Banco de pruebas para respiradores artificiales

Test Bench for Ventilators

Edgardo Hames[1] and Nicolás Papp

Bitlogic, Córdoba, Argentina
https://bilogic.io

Resumen La COVID-19 ha incrementado la necesidad de contar con respiradores artificiales que asistan a pacientes en unidades de cuidados intensivos. Un factor muy importante en la velocidad de producción de estos equipos es el tiempo de fabricación. En este trabajo presentamos la arquitectura de un banco de pruebas automatizadas para respiradores artificiales, los beneficios que se obtienen cuando la automatización se adopta en el proceso de control de calidad y una reseña de la adopción de prácticas ágiles en la verificación de sistemas críticos siendo éstas aplicadas por un equipo de ingeniería con orientación en electromedicina.

Keywords: banco de pruebas, pruebas de sistema, automatización, respirador artificial

Abstract COVID-19 has increased the need for ventilators that provide assistance for patients who are in intensive care units. An important factor in the production rate of these equipment is the manufacturing time. In this article, we introduce a) the architecture for an automated test bench for ventilators, b) the benefits obtained from automating the QA process, and c) a review of adopting agile practices applied by an engineering team that is focused on electromedicine.

Keywords: system testing, test bed, test automation, ventilator

Received December 2020; Accepted March 2021; Published June 2021

1. Introducción

En tiempos de la COVID-19, el respirador artificial se ha vuelto un equipamiento vital, indispensable y escaso para el tratamiento de pacientes con insuficiencias respiratorias [1]. Como respuesta a ello, entidades académicas [2–5], gubernamentales [6] y empresarias comenzaron a producir respiradores [7] o a aumentar su producción [8, 9].

Fabricar un respirador que cumpla con las normas, estándares y obtenga las aprobaciones necesarias para su uso en unidades de cuidados intensivos es un desafío enorme para equipos de ingeniería no especializados [10]. Se debería poner el foco en incrementar la capacidad de producción de aquellas organizaciones que ya los fabrican [11]. Ese incremento en la capacidad de producción puede darse mediante la incorporación de nuevos turnos de trabajo [12], y también se puede lograr mediante la automatización de procesos lentos, repetitivos y mecánicos.

En este informe, presentamos la experiencia de desarrollar y automatizar un banco de pruebas para respiradores artificiales. En la sección 2, se presenta una descripción conceptual del respirador artificial y los desafíos del desarrollo de software para sistemas críticos. En la sección 3, se describen la solución arquitectura de un banco de pruebas y las integraciones con herramientas de automatización de pruebas. En la sección 4, se presentan algunos desafíos en la incorporación de metodologías ágiles en industrias con procesos diferentes. Finalmente, en la última sección, se proponen posibles extensiones al trabajo presentado.

2. El respirador

El respirador es una máquina que suple la ventilación pulmonar espontánea por una ventilación mecánica en personas con insuficiencia respiratoria ya sea aguda o crónica [13].

En el gráfico de la Figura 1 se muestra una descripción esquemática de un respirador. Una mezcla de aire y oxígeno ingresan por medio de válvulas a un mecanismo de compresión, que es regulado por un software que, de acuerdo a configuraciones establecidas por el profesional médico, regula la mezcla de los gases y los administra al paciente a través de válvulas de salida. Finalmente, se puede medir la presión de salida de los gases o el dióxido de carbono que emite el paciente.

Según el modelo de respirador, la interfaz de configuración puede variar. Algunos modelos cuentan con teclas mecánicas y una pantalla para visualizar opciones. En otros, la pantalla puede ser táctil y permitir el ingreso de valores y la visualización de configuraciones.

Durante el desarrollo y prueba del aparato, será necesario poder controlar y medir todas esas interfaces. Por ejemplo, se deberán abrir y cerrar las distintas válvulas, simular la resistencia de un pulmón, probar las diferentes configuraciones posibles y medir los caudales de salida, observar los valores en pantalla o simular una interrupción de la alimentación eléctrica.

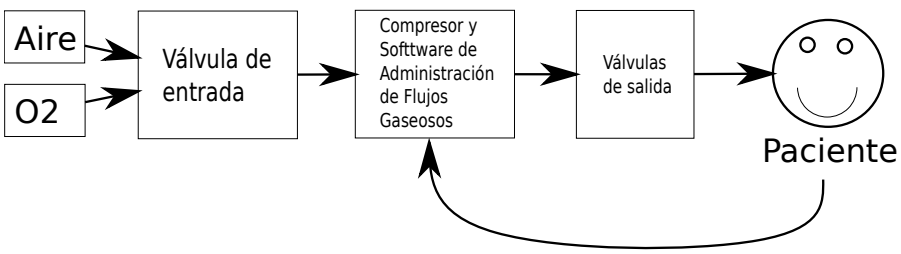

Figura 1. Diagrama esquemático de un respirador

2.1. Componentes

Antes de presentar el banco de pruebas, es necesario un entendimiento de alto nivel de los componentes del respirador, tal como se muestran en la Figura 2.

- Energía: la alimentación del respirador, conexión a 220V, batería y fuente.
- Control de teclado: teclado de comandos.
- Entrada de gases: válvulas que regulan la entrada de aire y oxígeno al respirador.
- Encoder: para moverse en los menús y seleccionar opciones.
- Alarmas: indicadores LED o de sonido.
- Pantalla: muestra el estado del respirador en todo momento.
- Controlador táctil: permite configurar el respirador como complemento del encoder y del teclado.
- Circuito de conexión al paciente: válvulas que regulan la salida de gases.

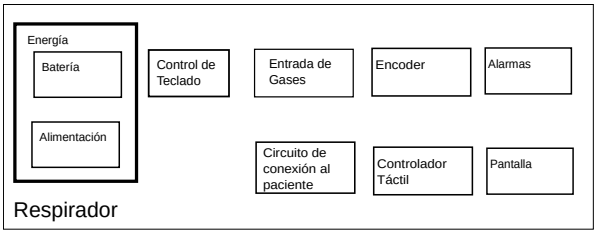

Figura 2. Componentes del respirador

2.2. Características de un sistema crítico

Un sistema de seguridad crítica (safety-critical system) es aquel cuya falla o mal funcionamiento puede ocasionar la muerte o lesiones graves para las personas, pérdidas o daños a la propiedad, o daño ambiental [14].

Los equipos médicos son sistemas críticos y, al igual que muchos otros productos, deben cumplir con numerosas etapas de pruebas y ensayos de producto antes de comenzar su producción masiva. En esas etapas, las mismas pruebas deben ejecutarse múltiples veces para el mismo producto para así verificar los cambios que se hacen en sus múltiples revisiones. Más aún, una vez fabricado el equipo, debe someterse a nuevas pruebas para garantizar que los valores operativos y de respuesta están dentro de los parámetros esperados.

Algunos escenarios donde se evidencia la criticidad del respirador incluyen la interrupción del flujo gaseoso al paciente (asfixia) o el exceso de presión (neumotórax). Algunas de estas fallas pueden ocurrir por problemas mecánicos (válvulas defectuosas), de software (rutinas con errores) o eléctricos (interrupción del suministro de energía, sobretensión, etc). Debería resultar evidente que estos equipos deben cumplir con controles mucho más estrictos que sistemas no críticos.

2.3. Normas y estándares

La incorporación de equipamiento electrónico y de software en el tratamiento de pacientes requiere el cumplimiento de normas internacionales [15–17] y las aprobaciones de los estados nacionales donde se utilicen [18, 19].

La norma IEC 60601 consiste en una serie de estándares técnicos respecto a la seguridad y el comportamiento de los equipos médicos eléctricos. Incluye, por ejemplo, los requerimientos relacionados con el ruido electromagnético, usabilidad, sistemas de alarmas, controladores fisiológicos de lazo cerrado, seguridad en respiradores para cuidados intensivos, etc.

Por su parte, la norma IEC 62304 regula el software de equipos médicos y los procesos de su ciclo de vida. Algunos de estos requerimientos hacen referencia al sistema de gestión de calidad, de riesgos, y de seguridad del software y sus pruebas [20] [21].

El cumplimiento de estas normas se certifica a través de procesos y auditorías que recolectan evidencia de la adhesión e implementación de sus requerimientos.

2.4. Suites de pruebas

Las secciones previas nos ayudan a entender la importancia que tienen las pruebas y el registro de sus resultados en el desarrollo de un respirador. En general, las suites de pruebas incluyen pruebas unitarias y de integración desarrolladas y ejecutadas por los mismos equipos de ingeniería, y también pruebas de sistema desarrolladas y ejecutadas por los equipos de control de calidad (*Quality Assurance*).

El objetivo de las pruebas unitarias es verificar que cada componente funciona correctamente de manera aislada. Las pruebas de integración se enfocan en las interacciones de los componentes entre sí; y las pruebas de sistema aseguran que el sistema como un todo cumple los requerimientos especificados [22–24].

Adicionalmente, con cada cambio que se va introduciendo al sistema (por ejemplo, la corrección de un defecto o el desarrollo de una nueva funcionalidad),

se debe garantizar que todas las funcionalidades (previamente desarrolladas y probadas) continúan funcionado correctamente. A esto se le llama pruebas de regresión.

Dado que las funcionalidades de los productos tienden a aumentar, las suites de pruebas también tienden a agrandarse. Por ende, los tiempos de ejecución también. Como el tiempo requerido para probar los equipos en cada etapa es un factor decisivo para cumplir los objetivos de negocio, la automatización de pruebas resulta un proyecto de inversión [25, 26].

La automatización de las pruebas ayuda a reducir el esfuerzo humano en su ejecución, disminuir la tasa de errores en ese proceso, y asegurar un registro sistemático y consistente de sus resultados (resultados, logs, capturas de pantallas, etc).

El objetivo del banco de pruebas es proveer un marco en el cual ejecutar las pruebas de sistema (como una caja negra) en ciclos de regresión de manera automática.

2.5. Especificaciones con Gherkin

La descripción de casos de pruebas utilizando el lenguaje Gherkin es una de las prácticas de la metodología Behavior-driven Development (BDD) [27].

Consiste en especificar el comportamiento del sistema mediante ejemplos partiendo los escenarios (pruebas) en tres secciones:

- En la sección Given (Dado), se especifica el estado del sistema antes de iniciar el comportamiento especificado en el escenario. Corresponde a las precondiciones de la prueba.
- En la sección When (Cuando), se describen el comportamiento o las acciones que se realizan sobre el sistema.
- En la sección Then (Entonces), se describen los cambios esperados en el sistema tras la ejecución de dichas acciones.

Esta herramienta ofrece un gran poder de abstracción, y brinda la ventaja adicional de permitir escribir los casos de prueba en un lenguaje común entre los expertos del dominio y los desarrolladores. Este tipo de descripciones también permite reducir la barrera de entendimiento para cualquier interesado en revisar las pruebas o mantenerlas. Los escenarios escritos en lenguaje Gherkin pueden convertirse en código ejecutable usando alguna herramienta como Cucumber [28], Behave [29] o Robot [30].

Otro beneficio de escribir las pruebas con Gherkin es que, en caso de falla, permite identificar rápidamente la situación, los casos de falla y el paso de ejecución correspondientes al error.

2.6. Pruebas del respirador

La suite de pruebas a automatizar con este banco están orientadas a la verificación del cumplimiento de normas de respuestas del equipo. Queda fuera del

alcance del proyecto la simulación de diferentes tipos de pacientes (neonatos, infantes, adultos) con distintos patrones de respiración (apnea o respiración voluntaria) o tratamiento (por ejemplo, suministro de drogas por vía respiratoria). Este tipo de simulaciones suele realizarse por medio de "pulmones artificiales"que permiten recrear dichos escenarios.

3. Arquitectura de la solución

3.1. Visión General

De la visión esquemática descrita en la sección 1, se puede apreciar que la complejidad de automatizar pruebas está dada por la cantidad de elementos que son necesarios controlar: el encendido del aparato, la pulsación de la pantalla táctil y teclas mecánicas, la manipulación de perillas, conexión y desconexión de mangueras con gases, y la lectura de valores presentados en pantalla. No solo debemos probar un software sino que, además, debemos ser capaces de interactuar con válvulas mecánicas, simular de manera fiel la operación del usuario e interpretar los datos mostrados en pantalla para verificar que sean los correctos. Por otro lado, al no tener acceso al código fuente del equipo no es posible analizarlo estáticamente ni emular el comportamiento de los dispositivos.

En el diagrama de la Figura 3, se muestra la arquitectura general de la solución.

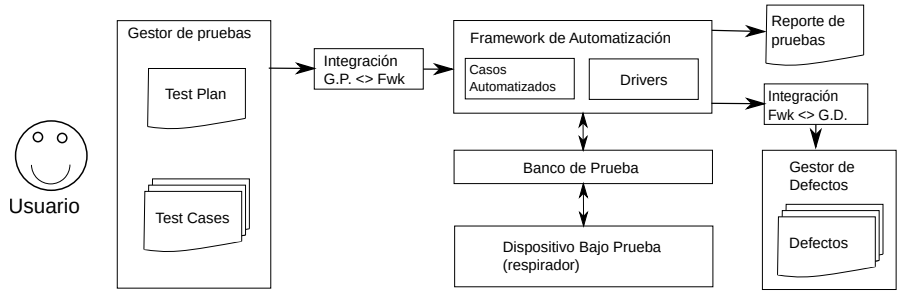

Figura 3. Arquitectura general de la solución

- El gestor de pruebas contiene las definiciones de los casos de prueba descriptos como escenarios (test cases) y los planes de prueba (test plan).
- El framework de Automatización contiene los casos de prueba automatizados.
- La integración entre el gestor de pruebas y el framework de automatización.
- Los drivers traducen los pasos automatizados a operaciones sobre el banco de pruebas.
- El banco de pruebas que interactúa con el modelo específico de respirador (dispositivo bajo prueba).

- La integración entre el framework de automatización y el gestor de defectos
- El gestor de defectos lleva el registro de los problemas detectados.
- El reporte de pruebas que incluye un registro de cada escenario ejecutado y sus resultados.

El procedimiento de uso es el siguiente:

1. El usuario (ingeniero de pruebas) define un plan de pruebas en el Gestor.
2. El usuario inicia la ejecución de pruebas en el framework de automatización a través del componente de integración.
3. El banco ejecuta los pasos dirigido por los drivers.
4. El framework de automatización genera un informe de pruebas con los resultados.

3.2. Banco de pruebas

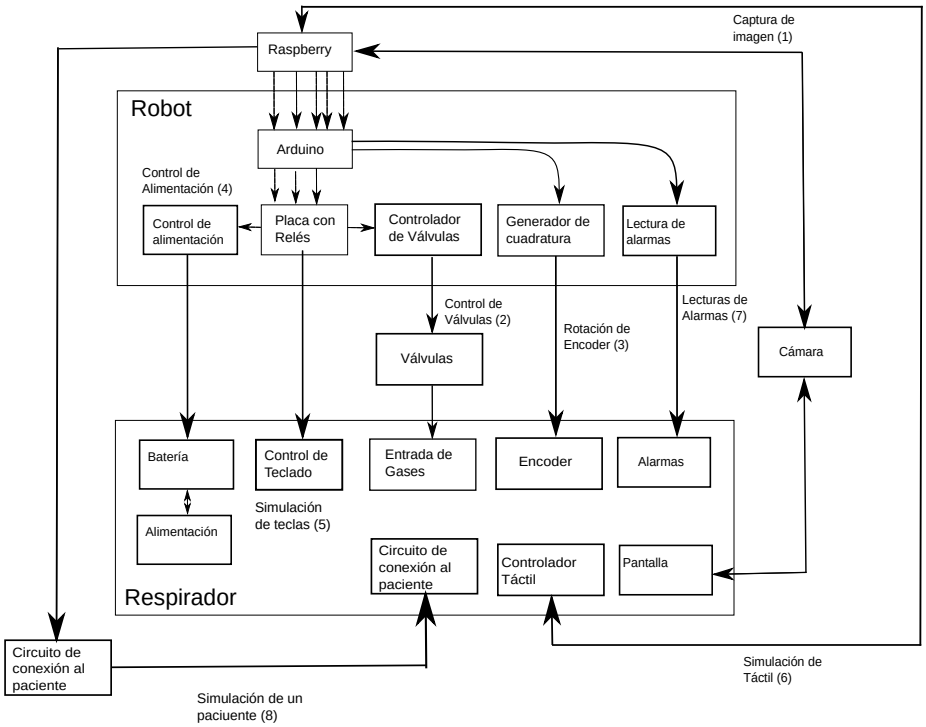

Figura 4. Esquema del banco de pruebas

La arquitectura del banco de pruebas es la que se presenta en la Figura 4 y cuenta con los siguientes componentes:

- Un ordenador (Raspberry) con los casos de pruebas y drivers. En este se incluyen:
  - Framework de pruebas: definición de cada uno de los casos de pruebas automáticos disponibles para su ejecución.
  - Visión Computarizada: integración de herramientas para el procesamiento de imágenes y detección de texto.
  - Drivers: herramientas utilizadas para poder comandar el robot de pruebas.
- Un robot de pruebas: compuesto principalmente por un microcontrolador con un circuito electrónico asociado, el cual puede llevar a cabo acciones comandadas por el gestor para generar los estímulos necesarios en el respirador:
  1. Captura de imágenes : captura de la pantalla del respirador, para que posteriormente se envíen al ordenador para su procesamiento. Se comanda la captura de una imagen a través de una cámara. Posteriormente esta captura se procesa con técnicas de Visión Computarizada para mitigar los siguientes problemas.
     - Adaptación de sistemas de coordenadas de la pantalla cuando ésta no es plana.
     - Filtrado de la imagen para remover reflejos y disparidad de luz en la captura y favorecer la detección de texto. Una vez procesadas, se recortan los sectores de la pantalla de interés para hacer detección de texto o presencia de distintos tipos de alarmas.
  2. Control de válvulas: suministro o interrupción del paso de los gases que necesita el respirador para poder observar su comportamiento en estos casos. Las válvulas son controladas por un relé que determina si se encuentran abiertas o cerradas.
  3. Rotación de encoder: simulación de la rotación de un encoder digital para poder seleccionar opciones o moverse a través de los menús. Para simular dicha interacción se reemplazaron sus entradas por salidas digitales del microcontrolador. Estas generan una señal en cuadratura cuando se desea simular la rotación del encoder.
  4. Control de alimentación: intervención de los circuitos de alimentación para que el robot sea capaz de controlar la alimentación del respirador para probar el funcionamiento de la batería. Debe permitir encender y apagar el respirador si así lo requiere o desconectar la batería para poder corroborar la respuesta del respirador ante su ausencia.
  5. Simulación de teclas: sistema comandado por relés para simular la pulsación de teclas en la pantalla del respirador. Para ello se realizó un mapeo de las distintas teclas a un relé de manera que el sistema de pruebas automáticas sepa qué relé accionar cuando se exigía la pulsación de una tecla.
  6. Simulación de dispositivo táctil: se conecta por un puerto al respirador y se envía un mensaje con las coordenadas de la pantalla donde se quiere simular la interacción. Para ello se debe conocer la ubicación exacta (coordenadas) del menú o botón a apretar, o bien obtenerla a partir del procesamiento de la imagen de la pantalla.

7. Lectura de alarmas: así como se pueden generar estímulos eléctricos se deben poder también realizar lecturas de señales correspondientes a LEDs y alarmas del respirador, para determinar si estas se encienden.
8. Simulación de un paciente: Un pulmón de pruebas que permite simular el comportamiento de un paciente.

3.3. Integraciones

Como en cualquier proyecto de software, es esencial concentrar los esfuerzos del equipo en el desarrollo de las partes esenciales del dominio y la integración de herramientas de terceros que resuelven otros problemas.

Los problemas no esenciales son los relacionados con el procesamiento de imágenes. Para resolverlos, dos herramientas libres fueron clave:

- OpenCV [31]: este framework de Visión Artificial se utilizó para capturar imágenes de la cámara, su procesamiento y reconocimiento de patrones.
- Tesseract OCR [32]: esta herramienta se utilizó para leer texto y números en las imágenes capturadas.

Otras integraciones necesarias son las relacionadas con las herramientas de gestión de pruebas y de defectos. Es fundamental hacer que el banco de pruebas se integre con las herramientas que usa el equipo que desarrolla el respirador para reducir la curva de adopción y seguir sus procesos de calidad.

La herramienta de gestión de pruebas contiene la definición de casos especificados en el formato Given-When-Then. El usuario define allí un plan de pruebas arbitrario según los cambios que se deseen validar. Para poder disparar las ejecuciones en el banco de pruebas es necesario vincular los pasos de los casos de prueba con los pasos en el framework de automatización. De este modo, se puede hacer un mapeo entre ambas herramientas y comandar la ejecución automatizada.

Por otro lado, cada paso del framework de automatización va generando un registro (log) de su ejecución que incluye el resultado de cada caso de prueba. En caso de falla de una prueba, se registra automáticamente un caso (issue) en la herramienta de gestión de defectos que vincula el identificador de la prueba, el registro de los pasos ejecutados (traza), y el resultado esperado vs el real.

3.4. Portabilidad/Extensibilidad

La portabilidad de las pruebas se logra mediante la jerarquización de los pasos de la prueba y la categorización en dos tipos de pasos:

- Sentencias de alto nivel de abstracción: son aquellas que van a componer los casos de pruebas y que deberían ser independientes del modelo o aparato utilizado.
- Sentencias de bajo nivel de abstracción: las sentencias de alto nivel están compuestas a su vez por éstos, que son dependientes de la implementación.

La extensibilidad se logra redefiniendo las sentencias de bajo nivel de abstracción para cada modelo o producto a automatizar. Por ejemplo, la selección de un determinado menú en un modelo puede ser a través de un Encoder pero ser Touch en el modelo siguiente.

4. Proyecto

4.1. Metodología

El proyecto funcionó como una suerte de piloto para determinar la factibilidad de implementar el banco de pruebas. Esto supuso un desafío para el cliente que estaba más acostumbrado a un desarrollo clásico usando proceso de cascada, y donde cada etapa estaba muy bien definida, con entradas y salidas muy claras.

En este caso, si bien había claridad respecto a lo que se deseaba construir, no estaban identificados los componentes o herramientas concretas a utilizar. Se esperaba que el equipo encargado de la construcción del banco aportara su experiencia en esas definiciones.

Se decidió ejecutar el proyecto siguiendo las prácticas de Scrum. Se definieron sprints de dos semanas con sesiones de planificación breves para identificar las prioridades en el backlog y un control de avance semanal. Una práctica que dio buen resultado fue hacer la revisión del sprint, y la planificación en la misma reunión entre el cliente y el equipo de desarrollo.

Otra de las ventajas de seguir un proceso ágil fue que permitió redefinir las prioridades a medida que el desarrollo avanzaba y se encontraban mayores oportunidades para agregar valor. Por ejemplo, el orden inicial del backlog se modificó al descubrir que la implementación temprana de algunos drivers permitiría cubrir más pasos de automatización.

Desafíos . En varias situaciones fue necesario ejecutar *spikes* para comparar alternativas y definir herramientas a usar. Por ejemplo, para determinar la estrategia de visión artificial para reconocer patrones. Una oportunidad de colaboración muy interesante se dio al favorecer las revisiones de código entre el equipo que desarrollaba el banco y el equipo que automatizaba las pruebas. Se incorporaron buenas prácticas de programación en la codificación de los pasos que favorecieron su reuso y redujeron la base de código a mantener.

4.2. Análisis de riesgo

Las pruebas automatizadas podrían enmascarar errores en el dispositivo que se está utilizando para llevarlas a cabo. La norma exige la documentación de los riesgos identificados con sus respectivos planes de mitigación y de las pruebas unitarias del software utilizado para tal fin.

4.3. Resultados

El banco de pruebas se utilizó para automatizar las pruebas de regresión en el desarrollo de un respirador. Como referencia, cada ciclo tomaba 4 días para la ejecución manual de las pruebas con 2 personas 100 % asignadas a la tarea. Tras la automatización, el esfuerzo de prueba se redujo 8 veces y solo algunas pruebas requirieron intervención manual. Por lo tanto, el primer resultado del proyecto fue la reducción del esfuerzo dedicado a la verificación. Por norma, un operador debe hacerse presente durante las pruebas de los equipos. Esto significa que la participación humana no se puede suplantar por completo.

Otra consecuencia de la ejecución del proyecto fue la capacitación del equipo de QA que incorporó nuevas habilidades al aprender a programar pruebas.

4.4. Lecciones aprendidas

A continuación, se listan algunas lecciones aprendidas durante el proyecto.

Casos de Prueba . Para la escritura de pruebas con Gherkin, se destacaron buenas prácticas que pueden correlacionarse fácilmente con las de programación:

1. Precondiciones para las pruebas: Identificar las condiciones comunes bajo las cuales debe ejecutarse una prueba individual o una suite de pruebas. Por ejemplo, en algunos casos, se debe configurar el equipo en cierto modo o realizar algunas calibraciones antes de correr una prueba. Según el *framework* de prueba elegido, estas condiciones se pueden aislar en pasos denominados *Setup* para una prueba individual o *Suite Setup* para una suite.
2. Restauración tras una prueba: Identificar las acciones necesarias para deshacer los efectos colaterales de una prueba y restaurar el estado inicial del dispositivo de prueba. Esto se suele denominar *Teardown* en los *frameworks* de automatización.
3. Redacción de pasos:
   a) Definir pasos en términos propios del dominio y con altos niveles de abstracción. Esta acción permite reutilizar dichos pasos en diferentes escenarios (por ejemplo, nuevas versiones de software), que sean portables a diferentes configuraciones. Por ejemplo, es preferible redactar un paso como “Seleccionar modo ‘Calibración’ ”, en lugar de “En el menú ‘Modos’, elegir la opción A”.
   b) Definir pasos parametrizables. Volviendo al ejemplo del punto anterior, si se pueden seleccionar diferentes modos de operación del equipo, definir un paso “Seleccionar modo” que reciba como parámetro los diferentes modos en que opera el equipo. Esto reduce el código duplicado y permite tener un inventario de pasos más mantenible.
4. Procesamiento de imágenes:
   a) Minimizar la cantidad de capturas de pantalla. Para leer múltiples valores presentados en pantalla, es conveniente hacer una única captura de la imagen, aplicarle los diferentes filtros y luego efectuar recortes para cada valor.

b) Definir constantes con las ubicaciones de las secciones a recortar para permitir una mayor portabilidad.

Hardware . Es importante documentar posibles problemas y soluciones que puedan sucederse durante la operación normal por desgaste (por ejemplo, relés), trabajo mecánico (piezas móviles) o conexionado. También se debe explicar cada componente del banco, su montaje, desmontaje y conexiones.

5. Conclusiones

El banco de pruebas resultó una herramienta de utilidad para reducir los tiempos de ejecución de las pruebas y sus costos.

Si bien el banco de pruebas fue construido para operar con un modelo particular de respirador, se espera poder adaptarlo para otros modelos y así acelerar los ciclos de desarrollo. Dicha reutilización se basa principalmente en la generación de pasos de pruebas con un alto nivel de abstracción, de manera que se requieran cambios mínimos de un producto al siguiente. La adaptación se llevaría a cabo reemplazando, cuando lo requiera, aquellas definiciones que dependen del respirador y que componen los pasos de nivel mayor de abstracción.

Como se mencionó anteriormente, el principal foco de la automatización fue la suite de pruebas de regresión. Sin embargo, se esperaría poder automatizar pruebas de sanidad que hagan una validación rápida de las funcionalidades básicas del aparato, o los controles posteriores a la fabricación para certificar su correcto funcionamiento.

Finalmente, se espera que el equipo de producto pueda incorporar algunas prácticas ágiles en su proceso como la integración continua y la ejecución de pruebas automatizadas en un pipeline de construcción. Una posibilidad radica en instalar nuevas versiones de software en el respirador de manera automática y ejecutar las pruebas con el banco desarrollado en este proyecto.

6. Agradecimientos

Agradecemos la revisión exhaustiva de Miguel Rizzo y sus contribuciones a la claridad técnica de este documento, a Mariana Borio, Carlos Hames y María Victoria Bruera por la revisión lingüística.

Referencias

1. https://www.eldiario.es/internacional/Ventiladores_0_1011749655.html
2. https://e-vent.mit.edu/
3. https://www.wsj.com/articles/rice-university-engineers-macgyver-an-inexpensive-ventilator-for-coronavirus-patients-11585832400
4. http://www.ku.ac.ke/component/k2/item/1655-k-u-makes-ventilators

5. https://eng.ox.ac.uk/news/oxford-and-kings-college-london-developing-prototype-for-rapidly-deployable-ventilator/
6. https://www.gov.uk/government/news/first-new-ventilators-to-roll-off-production-line-this-weekend-as-industry-answers-call-to-step-up-manufacturing
7. https://www.cnbc.com/2020/04/16/ge-ford-sign-336-million-federal-contract-for-ventilator-production.html
8. https://www.infobae.com/economia/2020/03/30/en-el-ojo-de-la-tormenta-la-historia-de-tecme-el-mayor-fabricante-de-respiradores-mecanicos-del-pais/
9. https://www.pagina12.com.ar/259358-coronavirus-la-historia-de-tecme-la-fabrica-argentina-de-res
10. https://www.nytimes.com/2020/03/29/business/coronavirus-us-ventilator-shortage.html
11. https://www.bbc.com/news/business-51914490
12. https://elpais.com/espana/madrid/2020-03-30/la-solucion-a-los-respiradores-made-in-spain-esta-en-mostoles.html
13. https://es.wikipedia.org/wiki/Respirador_artificial
14. https://en.wikipedia.org/wiki/Safety-critical_system
15. https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma/?c=N0046353
16. https://webstore.iec.ch/publication/2612
17. https://webstore.iec.ch/publication/6792
18. http://www.anmat.gov.ar/webanmat/normativas_productosmedicos_cuerpo.asp
19. https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfPCD/classification.cfm?ID=20
20. https://www.mddionline.com/developing-medical-device-software-iec-62304
21. https://webstore.iec.ch/preview/info_iec62304%7Bed1.0%7Den_d.pdf
22. https://en.wikipedia.org/wiki/System_testing
23. https://en.wikipedia.org/wiki/Regression_testing
24. https://glossary.istqb.org/en/search/
25. Dobslaw, Felix & Feldt, Robert & Michaelsson, David & Haar, Patrick & Neto, Francisco & Torkar, Richard. Estimating Return on Investment for GUI Test Automation Tools. (2019)
26. Garousi, Vahid & Mäntylä, Mika.. When and what to automate in software testing? A multi-vocal literature review. Information and Software Technology. 76. (2016) 10.1016/j.infsof.2016.04.015.
27. https://dannorth.net/introducing-bdd/
28. https://github.com/cucumber/cucumber-ruby
29. https://github.com/behave/behave
30. https://robotframework.org/
31. https://opencv.org/
32. https://github.com/tesseract-ocr/tesseract