampledProbe][0].Date).TotalSeconds))
                        {
                            worker.ReportProgress(1, "> Segment " + (numberOfSection - 1) + " (dwell)
                                completed.");
                            worker.ReportProgress(2, "> Dwell time in segment " + (numberOfSection - 1) +
                                " was shorter than minimum.");
                            VMUncertainty vMUncertainty = new VMUncertainty
                            {
                                IdTest = IDTestSelected,
                                SectionDescription = "Short dwell period",
                                NumberOfSection = numberOfSection - 1,
                                ReferenceTemperature = null,
                                ExpandedUncertaintyTemp = null,
                                ReferenceHumidity = null,
                                ExpandedUncertaintyHum = null
                            };

                            if (backgroundWorkerCalculations.CancellationPending == true)
                            {
                                unitOfWork.Reset();
                                e.Cancel = true;
                                return;
                            }

                         unitOfWork.DbContext.Uncertainty.Persist(mapper).InsertOrUpdate(vMUncertainty);
                         unitOfWork.Commit();
                        }
                        // Else if dwell time was longer than minimum.
```

```
                    else
                    {
                        #region 2.4.0) Get how many probes can be evaluated.
                        List<bool> enoughSamples = new List<bool>();
                        for (i = 0; i < iDProbesSelected.Length; i++)
                        {
                            enoughSamples.Add(new bool());
                            if (Math.Abs((probeSamplesCorrected[i][probeSamplesCorrected[i].Count -
                            1].Date- probeSamplesCorrected[i][0].Date).TotalSeconds) >=
                            VMProgramTest.MinimunDwellTime)
                            {
                                enoughSamples[i] = true;
                            }
                            else
                            {
                                enoughSamples[i] = false;
                            }
                        }
                        #endregion

                        #region 2.4.1) TLD Calibration
                        i = 0;
                        UncertaintyObject tLDCalibration = new UncertaintyObject
                        {
                            StandardUncertainty = 0m
                        };
                        foreach (VMProbeCalibration register in ListVMProbeCalibration)
                        {
                            if (enoughSamples[i] == true)
                            {
                                if ((decimal)register.Uncertainty >
                                    tLDCalibration.StandardUncertainty)
                            tLDCalibration.StandardUncertainty = (decimal)register.Uncertainty;
                            }
                            i++;
                        }
                        // Normal distribution (k=2), sensity coeficient = 1
                        tLDCalibration.Divisor = 2m;
                        tLDCalibration.SensityCoefficient = 1m;
                        tLDCalibration.Uncertainty = tLDCalibration.StandardUncertainty /
                            tLDCalibration.Divisor * tLDCalibration.SensityCoefficient;
                        tLDCalibration.DegreesOfFreedom = 1000000m;
                        tLDCalibration.TotalUncertainty = (tLDCalibration.Uncertainty *
                            tLDCalibration.Uncertainty *
                            tLDCalibration.Uncertainty * tLDCalibration.Uncertainty) /
                            tLDCalibration.DegreesOfFreedom;

                        #endregion


                        #region 2.4.2) Deviation of samples
                        UncertaintyObject deviationOfSamples = new UncertaintyObject
                        {
                            StandardUncertainty = 0m
                        };
                        for (i = 0; i < probeSamplesCorrected.Count; i++) // Calculate standard
                            deviation
                        {
                            if (enoughSamples[i] == true)
                            {
                                decimal standardDeviation;
                                decimal averageTempAux = 0m;
                                for (int j = 0; j < probeSamplesCorrected[i].Count; j++)
                                {
                                    averageTempAux = averageTempAux +
                                    (decimal)probeSamplesCorrected[i][j].Temp;
                                }
                                // Mean of samples
                                averageTempAux = averageTempAux / probeSamplesCorrected[i].Count;

                                decimal aux1 = 0m;
                                for (int j = 0; j < probeSamplesCorrected[i].Count; j++)
                                {
                                    aux1 = aux1 + (decimal)Math.Pow(probeSamplesCorrected[i][j].Temp
                                    - (double)averageTempAux, 2);
```

```
                }

                standardDeviation = (decimal)(Math.Sqrt((double)(aux1 /
                    (probeSamplesCorrected[i].Count - 1))));

                // Take the biggest standard deviation
                if (standardDeviation > deviationOfSamples.StandardUncertainty)
                {
                    deviationOfSamples.StandardUncertainty = standardDeviation;
                    deviationOfSamples.Divisor =
                    (decimal)Math.Sqrt(probeSamplesCorrected[i].Count);
                    deviationOfSamples.DegreesOfFreedom =
                    probeSamplesCorrected[i].Count - 1;
                }
            }
        }
        deviationOfSamples.SensityCoefficient = 1m;
        deviationOfSamples.Uncertainty = deviationOfSamples.StandardUncertainty /
            deviationOfSamples.Divisor * deviationOfSamples.SensityCoefficient;
        deviationOfSamples.TotalUncertainty =
            (decimal)Math.Pow((double)deviationOfSamples.Uncertainty, 4) /
            deviationOfSamples.DegreesOfFreedom;
        #endregion

        #region 2.4.3) TLD Drift
        i = 0;
        UncertaintyObject tLDDrift = new UncertaintyObject
        {
            StandardUncertainty = 0m
        };
        foreach (VMProbeCalibration register in ListVMProbeCalibration)
        {
            if (enoughSamples[i] == true)
            {
                if ((decimal)register.Drift > tLDDrift.StandardUncertainty)
                    tLDDrift.StandardUncertainty = (decimal)register.Drift;
            }
            i++;
        }
        // Normal distribution (k=2), sensity coeficient = 1
        tLDDrift.SensityCoefficient = 1m;
        tLDDrift.Divisor = (decimal)Math.Sqrt(12);
        tLDDrift.Uncertainty = tLDDrift.StandardUncertainty / tLDDrift.Divisor *
            tLDDrift.SensityCoefficient;
        tLDDrift.DegreesOfFreedom = 1000000m;
        tLDDrift.TotalUncertainty = (decimal)Math.Pow((double)tLDDrift.Uncertainty,
            4) / tLDDrift.DegreesOfFreedom;

        #endregion


        #region 2.4.4) Test Uniformity
        decimal[] average = new decimal[probeSamplesCorrected.Count];
        for (i = 0; i < probeSamplesCorrected.Count; i++)
        {
            average[i] = 0;
        }
        for (i = 0; i < probeSamplesCorrected.Count; i++)
        {
            for (int j = 0; j < probeSamplesCorrected[i].Count; j++)
            {
                average[i] = average[i] + (decimal)probeSamplesCorrected[i][j].Temp;
            }
            average[i] = average[i] / (decimal)probeSamplesCorrected[i].Count;
        }
        decimal averageSum = 0m;
        for (i = 0; i < probeSamplesCorrected.Count; i++)
        {
            averageSum = averageSum + average[i];
        }
        decimal averageOfAverageSum = averageSum / probeSamplesCorrected.Count;
        UncertaintyObject testUniformityUncertainty = new UncertaintyObject
        {
            StandardUncertainty = 0m
        };
```

```
                    for (i = 0; i < probeSamplesCorrected.Count; i++)
                    {
                        if (Math.Abs(averageOfAverageSum - average[i]) >
                        testUniformityUncertainty.StandardUncertainty)
                            testUniformityUncertainty.StandardUncertainty =
                                Math.Abs(averageOfAverageSum - average[i]);
                    }
                    testUniformityUncertainty.Divisor = (decimal)Math.Sqrt(3);
                    testUniformityUncertainty.SensityCoefficient = 1m;
                    testUniformityUncertainty.Uncertainty =
                        testUniformityUncertainty.StandardUncertainty /
                        testUniformityUncertainty.Divisor *
                        testUniformityUncertainty.SensityCoefficient;
                    testUniformityUncertainty.DegreesOfFreedom = 1000000m;
                    testUniformityUncertainty.TotalUncertainty =
                        (decimal)Math.Pow((double)testUniformityUncertainty.Uncertainty, 4) /
                        testUniformityUncertainty.DegreesOfFreedom;
                    #endregion

                    #region 2.4.5) Chamber Stability and Uniformity Uncertainties + Charge effect
                    UncertaintyObject chamberStabilityUncertainty = new UncertaintyObject
                    {
                        Divisor = 2m,
                        SensityCoefficient = 1m,
                        DegreesOfFreedom = 1000000m
                    };
                    UncertaintyObject chamberUniformityUncertainty = new UncertaintyObject
                    {
                        Divisor = 2m,
                        SensityCoefficient = 1m,
                        DegreesOfFreedom = 1000000m
                    };
                    UncertaintyObject chargeEfect = new UncertaintyObject
                    {
                        Divisor = (decimal)Math.Sqrt(12),
                        SensityCoefficient = 1m,
                        DegreesOfFreedom = 1000000m
                    };

                    bool tempOutOfRangeError = false;
                    if (averageTemp >= (decimal)VMChamberCalibration.TempLow1 && averageTemp <
                        (decimal)VMChamberCalibration.TempLow2)
                    {
                        chamberStabilityUncertainty.StandardUncertainty =
                        (decimal)VMChamberCalibration.StabGradient1;
                        chamberUniformityUncertainty.StandardUncertainty =
                        (decimal)VMChamberCalibration.Uniformity1;
                        chargeEfect.StandardUncertainty =
                        Math.Abs((decimal)VMChamberCalibration.Uniformity1 -
                        (decimal)VMChamberCalibration.UnifGradient1) +
                            Math.Abs(deviationOfSamples.StandardUncertainty -
                           (decimal)VMChamberCalibration.Stability1);
                    }
                    else if (averageTemp >= (decimal)VMChamberCalibration.TempLow2 && averageTemp
                        < (decimal)VMChamberCalibration.TempLow3)
                    {
                        chamberStabilityUncertainty.StandardUncertainty =
                        (decimal)VMChamberCalibration.StabGradient2;
                        chamberUniformityUncertainty.StandardUncertainty =
                        (decimal)VMChamberCalibration.Uniformity2;
                        chargeEfect.StandardUncertainty =
                        Math.Abs((decimal)VMChamberCalibration.Uniformity2 -
                        (decimal)VMChamberCalibration.UnifGradient2) +
                            Math.Abs(deviationOfSamples.StandardUncertainty -
                        (decimal)VMChamberCalibration.Stability2);
                    }
                    else if (averageTemp >= (decimal)VMChamberCalibration.TempLow3)
                    {
                        chamberStabilityUncertainty.StandardUncertainty =
                        (decimal)VMChamberCalibration.StabGradient3;
                        chamberUniformityUncertainty.StandardUncertainty =
                        (decimal)VMChamberCalibration.Uniformity3;
                        chargeEfect.StandardUncertainty =
                        Math.Abs((decimal)VMChamberCalibration.Uniformity3 -
                        (decimal)VMChamberCalibration.UnifGradient3) +
```

```
                    Math.Abs(deviationOfSamples.StandardUncertainty -
                    (decimal)VMChamberCalibration.Stability3);
            }
            else
            {
                worker.ReportProgress(2, "> Temperature out of range. Unable to calculate
                uncertainties.");
                tempOutOfRangeError = true;
            }
            #endregion


            if (tempOutOfRangeError == false)
            {
                chamberStabilityUncertainty.Uncertainty =
                chamberStabilityUncertainty.StandardUncertainty /
                chamberStabilityUncertainty.Divisor *
                chamberStabilityUncertainty.SensityCoefficient;

                chamberUniformityUncertainty.Uncertainty =
                chamberUniformityUncertainty.StandardUncertainty /
                chamberUniformityUncertainty.Divisor *
                chamberUniformityUncertainty.SensityCoefficient;
                chargeEfect.Uncertainty = chargeEfect.StandardUncertainty /
                chargeEfect.Divisor * chargeEfect.SensityCoefficient;

                chamberStabilityUncertainty.TotalUncertainty =
                (decimal)Math.Pow((double)chamberStabilityUncertainty.Uncertainty, 4) /
                chamberStabilityUncertainty.DegreesOfFreedom;

                chamberUniformityUncertainty.TotalUncertainty =
                (decimal)Math.Pow((double)chamberUniformityUncertainty.Uncertainty, 4) /
                chamberUniformityUncertainty.DegreesOfFreedom;
                chargeEfect.TotalUncertainty =
                (decimal)Math.Pow((double)chargeEfect.Uncertainty, 4) /
                chargeEfect.DegreesOfFreedom;

                #region 2.4.6) Radiation efect
                UncertaintyObject radiationEfect = new UncertaintyObject
                {
                    StandardUncertainty = 0.1m * chargeEfect.StandardUncertainty,
                    Divisor = (decimal)Math.Sqrt(12),
                    SensityCoefficient = 1m,
                    DegreesOfFreedom = 1000004m
                };
                radiationEfect.Uncertainty = radiationEfect.StandardUncertainty /
                radiationEfect.Divisor * radiationEfect.SensityCoefficient;
                radiationEfect.TotalUncertainty =
                (decimal)Math.Pow((double)radiationEfect.Uncertainty, 4) /
                radiationEfect.DegreesOfFreedom;
                #endregion

                #region 2.4.7) Combined uncertainty
                List<UncertaintyObject> listUncertaintyObject = new
List<UncertaintyObject>
{
    tLDCalibration,
    tLDDrift,
    deviationOfSamples,
    testUniformityUncertainty,
    chamberStabilityUncertainty,
    chamberUniformityUncertainty,
    chargeEfect,
    radiationEfect
};

                decimal combinedUncertainty = 0m;
                decimal totalCombinedUncertanty = 0m;
                decimal combinedUncertaintyDegreesOfFreedom;
                foreach (UncertaintyObject item in listUncertaintyObject)
                {
                    combinedUncertainty = combinedUncertainty +
                        (decimal)Math.Pow((double)item.Uncertainty, 2);
                    totalCombinedUncertanty = totalCombinedUncertanty +
```

```
                        item.TotalUncertainty;
                        }
                        combinedUncertainty = (decimal)Math.Sqrt((double)combinedUncertainty);
                        combinedUncertaintyDegreesOfFreedom =
                        (decimal)Math.Pow((double)combinedUncertainty, 4) /
                        totalCombinedUncertanty;
                        #endregion

                        #region 2.4.8) Expanded uncertainty
                        decimal expandedUncertaintyDegreesOfFreedom =
                        Math.Truncate(combinedUncertaintyDegreesOfFreedom);
                        // Calculate inverse of T-Student distribution (like Excel)
                        Chart Chart1 = new Chart();
                        double expandedCoverFactor =
                        Chart1.DataManipulator.Statistics.InverseTDistribution(1 - 0.9554,
                        (int)expandedUncertaintyDegreesOfFreedom);
                        expandedUncertainty = combinedUncertainty * (decimal)expandedCoverFactor;
                        #endregion

                        #region 2.4.9) Add section uncertainty to database.
                        VMUncertainty vMUncertainty = new VMUncertainty
                        {
                            IdTest = IDTestSelected,
                            SectionDescription = "Normal dwell period",
                            ReferenceTemperature = (double)averageOfAverageSum,
                            ExpandedUncertaintyTemp = (double)expandedUncertainty,
                            ReferenceHumidity = null,
                            ExpandedUncertaintyHum = null
                        };
                        vMUncertainty.NumberOfSection = testFinished == true && currentPeriod ==
                        true ? numberOfSection : numberOfSection - 1;

                        worker.ReportProgress(1, "> Segment " + (numberOfSection - 1) + " (dwell)
                        completed.");
                        worker.ReportProgress(1, "> Uncertainty in section " + (numberOfSection -
                        1) + " is +-" + expandedUncertainty.ToString("0.00") + " .");


                        // Check alarms
                        foreach (VMAlarm alarm in ListVMAlarm)
                        {
                            CheckAlarms(alarm.Code, listAverageTemp, expandedUncertainty,
                                numberOfSection, worker);
                        }

                        // Clear lists of samples.
                        for (i = 0; i < probeSamples.Count; i++)
                        {
                            probeSamples[i].Clear();
                            probeSamplesCorrected[i].Clear();
                        }
                        listAverageTemp.Clear();

                        if (backgroundWorkerCalculations.CancellationPending == true)
                        {
                            unitOfWork.Reset();
                            e.Cancel = true;
                            return;
                        }

                    unitOfWork.DbContext.Uncertainty.Persist(mapper).InsertOrUpdate(vMUncertainty);
                    unitOfWork.Commit();
                            #endregion
                        }

                    }

                }
                #endregion

                #region 2.6) Add section (without uncertainty) to database if it was a ramp
                // Add ramp section to database table "Uncertainty".
                if (lastPeriod == false && currentPeriod == true || currentPeriod == false &&
                    testFinished == true) // Ramp
```

```
                {
                    // Subir NumberOfSection - 1 (ramp) si no se ha subido ya.
                    VMUncertainty vMUncertainty = new VMUncertainty
                    {
                        IdTest = IDTestSelected,
                        SectionDescription = "Ramp period",
                        NumberOfSection = numberOfSection - 1,
                        ReferenceTemperature = null,
                        ExpandedUncertaintyTemp = null,
                        ReferenceHumidity = null,
                        ExpandedUncertaintyHum = null
                    };
                    // Clear lists of samples.
                    for (i = 0; i < probeSamples.Count; i++)
                    {
                        probeSamples[i].Clear();
                        probeSamplesCorrected[i].Clear();
                    }
                    listAverageTemp.Clear();

                    if (backgroundWorkerCalculations.CancellationPending == true)
                    {
                        unitOfWork.Reset();
                        e.Cancel = true;
                        return;
                    }

                    unitOfWork.DbContext.Uncertainty.Persist(mapper).InsertOrUpdate(vMUncertainty);
                    unitOfWork.Commit();

                    worker.ReportProgress(1, "> Segment " + (numberOfSection - 1) + " (ramp)
                    completed.");

                }

                #endregion


            }
            else
            {
                IsTestFinished(iDProbesSelected, tempReaders, numberOfSamples);
            }

            if(IsTestOnline(iDProbesSelected, tempReaders, numberOfSamples) == true)
            {
                List<double> auxList = new List<double>
                {
                    (double)numberOfSamples[mostSampledProbe],
                    (double)tempReaders[mostSampledProbe].GetNumberOfSamples()
                };
                worker.ReportProgress(4, auxList);
                worker.ReportProgress(5, "ONLINE");
            }
            else
            {
                List<double> auxList = new List<double>
                {
                    (double)numberOfSamples[mostSampledProbe],
                    (double)tempReaders[mostSampledProbe].GetNumberOfSamples()
                };
                worker.ReportProgress(4, auxList);
                worker.ReportProgress(5, "OFFLINE");
            }

        }
        #endregion

        #region 3) Once test has finished.
        worker.ReportProgress(3);
        worker.ReportProgress(1, "> Calculations finished at " + DateTime.Now.ToShortTimeString() + "
            , " + DateTime.Now.ToShortDateString());

        unitOfWork.Commit();
```

```
            // Finally, this code writes down that the test has finished.
            while (testFinished == true)
            {
                if (backgroundWorkerCalculations.CancellationPending == true)
                {
                    unitOfWork.Reset();
                    e.Cancel = true;
                    return;
                }
                worker.ReportProgress(1, "> Click Back button to return to main menu.");
                WaitMilliseconds(10000);
            }
            #endregion

        }
    }
}
```

SegmentsForm

```
1.  using AutoMapper;
2.  using AutoMapper.EntityFramework;
3.  using System;
4.  using System.Collections.Generic;
5.  using System.ComponentModel;
6.  using System.Data;
7.  using System.Drawing;
8.  using System.Linq;
9.  using System.Text;
10. using System.Threading.Tasks;
11. using System.Windows.Forms;
12. using VirtualLab_TestBinding_v2.Data.Infrastructure;
13. using VirtualLab_TestBinding_v2.Data.Repositories;
14. using VirtualLab_TestBinding_v2.DataReader;
15. using VirtualLab_TestBinding_v2.DataReader.Interfaces;
16. using VirtualLab_TestBinding_v2.Domain.Models;
17. using VirtualLab_TestBinding_v2.Infrastructure;
18. using VirtualLab_TestBinding_v2.Services;
19. using VirtualLab_TestBinding_v2.ViewModel;
20. using VirtualLab_TestBinding_v2.Forms;
21. using System.Threading;
22.
23. namespace VirtualLab_TestBinding_v2.Forms
24. {
25.     public partial class SegmentsForm : Form
26.     {
27.        IUnitOfWork unitOfWork = CompositionRoot.Resolve<IUnitOfWork>();
28.        IMapper mapper = CompositionRoot.Resolve<IMapper>();
29.        List<Segment> listSegment;
30.        List<VMSegment> listVMSegment;
31.        int iDProgramTestSelected;
32.        string programAppName;
33.
34.        public SegmentsForm(int n,string pname)
35.        {
36.             InitializeComponent();
37.             iDProgramTestSelected = n;
38.             programAppName = pname;
39.             InitializeSegmentsGridView();
40.        }
41.
42.        internal class VMSegmentOnGrid
43.        {
44.             public Nullable<int> NumberOfSegment { get; set; }
45.             public Nullable<double> Time { get; set; }
46.             public Nullable<double> Temperature { get; set; }
47.             public Nullable<double> Humidity { get; set; }
48.        }
49.
50.        private void InitializeSegmentsGridView()
```

```
51.         {
52.              ISegmentService segmentService = CompositionRoot.Resolve<ISegmentService>();
53.
54.              // Write program name in ProgramNameTextBox
55.              ProgramNameTextBox.AppendText(programAppName);
56.              // Fill SegmentGridView
57.             listSegment = segmentService.GetAll().ToList();
58.             listVMSegment = mapper.Map<List<Segment>, List<VMSegment>>(listSegment);
59.             listVMSegment = listVMSegment.FindAll(u=>u.IdProgramTest == iDProgramTestSelected);
60.             List<VMSegmentOnGrid> listVMSegmentOnGrid = new List<VMSegmentOnGrid>();
61.             foreach (VMSegment register in listVMSegment)
62.             {
63.                 listVMSegmentOnGrid.Add(new VMSegmentOnGrid { NumberOfSegment =
                    register.NumberOfSegment, Time = register.Time, Temperature = register.Temperature,
                    Humidity = register.Humidity });
64.             }
65.             BindingList<VMSegmentOnGrid> bindingListVMSegmentsOnGrid = new
                BindingList<VMSegmentOnGrid>();
66.             for (int i = 0; i < listVMSegmentOnGrid.Count; i++)
67.             {
68.                 bindingListVMSegmentsOnGrid.Add(listVMSegmentOnGrid[i]);
69.             }
70.             bindingSourceSegments.DataSource = bindingListVMSegmentsOnGrid;
71.             SegmentsGridView.DataSource = bindingSourceSegments;
72.             Controls.Add(SegmentsGridView);
73.         }
74.     }
75. }
```

ErrorForm

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.ComponentModel;
4.  using System.Data;
5.  using System.Drawing;
6.  using System.Linq;
7.  using System.Text;
8.  using System.Threading.Tasks;
9.  using System.Windows.Forms;
10.
11. namespace VirtualLab_TestBinding_v2.Forms
12. {
13.    public partial class ErrorForm : Form
14.     {
15.         public ErrorForm()
16.         {
17.             InitializeComponent();
18.         }
19.
20.         private void OkButton_Click(object sender, EventArgs e)
21.         {
22.             Close();
23.         }
24.     }
25. }
```

CancelMonitoringForm

```
1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Drawing;
6. using System.Linq;
7. using System.Text;
8. using System.Threading.Tasks;
9. using System.Windows.Forms;
10.
11. namespace VirtualLab_TestBinding_v2.Forms
```

```
12. {
13.     public partial class CancelMonitoringForm : Form
14.     {
15.         TestInProgressForm testInProgressForm;
16.
17.         public CancelMonitoringForm(TestInProgressForm _testInProgressForm)
18.         {
19.             InitializeComponent();
20.             testInProgressForm = _testInProgressForm;
21.         }
22.
23.         private void YesButton_Click(object sender, EventArgs e)
24.         {
25.             testInProgressForm.MonitoringCanceled = true;
26.             Close();
27.         }
28.
29.         private void NoButton_Click(object sender, EventArgs e)
30.         {
31.             testInProgressForm.MonitoringCanceled = false;
32.             Close();
33.         }
34.     }
35. }
```