Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Requerimientos No Funcionales en Ingeniería de Software: Avances y Desafíos

Tesis presentada para optar al título de
Licenciada en Ciencias de la Computación

Maria Gabriela Croce

Director: Diego Garbervetsky
Co-directora: María Virginia Brassesco

Buenos Aires, 2024

REQUERIMIENTOS NO FUNCIONALES EN INGENIERÍA DE SOFTWARE: AVANCES Y DESAFÍOS

Los requerimientos no funcionales (NFR) se reconocen como un factor muy importante para el éxito de un proyecto de software [1] [2]. Si los NFR no se abordan adecuadamente, pueden derivar en serios problemas en detrimento de la calidad del software desde aspectos que tienen que ver con la aceptación del software tales como *performance*, *security*, *scalability*, *usability* [3], como con la *maintainability* del mismo, es decir, facilidad de cambio, *reliability*, *testability*, etc [4]. De la misma manera que los requerimientos funcionales son clave y deben resolverse lo más pronto posible, la no resolución de NFR puede provocar un exceso de tiempo y costos para corregir errores de software [1].

Dado que los NFR se reconocen como elementos muy importantes para el éxito de los proyectos de software, es importante conocer cuál es el estado del arte en este área de la Ingeniería de Software.

En este trabajo partiremos de la situación presentada por Bajpai et al. en el año 2012 [5] y a partir de sus citas bibliográficas, autores derivados de las mismas, palabras clave en revistas y conferencias mencionadas en [5] y [4] recopilaremos aquellos artículos que tengan una cantidad de citas destacada demostrando ser relevantes en la actualidad. Analizaremos el impacto que pueden tener los NFR en la industria y en la vidas humanas a través de casos reales. Seleccionaremos algunas de las metodologías existentes hasta ese momento para clasificar y trabajar con NFR en el desarrollo de productos por su impacto u originalidad, las compararemos a partir de su contribución en el análisis de los NFR y analizaremos cómo continuó su desarrollo desde el 2012 hasta la actualidad. Por último, nos interesa ver qué sucedió desde el trabajo de Bajpai et al. hasta la actualidad, realizando para ello una nueva búsqueda de metodologías utilizando el mismo procedimiento.

Palabras claves: Ingeniería de Requerimientos, requerimientos no funcionales, GORE, NFR Framework, i* Framework

AGRADECIMIENTOS

Índice general

1. INTRODUCCIÓN

La primera medida de éxito de un sistema de software es el grado con el cual cumple el objetivo para el cual fue creado. La Ingeniería de Requerimientos (RE) es el proceso que nos ayuda a descubrir ese propósito: lo hace proporcionando documentación que facilita el análisis y luego implementación, así como también ayudando a individuar los *stakeholders* y sus necesidades. Este proceso, sin embargo, presenta una serie de dificultades: los *stakeholders* pueden ser muchos, estar distribuidos y tener requisitos que conflictúan entre sí; los requisitos pueden cambiar, no ser explícitos, difíciles de articular y su satisfacción puede estar sujeta a factores externos [6, pag. 1].

Dentro de la Ingeniería de Requerimientos, los requerimientos se clasifican tradicionalmente como requerimientos funcionales (FR) y requerimientos no funcionales (NFR), también llamados atributos de calidad. Los FR son aquellos que describen el "qué" de un sistema mientras los NFR describen el "cómo".

Los FR, su análisis y diseño, han recibido mucha atención desde su surgimiento en la Ingeniería de Software [7]; sin embargo los NFR, cuyo término fue acuñado en los años '60, siguen siendo un tema álgido hoy en día y a pesar de que ha habido mucha investigación al respecto, en general son ignorados y no hay estándares definidos [5]. Esto no es así para los FR, para los cuales surgieron estándares internacionales como el IEEE Std 1233 o IEEE Std 830, que permiten asegurar que las necesidades de los clientes son efectivamente registradas en el ciclo de desarrollo del software. En cambio, cuando se trata de NFR, nos encontramos con que los llamados "estándares" son en realidad taxonomías que presentan algunos NFR y donde el proceso de elicitación es vago (ejemplo ISO 9126) [2, pag. 2]; algunos de estos de hecho ni siquiera consideran el manejo de los NFR [8].

Tal es el debate sobre los NFR que en la comunidad de Ingeniería de Requerimientos ni siquiera hay consenso sobre su definición; Glinz en [9] de hecho hace un análisis sobre las diferentes definiciones disponibles en la bibliografía y sus problemáticas.

Los programadores conocen perfectamente la existencia de los NFR y su importancia, sobre todo en sistemas complejos, pero también suelen ignorarlos por falta de tiempo y documentación [5]. Dado que además la tendencia de la industria ha sido desarrollar primero los FR y luego incorporar los NFR uno a uno una vez que los FR ya fueron completados (*product-oriented approach*) [10], a esa altura del proyecto en general ya no hay tiempo, y terminan por no realizarse o realizarse parcialmente. Esta tendencia está cambiando, ya que la complejidad de los sistemas actuales hace que los NFR sean tan importantes como los FR; las demandas actuales de los productos de software hace que atributos de calidad como *security*, *portability*, *performance*, etc., se encuentren a la altura de las funcionalidades. Es más, ignorar los NFR ha llevado a más de una falla en el desarrollo de software, algunas de las cuales se volvieron muy famosas y que serán tratadas en la sección 2.1.

Muchas veces los NFR ni siquiera pueden ser satisfechos en su totalidad (al menos no en el mismo sentido que los FR) y por esto se dice que en vez de ser *satisfechos* deben ser *satisfascibles*: es decir, se espera que se hagan cumplir dentro de ciertos límites que se consideran aceptables y que se definirán de acuerdo al contexto y al requerimiento en cuestión [4].

En sistemas con mucho desarrollo de software, aumenta el riesgo y la dificultad, y

los desafíos ya no son tanto técnicos sino de gestión y requerimientos. La calidad de un sistema está dada no sólo por cumplir las funcionalidades requeridas por el cliente sino por otras características como la *usability*, *manteinability*, *reliability*, *performance*, etc. Esto hace que sea necesario invertir tiempo pensando en los NFR desde el diseño [2].

Las investigaciones (ni siquiera tan recientes) alertan que los NFR son los más costosos y difíciles de analizar [8], entonces, ¿por qué esta tendencia histórica a ignorarlos?

- algunas investigaciones demuestran que son las mismas características de los NFR las que producen que sean desestimados: el hecho de que sean subjetivos, abstractos y no uniformes hace que sean difíciles de modelar, verificar, testear y medir en comparación con los FR. Muchas veces hay contradicción entre los NFR (ejemplo *performance* y *security*) o generan nuevos FR y/o NFR [4];
- a veces la incorporación de los NFR hace que no puedan satisfacerse algunos de los FR [10];
- hay una gran falta de conocimiento por parte de los programadores sobre cómo trabajar con ellos y no mucha bibliografía al respecto; Boehm e In en [11] mencionan el caso del primer diseño del ARPANET, que fue descartado gracias a contar con un equipo altamente calificado que pudo identificar el problema y cambiar el rumbo del diseño a tiempo; pero esta no suele ser la realidad de la mayoría de los proyectos de software;
- los NFR tienen efectos *globales*, al contrario de los FR que son *locales*; suelen además especificar restricciones al software: *security*, *fault tolerance*, *performance*.
  Esta característica de transversalidad al proyecto ha hecho que algunos autores los traten como *concerns* (ejemplo [7]) o el mismo Glinz en su propio conjunto de definiciones [9, sección 4.2].

En esta tesis tomamos como punto de partida la situación presentada por Bajpai et al. en [5] en el año 2012 y por Mairiza et al. en [4] en el año 2010 (trabajo referenciado en el paper de Bajpai et al.) y, examinando también la bibliografía citada de ambos trabajos, analizamos el estado del arte de los requerimientos no funcionales hasta el año 2012 y de las metodologías surgidas para trabajar con ellos. En la segunda parte en cambio, hacemos un análisis del suceso de estas técnicas e investigamos qué sucedió desde el 2013 hasta la actualidad en término de nuevas metodologías.

1.1. Un poco de contexto

A pesar de que el término *requerimiento no funcional* surge en los años 1960, su definición precisa es ambigua. Glinz intenta explicar el por qué de esta confusión: diversos autores dan definiciones diferentes donde a veces no hay consenso sobre los conceptos que utilizan; los NFR son dependientes de su representación, haciendo que a veces el mismo requerimiento pueda ser funcional o no funcional; falta de consenso sobre cuál es el documento en el cual se registran los NFR [9]. Glinz dará una taxonomía de definiciones que incluye la de NFR basada en el concepto de *concerns*, para asegurar la independencia de la representación. Otros autores, Mairiza et al. [4], también realizan una investigación sobre las definiciones disponibles en la literatura para concluir que no existe una única posible

y citan los términos que más aparecen en cada una; clasifican a los NFR por aquellos que cuentan con una definición, los que no tienen definición pero tienen un atributo (es decir el elemento más importante del NFR), y los que no tienen ni definición ni atributos [4, fig. 4]; la investigación también revela que algunos tipos de NFR en realidad son expresados como atributos de otros, como por ejemplo *accuracy* que aparece como un NFR en la figura 3 del trabajo de los autores y luego como atributo de *reliability* en la tabla 2.

Cuando se habla de NFR, dada su complejidad y diversidad, son muy comunes las taxonomías. Ya Boehm e In en 1996 arman una base de conocimiento donde la táctica es categorizar y priorizar: mapean los *stakeholders* más usuales con el NFR más común para ese tipo de *stakeholder* [11, fig. 5], dan estrategias y efectos positivos/negativos para cada atributo de calidad [11, tabla 1] así como también estrategias a nivel arquitectura por cada atributo [11, fig. 6]. Mairiza et al [4] presentan distintas taxonomías: lista de tipos de NFR [4, fig. 3], lista de NFR más comunes por tipo de sistema [4, fig. 5] y por dominios de aplicación [4, tabla 3].

Otro concepto utilizado para clasificar a los NFR es el de *catálogos*, que constituyen una base de conocimiento resultante de experiencias previas y por ende promueven la reutilización de requerimientos. En general son representados por una estructura de árbol. Los catálogos pueden ser de diferentes tipologías: catálogo donde se visualizan el tipo de NFR junto con sus conceptos asociados y terminología; catálogo con métodos de implementación por tipo de NFR; catálogo de interdependencias y conflictos comunes entre NFR [1, sección 2.1]. En general los catálogos se utilizan en fases iniciales de un proyecto ya que sirven justamente para ayudar a identificar potenciales NFR, conflictos entre ellos o técnicas de implementación.

No existe un solo criterio a la hora de trabajar con requerimientos. Un enfoque que aparecerá en varios puntos de este trabajo, es el *Goal-oriented Requirements Engineering* (GORE), que sirve para elicitar, elaborar, estructurar, especificar, analizar, documentar, negociar y modificar requerimientos [12]. Una metodología que se basa en GORE se dirá que tiene un *goal-oriented approach*.

GORE pone a los objetivos en el centro de la escena. Los objetivos capturan, con distintos grados de abstracción, las metas que un sistema en cuestión quiere lograr. Los objetivos pueden expresar distintos tipos de *concerns*, sean funcionales o no funcionales, y pueden existir en el sistema *as-is* o *to-be* [12, pag. 2]. Mientras que un requerimiento en GORE es un objetivo del sistema *to-be* que recae bajo la competencia de un solo agente, los objetivos pueden necesitar de la cooperación de varios de ellos para realizarse.

Los objetivos en GORE pueden dividirse como funcionales (definen qué debe hacer el sistema, y cuáles son sus funciones y características) y no funcionales (describen las propiedades generales del sistema, es decir sus atributos de calidad) o también como objetivos duros (para los cuales se determina su satisfacción a través de técnicas de verificación) y blandos (para los cuales su satisfacción no puede ser determinada con precisión). Los objetivos se conectan entre sí a través de *links*, que indicarán distintos tipos de especializaciones entre el objetivo padre y sus hijos. Para los objetivos blandos se introduce el concepto de *satisfacibilidad*, que se refiere a encontrar soluciones que sean lo suficientemente buenas en vez de óptimas [1].

GORE propone técnicas para la verificación y validación de objetivos, contribuye en el proceso de elaboración de requerimientos, en el manejo y resolución de conflictos.

[1] Esta noción fue introducido por Herbert Simon [1, pag. 86]

Un enfoque ortogonal que suele utilizarse para clasificar el modo en que las metodologías afrontan la integración de los NFR, es catalogar por *process-oriented approach* o *product-oriented approach*. El primero se refiere a considerar los NFR a la misma altura de los FR y por ende considerarlos juntos en la construcción del software desde el análisis, en cambio el segundo, de carácter más tradicional, considerará primero los FR y una vez obtenidos éstos, se continuará por integrar los NFR al producto construído [10, pag. 1-2]. Esta clasificación será citada en varias de las metodologías tratadas en este trabajo.

1.2. Metodo de trabajo

Nuestro material de investigación parte de otros dos trabajos: el de Bajpai et al. en [5] que expone un estudio sobre los NFR realizado en el año 2012, y el trabajo de Mairiza et al. en [4] del año 2010 (trabajo por otro lado referenciado en [5]) que presenta un análisis sistemático de los NFR en la literatura en 3 aristas: definición y terminología; tipos; y NFR más usuales por tipo de sistemas/dominios de aplicación. Básicamente queremos ver:

1. qué sucede si no se toman en cuenta los NFR: para esto exponemos casos reales de la industria del software hasta el año 2012;

2. metodologías existentes para analizar, elicitar y modelar los NFR hasta el año 2012;

3. nuevas metodologías surgidas desde el año 2013 hasta la actualidad.

Para el punto 1 nos basamos principalmente en el trabajo [5] y sus citas bibliográficas para encontrar ejemplos de fracasos en la industria del software hasta el año 2012 y la gestión específica de los NFR en cada caso.; sólo el ejemplo 2.1.2 escapa a esta regla, que fue hallado realizando una búsqueda de las mayores fallas del software en la historia [13].

Para el punto 2 realizamos primero una búsqueda bibliográfica de las técnicas citadas en [5] y [4], y luego de la bibliografía/autores citados por esos trabajos. Una vez encontradas estas metodologías, se seleccionan algunas de ellas; todas como dijimos antes son anteriores al año 2013 (ya que el año del trabajo de Bajpai et al. es 2012) y se presentan en estricto orden cronológico de aparición para poder visualizar la influencia de unas en otras.

La elección de las metodologías elegidas se basa en el grado de innovación del método propuesto, el impacto que tuvo luego en la comunidad científica o en la generalidad del mismo. Así, los métodos *NFR Framework* y *i* Framework* son claramente los más disruptivos y en particular el *NFR Framework* sentó las bases para más de una de las técnicas que aparecen en la biblografía de ambos trabajos, como veremos luego; *NFR integrados a diagramas funcionales* hace un intento por integrar los NFR con los FR; y por último *Classpects como requerimientos no funcionales* busca integrar conceptos ya existentes como los aspects y las clases para modelar el mundo no funcional.

Por último, se realizará una comparación de las metodologías elegidas, junto con una explicación de la elección de las mismas.

En cambio en el punto 3 veremos cómo evolucionaron las metodologías presentadas en el punto 2, tarea para la cual realizamos un análisis cualitativo y cuantitativo de las mismas. Se presentan luego algunas metodologías de desarrollo surgidas posteriormente al 2012, con la intención de evaluar cómo evolucionó el análisis de los NFR hasta el día de hoy;

la selección de las propuestas elegidas fue hecha a partir de los autores de las metodologías de la sección 2.2, buscando nuevos trabajos de los mismos citados en revistas y conferencias mencionadas en las citas de [5]; por otro lado quisimos estudiar qué sucedió con las nuevas tendencias en la Ingeniería de Software, como big-data, Cloud, IOT, LLMs, blockchain, etc., con lo cual otro determinante en la búsqueda fue analizar metodologías que incluyan una o más de las mismas.

De los trabajos encontrados se seleccionaron tres que son presentados en la sección 3.2. La elección se basó en distintos criterios con el objetivo de lograr una cierta diversidad: el trabajo en 3.2.1 fue elegido principalmente por tener como autor a Lawrence Chung y dado que queríamos analizar una nueva propuesta de quien fue co-autor de un framework importante como el *NFR Framework*; el trabajo en 3.2.2 en cambio nos pareció interesante más allá de que tiene 2 autores en común con la técnica *NFR integrados a diagramas funcionales*: también usan los objetivos (duros y blandos) como los utilizó Yu en el *i* Framework*, conceptos que aparecen luego como parte del método presentado; en cambio con 3.2.3 quisimos ver algo distinto desde el punto de vista de autores y metodologías, y nos resultó de interés por su posible aporte al mundo Agile, que ha tenido tanta popularidad en los últimos años.

Con el fin de decidir un conjunto unívoco de atributos de calidad, en este trabajo tomaremos como base la figura [4, fig. 3] que da un lista posible de los tipos existentes de NFR en la bibliografía, resultado de una investigación llevada a cabo por los mismos autores. Por claridad, reportamos a continuación los NFR de esa lista que aparecerán como NFR también en nuestro trabajo:

1. Accuracy	2. Availability
3. Confidentiality	4. Correctness
5. Fault Tolerance	6. Integrity
7. Maintainaility	8. Operability
9. Performance	10. Portability
11. Reliability	12. Robustness
13. Safety	14. Scalability
15. Security	16. Testability
17. Usability	

Tab. 1.1: Lista de NFR utilizados

Cuando utilizamos un NFR lo haremos con su nombre en inglés ya que así aparecen en la bibliografía utilizada y también por mayor claridad conceptual (por ejemplo, sería muy fácil agrupar *safety* y *security* bajo "seguridad" cuando en realidad son dos conceptos bien diferentes).

1.3. Organización de la tesis

Esta tesis está organizada como sigue: en el capítulo 2 nos focalizamos en el estado de la situación hasta el 2012 y para ellos presentamos casos de fallas en la industria del software junto a su relación con los requerimientos no funcionales, metodologías existentes

hasta el mismo año y por último una comparación entre las técnicas presentadas; en el capítulo 3 mostramos un análisis cualitativo y cuantitativo de la evolución de los dos trabajos principales de esta tesis ([5], [4]) y de las metodologías utilizadas en el capítulo 2, junto con nuevas técnicas estrictamente posteriores al 2012 y una comparación entre las mismas; por último, en el capítulo 4 presentamos las conclusiones de nuestro trabajo.

2. REVISIÓN SISTEMÁTICA PREVIA AL 2012

En este capítulo tomamos como trabajos de referencia a [5] y [4]. A partir de ellos y de sus citas bibliográficas, se buscará analizar principalmente dos cosas: las metodologías de desarrollo para los NFR hasta el año 2012 incluído (año de publicación del trabajo de Bajpai et al.) así como también ejemplos de fracasos en la industria del software hasta el mismo año y cómo fue la gestión de los NFR en cada caso.

Por último, se realizará una comparación de las metodologías elegidas.

2.1. Ejemplos de una mala gestión de los NFR

A través de los años se han conocido y documentado diversos casos de fallas en sistemas de software que han sido causados, en mayor o menor medida, por desestimación de los NFR, provocando incluso en algunos casos la pérdida de vidas, en otros pérdidas millonarias. Describiremos aquí cuatro de ellos, con fecha previa al 2013, junto a los NFR que fallaron en cada caso.

2.1.1. Therac-25

Un caso emblemático fue el del Therac-25 [14]. El Therac-25 era una sistema crítico para la seguridad que involucraba radioterapia controlada por una computadora, y provocó al menos 6 accidentes por intoxicación por radiación entre 1985 y 1987; venía a reemplazar a las versiones anteriores Therac-6 y Therac-20, ambas con poca intervención del software. El fabricante era *Atomic Energy of Canada Limited* (AECL) y los productos se usaban en Canadá y EE.UU.

El Therac-6 era un acelerador que se usaba sólo para rayos X, en cambio el Therac-20 era un acelerador de modo dual (modo electrón o modo fotón/rayos X). Ambos usaban una minicomputadora, el DEC PDP-11, que tenía funciones limitadas pero hacía el control del hardware más "amigable"; el hardware por otro lado era standalone.

Con la introducción del Therac-25 hubo varios cambios en cuanto a funcionalidad del software se refiere:

- cambió el concepto: los dos primeros Therac eran máquinas que usaban software para algunas rutinas, el Therac-25 era una máquina controlada por el software;
- ya no era más una máquina standalone;
- el software tenía más responsabilidades que en las dos versiones anteriores: sacaron mecanismos de seguridad por hardware (*interlocks*, circuitos para monitorear el haz de electrones) y los pasaron al software;
- servía para ambos modos como el Therac-20, pero era más pequeña y económica.

Se sabe también que el Therac-25 usaba una computadora DEC PDP-11 como las dos anteriores y su software fue hecho por un solo programador en lenguaje ensamblador. Esta

persona no se pudo identificar.

El Therac-25 tenía una placa giratoria donde el operador podía elegir entre 3 modos:

- modo escaneo: servía para posicionar correctamente al paciente. En este modo se iluminaba la zona a tratar, se posicionaba correctamente el colimador[1] y no se esperaba ninguna radiación;
- modo electrón: también llamado modo de baja potencia, usaba 4 imanes para escanear el área a tratar;
- modo fotón o rayos X: también llamado de alta potencia, necesita de una potencia mucho más alta que el modo electrón (puede llegar hasta +100 veces). Esta potencia es luego dosificada por un filtro que hace que el tratamiento sea uniforme en el área a tratar.

El riesgo se puede presentar si la placa giratoria no está en la posición correcta; este es el problema general de las máquinas multipropósito. Respecto al Therac-20 que también era un acelerador con modo dual, el Therac-25 presentó errores al operarlo que no se habían manifestado antes:

- el operador quería usar el modo electrón, pero el resultado real era que quedaba activado el modo fotón sin el objetivo en la posición correcta;
- el operador quería usar el modo luz para alinear el paciente con el colimador, pero quedaba activado el haz de electrones.

Al contrario a lo que los ingenieros de AECL pensaron inicialmente esto se debió a más de una falla:

- se asumió que las rutinas de software provenientes del Therac-20 no presentaban defectos; esto era incorrecto, ya que la secuencia de pasos que llevaban a los errores en la operación del Therac-25 era (más de) una *race-condition*[2] heredada de la versión anterior del software y al *overflow* de una variable;
- algunos los defectos existían desde antes pero no se habían reportado accidentes. ¿Por qué? Porque los *interlocks* y circuitos por hardware actuaban justamente como mecanismos de seguridad para prevenir que la placa giratoria quedara mal posicionada.

Este caso se conoce como uno de los accidentes más críticos en la historia del software. Algunos de los factores que contribuyeron a los accidentes del Therac-25 son:

- falta de testing: en 1983 AECL hizo un análisis de seguridad del Therac-25 donde, aparentemente, excluyeron al software;
- fallas de seguridad: en versiones anteriores tenían mecanismos de seguridad por hardware que fueron pasados al software, sin agregar un mecanismo de redundancia;

[1] https://en.wikipedia.org/wiki/Collimator

[2] Por definición, una *race-condition* es una condición de un programa en la que su comportamiento depende de la sincronización relativa o la intercalación de varios subprocesos o procesos. https://en.wikipedia.org/wiki/Race_condition

- mensajes confusos y frequentes: los operadores del acelerador declararon que los mensajes poco claros que mostraba la pantalla del Therac-25 sucedían cotidianamente y no estaban explicados en el manual de usuario;

- sospecha de no buenas prácticas de ingeniería de software;

- mala gestión, que se vio en la falta de procedimientos de la empresa: incluso luego de reportar los primeros incidentes AECL no fue capaz de hacer un seguimiento ordenado de los casos.

Gestión de los requerimientos no funcionales

Cambiando el enfoque a cómo fueron gestionados los requerimientos no funcionales, podemos identificar como no satisfechos al menos los siguientes:

- *safety*: este era un sistema crítico para la vida que falló estrepitosamente. Miller, director de la *Division of Standards Enforcement* (CDRH) hablando particularmente de las lecciones aprendidas con los accidentes del Therac-25 en 1987, enfatiza que el diseño de software debe ser seguro y tener métodos de prevención para minimizar los riesgos [14, pag. 21].
  Algunas de las decisiones que fueron en contra del *safety* por parte de AECL fue sacar los mecanismos de seguridad por hardware dejando al software como primera y única barrera de contención ante cualquier falla, así como también haber excluído al software del *safety-test* en forma de *fault-tree* que realizaron en 1983.
  Eventualmente, luego del último accidente reportado en 1987 y de las exigencias de la *Food and Drug Administration* (FDA) que demandó varias revisiones al Therac-25, AECL fue obligado a realizar un *safety analysis* exhaustivo que incluyó esta vez un análisis del software. Luego de esto AECL anunció varios cambios, como por ejemplo volver a incorporar los *interlocks* por harware como backup al control por software de ambos modos electrón y fotón;

- *reliability*: el sistema presentó al menos 6 fallas en 2 años, que provocaron la muerte de 3 pacientes y agravaron la condición de salud de los demás [14, pag. 19, 21].
  En el mismo *safety analysis* mencionado antes un consultor externo realizó un análisis cualitativo de la confiabilidad y recomendó algunos cambios en el programa para corregir deficiencias y mejorar la confiabilidad general del producto [14, pag. 21];

- *maintainability*: el software del Therac-25 estaba escrito en lenguaje ensamblador que consideramos no es fácilmente mantenible.
  Se infiere también de lo descripto por Leveson et al. que la falta de mantenibilidad quedó reflejada luego de las varias actualizaciones de software que AECL tuvo que realizar cada vez que los hospitales reportaban un nuevo caso de sospechada intoxicación por el uso del acelerador.
  En el reporte final del *safety analisys* de 1987 se recomienda a AECL hacer más foco en la mantenibilidad del software; en particular, se recomiendan 2 o 3 nuevas versiones de software antes de que todos los cambios requeridos puedan considerarse completos [14, pag. 19];

- *robustness*: el Therac-25 era más pequeño y más barato que el Therac-20, y al quitar mecanismos de seguridad por hardware sin agregar redundancia, el resultado fue una máquina menos robusta;

- *usability*: los operadores del Therac-25 tuvieron problemas para interpretar algunos de los mensajes que aparecían en la pantalla, cosa que declararon sucedía cotidianamente. Los mensajes además no estaban explicados en el manual de usuario;

- *traceability*: el Therac-25 no contaba con un historial documentado de desarrollo, producción o testing. Esto se vio reflejado cuando iniciaron las demandas contra AECL.

Un error que, quizás, contribuyó a los accidentes del Therac-25 es confundir *reliability* con *safety*: la ausencia de errores de software en el pasado no implica el mismo comportamiento en el futuro [14, pag. 21]. En este sentido se puede inferir que AECL no debería haber asumido que el software era confiable y por ende excluirlo del testing.

Tampoco debería priorizarse *usability* sobre *safety*: los operadores habían manifestado su descontento en el pasado por deber ingresar datos más de una vez en el acelerador antes de poder comenzar un tratamiento ya que les consumía mucho tiempo, pero hacer la máquina lo más amigable y simple posible puede conflictuar con este objetivo [14, pag. 23].

2.1.2. Instituto Oncológico Nacional (ION)

Otro accidente más reciente de un sistema crítico para la seguridad es el del Instituto Oncológico Nacional (ION) en Panamá (año 2000) [15]. El departamento de radioterapia de ION usaba un sistema de planificación de tratamiento (TPS); el software usado por el TPS en ese momento era el TPS RTP/2 de *Multidata System*, versión 2.11, hecho por *Multidata Systems International*, una compañía estadounidense.

En un tratamiento las dosis se dividen en fracciones diarias, que se aplican al paciente en diferentes días por un determinado período de tiempo (en general varias semanas). Cada fracción se compone de "regiones", donde en cada región se dispara la radiación al tumor desde distintas direcciones. Cada región a su vez tiene "bloques", que son las zonas de tejido sano que queremos proteger de la radiación.

El TPS define un tratamiento: calcula el tiempo, la distribución de las dosis, las zonas a irradiar y aquellas a no irradiar (las que son cubiertas por los bloques). El software usado para el TPS tenía 4 modos, pero nos vamos a enfocar en dos de ellos:

1. *Dose Chart Calculator*: se usa para calcular el tiempo de tratamiento para aplicar una dosis determinada en un punto de prescripción;

2. *External Beam*: se usa cuando además de lo anterior se quieren generar otros tipos de distribuciones. Sólo permite 4 bloques.

El problema se presentó para los casos de cáncer en la región pélvica (próstata, cuello uterino, colon). En general los médicos usaban una técnica donde se definían 4 regiones; si algún paciente necesitaba más regiones para su tratamiento, éste se dividía en 2 días y se trataban las regiones alternadamente. Un ejemplo de las zonas a cubrir por los bloques puede verse en la figura 2.1.

Para algunos tratamientos del cuello uterino se sabía que se necesitaba cubrir, además de las 4 zonas usuales, una extra central (como puede verse en la figura 2.2), para cubrir partes críticas involucradas en la cirugía. Esto estuvo bien mientras no se pidió calcular las distribuciones, cosa que eventualmente sucedió.

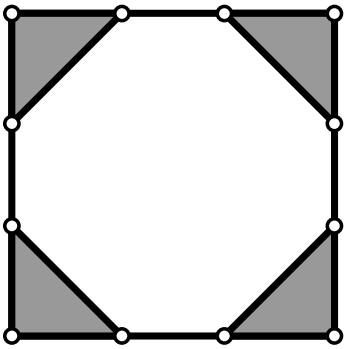

Fig. 2.1: Zonas de tratamiento usando la "técnica de la caja". Figura basada en [15, fig. 4]

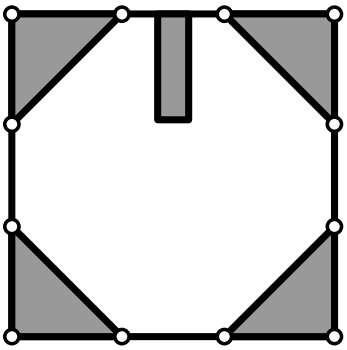

Fig. 2.2: Zonas de tratamiento con la 5ta zona central agregada. Figura basada en [15, fig. 4]

Los médicos encontraron un modo de engañar al TPS para digitalizar[3] los 5 bloques que necesitaban en uno solo: la técnica alternativa consistía en, dada una figura como la de 2.1, recorrer primero los vértices internos y luego los externos. De esta manera se cubría el área que querían proteger de la radiación y se podían definir más de 4 bloques libremente. La computadora, que mostraba el diseño final en la pantalla, parecía captar correctamente el requisito de los médicos. Lo que éstos no sabían era que el algoritmo de cálculo cambiaba de acuerdo al modo en que se seleccionaban los vértices: elegirlos todos en el mismo sentido (sea horario o antihorario) daba un resultado de +100 % de la radiación deseada, en cambio si se elegían primero los vértices internos en un sentido y luego los externos en el contrario se llegaba al resultado correcto; ambas situaciones pueden verse en 2.3. Para los médicos era más cómoda esta manera de diseñar los bloques, con lo cual en algunos casos la usaron incluso cuando sólo se necesitaban 4 bloques.

El error fue descubierto varios meses después, cuando los médicos, sospechando de intoxicación por radiación, hicieron una simulación del tratamiento y midieron la dosis de radiación aplicada.

Varios componentes llevaron a este resultado fatal para los pacientes:

- el manual de usuario del TPS especificaba que podían definirse hasta 4 bloques para la opción *External Beam*, pero no da instrucciones sobre cómo diseñar el contorno de los bloques;
- el software permitía que el mismo bloque (desde el punto de vista del diseño final

[3] Se llama así al proceso de ingresar las coordenadas de los vértices de un bloque en el TPS.

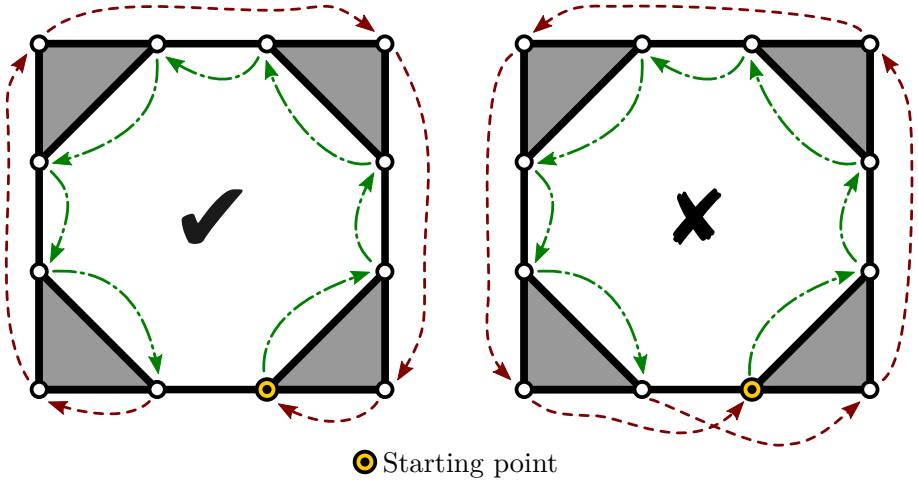

Fig. 2.3: Las dos formas de armar la misma figura. Figura basada en [15, fig. 6-7]

para el usuario) pudiera crearse de más de una manera, pero el área se calculaba de distintas maneras;

- no había ninguna advertencia para el usuario, en el caso en que creara un bloque de una manera que no era la indicada en el manual de usuario;
- el mismo manual explicita que el cálculo es simétrico respecto a si los puntos son ingresados en sentido horario o antihorario.

Gestión de los requerimientos no funcionales

Hablando de requirimientos no funcionales, observamos dificultades en la gestión de los siguientes:

- *safety*: como el Therac-25 este era un sistema crítico para la vida. El TPS RTP/2 de *Multidata System*, versión 2.11 se usó con la técnica alternativa en 28 pacientes entre agosto del 2000 y marzo del 2001, de los cuales 8 fallecieron y 5 de ellos por comprobada intoxicación por radiación. Se puede deducir que el requerimiento de *safety* fue subestimado y no se dieron mecanismos de control alternativos;
- *reliability*: el sistema no fue capaz de operar sin presentar fallas frequentes;
- *usability*: la interfaz del TPS mostraba el mismo diseño final, sea cuando los bloques se ingresaban correctamente que cuando no; no había ninguna advertencia si los bloques se digitalizaban de un modo que no era el esperado; en el manual de usuario estaba escrito que se podían digitalizar hasta 4 bloques pero no indicaba cómo [15, pag. 13, 14];
- *correctness*: tomando como diseño final el presentado al usuario en la interfaz del TPS, el algoritmo no daba el mismo resultado para diseños iguales, concluyendo por lo tanto que este requerimiento no estuvo bien contemplado.

2.1.3. LAS

En el año 1992 el Servicio Nacional de Salud (NHS) de Londres decidió digitalizar su servicio de ambulancias [16]. El sistema de envío de ambulancias (LAS) es el responsable de entender la importancia de cada llamada y asignar los recursos disponibles de acuerdo a eso, y de monitorear el progreso de la respuesta a la llamada. La mejora a implementar era reemplazar el sistema manual con un sistema de envío asistido por computadora, que iba a contar con un sistema automático de localización de vehículos (AVLS), para entender dónde estaba cada ambulancia y terminales de datos móbiles (MDTs), para comunicarse con las ambulancias.

El proceso manual que querían reemplazar llevaba menos de 3 minutos desde que se recibía una llamada hasta que se localizaba una unidad (que podía hacerse contactando a la central de ambulancias correspondiente o pasando las instrucciones por radio al operador si es que la ambulancia ya estaba en camino). El éxito del nuevo sistema dependía casi en su totalidad de la tecnología y precisión del sistema de geolocalización.

Cuando se puso en funcionamiento lo que sucedió fue lo siguiente:

- el AVLS no era capaz de mantener el estado y ubicación de las ambulancias, lo que generó errores en la base de datos e hizo que el envío de las unidades no fuera el óptimo así como también que se asignara más de una unidad a la misma llamada;
- como consecuencia de lo anterior, hubo un gran número de excepciones que hicieron más lento el sistema. La lista de excepciones crecía rápidamente y aquellas no respondidas se repetían, generando un efecto avalancha;
- el personal de ambulancia tuvo dificultades para entender las MDTs, estaban bajo mucha presión y por ende se retrasaban en actualizar el estado correcto de su unidad;
- los usuarios repetían sus llamadas ya que no obtenían una rápida respuesta, lo que sólo generaba más tráfico.

Esto llevó a algunas situaciones dramáticas: una unidad llegó cuando el paciente ya había muerto, en otro caso llegó 11 horas después de una llamada por un derrame cerebral y cuando ya hacía 5 que el paciente estaba en el hospital. Finalmente, el NHS volvió al sistema manual.

Algunos de los problemas de software encontrados por la investigación posterior fueron:

- el sistema propuesto no representaba fielmente la estructura organizativa del servicio de ambulancias;
- usuarios y clientes clave que no fueron involucrados en el proceso: por ejemplo personal de ambulancias;
- software incompleto y no debidamente testeado;
- el enfoque de la implementación era de alto riesgo;
- el sistema no era robusto y era lento.

Gestión de los requerimientos no funcionales

Se encuentran los siguiente problemas referidos a los requerimientos no funcionales:

- *performance*: el gran número de excepciones generadas que se repetían y crecían, ralentizó el sistema. Como consecuencias directas se pueden mencionar la ambulancia que llegó cuando el paciente ya había fallecido hacía horas y otra que llegó 5 horas después que el paciente fuera a un hospital por sus propios medios [16, pag. 2];
- *availability*: el sistema no respondía por tiempos muy largos [16, pag. 2];
- *robustness*: muy poco después de ponerse en funcionamiento el sistema empezó a generar excepciones y no poder responder en tiempo y forma a los pedidos [16, pag. 2];
- *usability*: el paper menciona la dificultad de los operadores de ambulancias para usar las MDTs dado que la interfaz de usuario era bastante pobre, por lo tanto podemos inferir que la usabilidad del sistema no estaba a la altura del proyecto;
- *integrity*: los datos en la base de datos no eran correctos;
- *scalability*: no era capaz de manejar una situación con una cantidad de llamadas considerada dentro de los parámetros normales;
- *fault tolerance*: el tiempo de respuesta a las llamadas era un punto crítico en este sistema; por las situaciones presentadas en este trabajo se puede inferir que no estaba preparado para soportar fallas y continuar operando.

2.1.4. El bug FDIV de Pentium

Los chips 486DX y Pentium de Intel incluían una FPU, que era una novedad respecto a CPUs anteriores; la ventaja que daba la FPU integrada era la posibilidad de resolver cálculos complejos en menor tiempo.

La FPU de Intel usaba un algoritmo radix-4 SRT para hacer divisiones: para esto necesitaba tener una tabla de valores guardada, llamada "tabla de división". Esta tabla, que había sido mal cargada en los circuitos de la FPU, tenía 5 entradas faltantes por cada 1000. El resultado fue que algunas divisiones contenían errores: el peor caso que se conoció fue el de 4195835/3145727, que con la FPU de Pentium devolvía 1.33374 cuando el resultado correcto es 1.33382 (error del 0.006 %).

Inicialmente el problema fue negado por Intel, luego desestimado hasta que llegó a la prensa. Eventualmente cambiaron todos los procesadores afectados, con un costo total para la empresa de USD 475 millones [17] [18].

Gestión de los requerimientos no funcionales

Este es el único bug de hardware de los presentados en esta sección. Desde el punto de vista de los requerimientos no funcionales, se puede inferir que la FPU no satisfacía el requerimiento de *correctness*, dado que las entradas cargadas en la tabla de división no estaban completas, así como también la *accuracy* de los resultados era incorrecta.

Podemos también concluir, dado que para Intel la FPU era una tecnología nueva, que probablemente la presión por salir al mercado antes que la competencia hizo que se vea afectada la *performance*. El potencial impacto, si los chips 486DX o Pentium se usaban en un sistema crítico, era enorme.

2.1.5. Switches 4ESS de AT&T

La empresa *AT&T Corporation* fue una de las empresas de telecomunicaciones más grandes e importantes de Estados Unidos; en su apogeo en las décadas del '50 y '60 empleó a más de un millón de personas. Durante gran parte del siglo 20 mantuvo el monopolio del servicio telefónico en Estados Unidos y Canadá.

En enero de 1990 el centro de operaciones de Nueva Jersey detectó un malfuncionamiento generalizado de la red, que estuvo caída por 9 horas, con un porcentaje de fallas en las conexiones de llamadas del 50 %: AT&T perdió más de USD 60 millones debido a llamadas que no pudieron llevarse a cabo, y hubo otras consecuencias colaterales como por ejemplo los más de 500 vuelos demorados que afectaron a 85.000 personas, o las pérdidas no calculables de aerolíneas, hoteles, etc. que dependían de la red telefónica para sus sistemas de reservas y no pudieron operar durante mínimo ese tiempo [19].

La red de larga distancia de AT&T era supuestamente un modelo de eficiencia; el sistema en general completaba el routing de una llamada en pocos segundos. Además la empresa hacía mucho énfasis en su confiabilidad y seguridad: el incidente de hecho fue una sorpresa porque AT&T era famosa por su riguroso proceso de testeo.

El problema se debió a una sola línea de código en una actualización de software hecho en *todos los switches de la red*; el pseudocódigo puede verse en 1:

Alg. 1 Pseudocódigo de los switches de AT&T (código en *C*)

```
1:  while not_empty_buffers do
2:      ...
3:      switch message
4:          case incoming_message
5:              ...
6:              if write_buffer_is_empty then
7:                  ...
8:              else
9:                  break          → línea con el defecto
10:             end if
11:             process incoming message, additional settings
12:             break
13:         endCase
14:     endSwitch
15:     optional parameter work
16: end while
```

La intención de la sentencia **break** en la línea 9 era salir del bloque **if-then-else** y continuar en la línea 11, pero en cambio salía de la sentencia **switch** y continuaba en la línea 15 que sobreescribía los datos, lo que causaba que el switch se reseteara.

En restrospectiva en la falla de AT&T se aprendieron una serie de lecciones [19]:

- el lenguaje elegido es importante: un compilador de C más estricto o utilizar otro lenguaje hubiera impedido que se introduzca el defecto del **break**;
- un hardware más tolerante a fallas podría haber ayudado;
- un software más tolerante a fallas, que no necesariamente apagara los switches ante el primer problema.

Gestión de los requerimientos no funcionales

En cuanto a requerimientos no funcionales en cambio, no se satisfacían al menos los siguientes:

- *robustness*: el sistema no era tolerante a fallas, lo que se pudo ver cuando en cada switch se llegaba a la ejecución de la línea con el defecto, que provocaba la sobreescritura de datos y el reseteo del mismo;
- *availability*: si tomamos como parámetro un mes calendario, la red de AT&T estuvo caída por 1.25 % de ese tiempo. Dada la criticidad del sistema consideramos que se debería haber requerido una *availability* de al menos un 99 %;
- *operability*: la nueva actualización no se operó de manera correcta, conclusión que se puede deducir de la decisión de AT&T de actualizar todos los switches de la red al mismo tiempo en vez de a un pequeño subconjunto por vez.[4]

En este caso podría hablarse también del no cumplimiento de *deployability* o *upgradability*; ninguno de ellos está presente en el análisis hecho en [4], ni como requerimientos no funcionales ni como atributos, razón por la cual no los incluímos entre los NFR no gestionados.

2.2. Tipos de metodologías de gestión de NFR

En esta sección presentamos cuatro metodologías surgidas con anterioridad al año 2012; los criterios para la selección de las mismas fueron explicados en 1.2.

2.2.1. NFR Framework (1992)

Una metodología que tuvo bastante éxito fue el *NFR Framework*, de Chung et al. [1] [20].

El *NFR Framework* es un *process-oriented approach*, que como mencionamos en 1.1 considera a los NFR como elementos reales en el desarrollo del sotware y por ende son tomados en cuenta junto con los FR para guiar la construcción del software desde el inicio[5]; es también un *goal-oriented approach* e introducen otro concepto de objetivo, el *softgoal*, una entidad flexible que puede llegar a no ser satisfacido en su completitud y donde las decisiones de diseño pueden influirlo positiva o negativamente [1]. El *NFR Framework* tiene además un enfoque *cualitativo*:[6] los autores lo decidieron así dado que pensaron que tener que medir un producto parcial en modo cuantitativo no resultaba tan simple.

Uno de los aportes más significativos de este framework es ofrecer una estructura para representar el proceso de razonamiento y diseño usando grafos, llamado el *Softgoal*

[4] Si bien [4] no define el requerimiento *operability*, lo entendemos como la capacidad de un sistema de realizar actualizaciones de software con posiblidad de revertir los cambios.

[5] La unión entre FR y NFR se encuentra fuera del alcance de esta tesis.

[6] Ortogonalmente, los requisitos no funcionales se pueden clasificar en cuantitativos y cualitativos, clasificación que se refiere al tipo de métricas que se usan para medir el grado con el cual un software satisface un requerimiento no funcional [1, pag. 3].

Interdependency Graph (SIG)[7]. La creación de este grafo no es secuencial: es posible y esperable que el programador necesite hacer más de una iteración en el grafo durante la fase de diseño, refinando los requerimientos, agregando operacionalizaciones, etc.

Un concepto importante que usa el *NFR Framework* es el de *catálogos*, que son conocimiento adquirido por los programadores de experiencias previas: acá estarán catalogados los NFR, técnicas e *interdependencias* entre objetivos blandos.

Para construir el SIG, se parte de una primera iteración donde se identifican los NFR que entendemos deberían ser *satisfacibles* por nuestro sistema. Cada objetivo blando tiene un *tipo* para indicar de qué NFR en particular se trata y un *tópico* que representa el argumento[8]; estos son luego descompuestos en otros, de acuerdo a su *tipo* o su *tópico*: se dice que los hijos *contribuyen* (todos o algunos) a satisfacer al padre (que está en un nivel más alto), y que el padre es *refinado* en objetivos blandos. En cualquier momento los programadores pueden valerse de los catálogos para consultar técnicas o tomar decisiones.

Una vez que un objetivo blando fue refinado hasta que se considera lo suficientemente detallado, se proponen posibles *operacionalizaciones*: otro tipo de objetivo blando que representa técnicas de implementación.

Por último, se agregan también *enunciados* (del inglés *claim*), que son el tercer y último tipo de objetivos blandos propuestos por el SIG y que tienen dos posibles usos: pueden representar *afirmaciones* (a favor o en contra de algo) o *compromisos* entre objetivos (estos últimos en general surgen para solventar los problemas generados por las correlaciones); los enunciados tienen un aporte positivo o negativo hacia otros objetivos/interdependencias, que se representa también en el gráfico.

Cuando las iteraciones terminaron, se eligen entre los objetivos blandos aquellos que quedarán seleccionados como parte del diseño final. Un ejemplo de un SIG donde pueden observarse los conceptos introducidos puede verse en la figura 2.4.

El *NFR Framework* fue muy bien aceptado y usado posteriormente por otros autores para crear nuevas técnicas; algunos ejemplos sólo dentro del marco de esta tesis son Cysneiros et al. [21], Cysneiros et al. [8], Lin et al. [22], Rajan et al. [23].

Limitaciones y ventajas

La ventaja más obvia del *NFR Framework* es que da un framework concreto para analizar los NFR e integrarlos como parte fundamental en el diseño del software. Esta es una herramienta integral: tiene en cuenta no sólo el proceso de elicitación de requerimientos no funcionales, sino sus posibles conflictos, compromisos a tomar entre distintos objetivos, priorización entre operacionalizaciones y, aunque mínimamente, también trata la integración con requerimientos funcionales. Los autores dan asimismo versiones del framework especializadas en distintos NFR, como por ejemplo *accuracy* y *performance*, dos de los NFR más utilizados.

Chung et al. mencionan varias veces que es necesario el aporte de un programador en la toma de decisiones en todas las etapas, lo cual puede ser una ventaja o desventaja dependiendo del proyecto: por un lado facilitará la unión entre requisitos funcionales y no funcionales ya que probablemente si los programadores están a cargo de ambas áreas utilicen un vocabulario común, pero no en todos los proyectos ambos análisis serán hechos por el mismo equipo de trabajo (es más, podría ser que no sean programadores los encargados

[7] Softgoal u objetivo blando es un término que se utiliza para referirse a requerimientos que son satisfacibles en vez de satisfechos.

[8] La raíz del SIG se representará en la forma tipo[tópico].

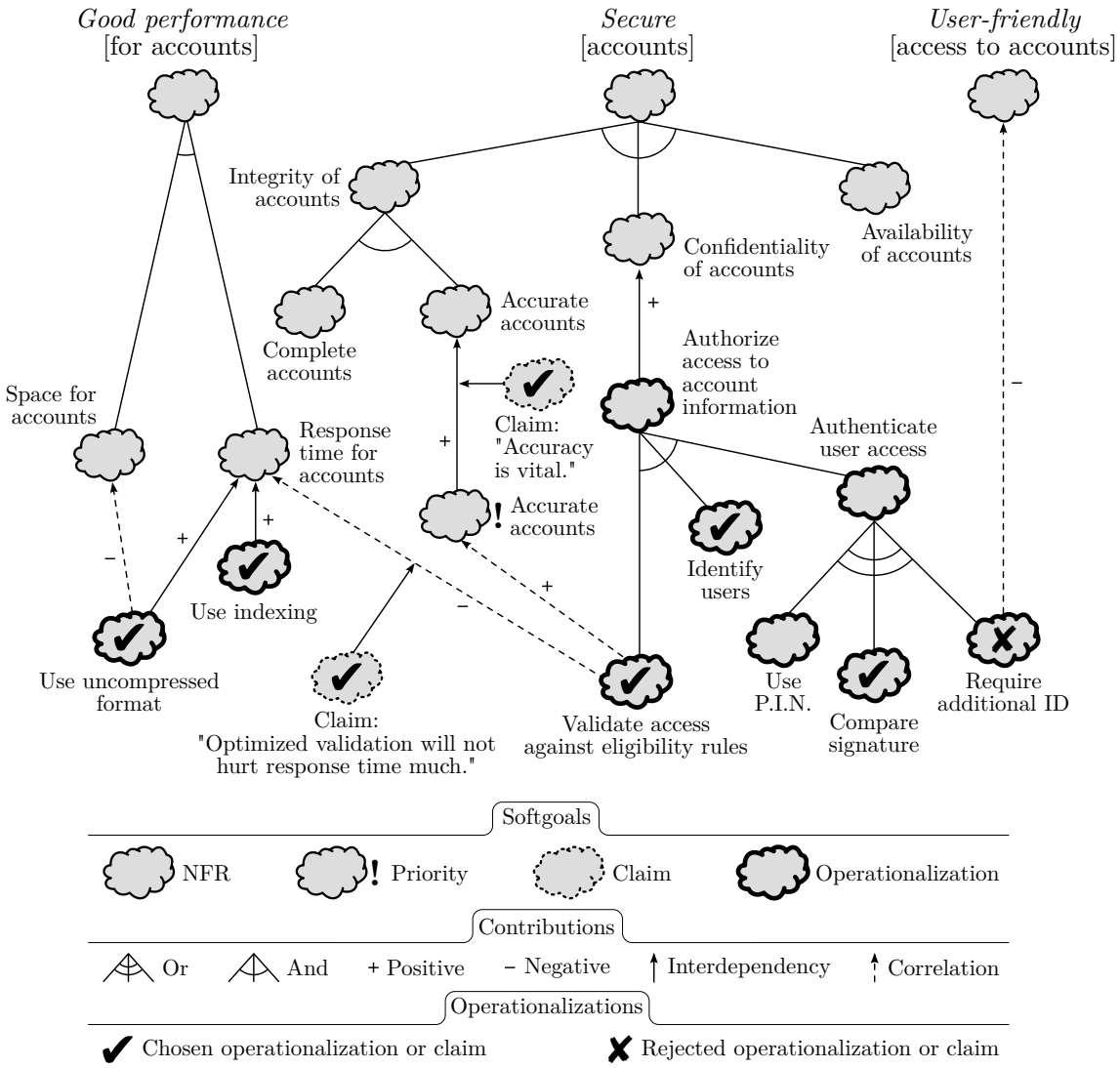

Fig. 2.4: SIG tomado de ejemplo de [1, fig. 2.12]

de esta tarea), lo que puede entorpecer el desarrollo del proyecto.

El *NFR Framework* no presenta herramientas propias para modelar los FR sino que se basa en trabajos de otros autores para esta actividad [1, pag. 88], explicitando que el framework puede integrarse con la mayoría de ellos; dado que no presentan estudios ni resultados al respecto y que lo expresan como una "sensación" inferimos que es una conclusión empírica que debería ser usada con cautela. Por otro lado los autores concluyen que en realidad el framework puede ser utilizado para todo tipo de requerimientos ya que "todos los requerimientos pueden ser vistos como objetivos blandos que compiten entre ellos y que pueden flexibilizarse para dar lugar a otros requerimientos" [1, pag. 396], conclusión que también debería ser respaldada con casos de estudio.

El hecho de que presente un solo método gráfico para analizar los NFR pensamos que es algo positivo; sabiendo que los NFR no solamente son difíciles de elicitar sino que también son difíciles de visualizar, el tener la información ordenada en una estructura de grafo

permite al programador encontrar más rápidamente los conflictos, contribuciones positivas o negativas, objetivos faltantes, etc. Por otro lado, están dando una notación nueva que no existe para los FR, que demanda además un buen conocimiento de lógica (que quizás no todos los programadores tienen), y esto dificultará seguramente la integración entre los dos tipos de requerimientos[9] [10, pag. 2].

Creemos también que una desventaja de este framework es que la semántica para cada tipo de NFR debe ser dada individualmente, aumentando así la dificultad para extenderlo. Para Cysneiros et al. el framework no pone tanto énfasis en cómo gestionar los conflictos entre requerimientos funcionales y no funcionales [21]. Según [22, pag. 3] en cambio, el *NFR Framework* no está a la altura de las demandas actuales del software (año 2008) y carece de herramientas importantes para el desarrollo del mismo, como herramientas de modelado de ingeniería de software o soporte para UML.

Los autores mencionan que serían necesarias algunas extensiones para permitir la posiblidad de cambios en el framework, que al momento de escribir el libro no existen [1, pag. 413].

2.2.2. i* Framework (1995)

Otra de las metodologías exitosas de la época fue el *i* Framework*; este trabajo inicia con la tesis de doctorado de Yu en 1995[10] [24] y se consolida en el libro del mismo autor en el 2010 [25]. Para esta tesis se utilizó como fuente principal el análisis realizado por Franch et al. en [26]; los trabajos [24] y [25] fueron utilizados en menor grado.

El *i* Framework* cambia la manera de ver al mundo del enfoque tradicional a una visión *social* que traslada a la Ingeniería de Requerimientos: el *mundo social* es presentado como desconocido e incontrolable y tiene *intencionalidad*, es decir motivaciones, intenciones y razones detrás de cada comportamiento. Los conceptos sociales serán el núcleo de las actividades diarias de los analistas y diseñadores. El análisis social no se suma al análisis técnico sino que pasa a ser la base para el desarrollo del sistema. Un actor en este mundo es el centro del modelado: es una entidad activa que es capaz de accionar de modo independiente; los actores pueden ser humanos, hardware, software o una combinación de los mismos. Los actores son autónomos y sus acciones no pueden ser completamente controladas y no son conocidas en su totalidad, aunque tampoco son completamente aleatorias ya que tienen motivaciones e intenciones que nos ayudan a obtener una descripción muy precisa aunque no se especifica su comportamiento exacto [25, cap. 1].

El *i* Framework* transforma el modo de pensar de la Ingeniería de Requerimientos. Yu plantea una nueva estrategia para afrontar los requerimientos; con la premisa central de que para que el diseño del sistema que queremos modelar sea real se tienen que analizar las relaciones entre *actores sociales*, el foco ya no está más en elicitar requerimientos funcionales (el "qué" del sistema) y no funcionales (el "cómo" del sistema) de los distintos *stakeholders*, sino en una nueva arista que trata de entender las motivaciones que mueven a cada actor a solicitar un requisito: el "por qué". Los autores llaman a este enfoque *Ingeniería Temprana de Requerimientos*: se basa en la hipótesis de que los actores no son una entidad estática sino móvil y compleja; los actores muchas veces no entienden exactamente qué quieren, tienen intereses ocultos, preconcepciones, interdependencias entre ellos,

[9] Usamos lógica en el sentido amplio de la palabra, que incluye lógicas temporales, lógicas de primer orden y lógica de predicados.

[10] El director de tesis de Eric Yu fue John Mylopoulos, coautor del *NFR Framework*.

y demás características, cada una de las cuales el analista debe ser capaz de capturar y reflejar en el diseño. El foco está así en el *entendimiento social* del problema.

Los autores presentan al *i* Framework* como un *goal-oriented, agent-oriented approach*. Yu se basará en una nueva definición de *objetivo* (*goal*) presentada por él mismo: es una condición o un estado de las cosas en el mundo que a un actor le gustaría lograr [27, pag. 3]. En cambio un *agente* para el i* Framework es un tipo de actor.

El *i* Framework* ha tenido una evolución importante y hoy en día existen diversas variantes del mismo; presentaremos aquí los conceptos más importantes y comunes a la mayoría de ellas.

Los *actores* son entidades activas, capaces de realizar acciones en autonomía para lograr sus objetivos, usando su conocimiento; dado que habrá objetivos que no puede realizar sin colaboración de otros actores, deberán entablecer relaciones (interdependencias) e interactuar con ellos [26]; los actores tienen un *tipo* que puede ser un *rol* (un comportamiento de un actor social en un contexto particular) o un *agente* (un actor en una manifestación física y concreta, que puede ser o no humano). Por ejemplo si nuestro actor es *Doctor*, un agente podría ser *Pediatrician* y un rol *Head of Department*.

Los actores se relacionan entre sí a través de distintos tipos de *links*: *juega* (*plays*) que indica que un actor está actuando en un cierto rol en particular (siguiendo con el ejemplo anterior, podría ser *Doctor plays as Head of Department*) *parte-de* (*is-part-of*) indica que un actor de cualquier tipo se descompone en varios actores de ese mismo tipo, y *es-un* (*is-a*) que indica que un actor de cualquier tipo se especializa como otro actor del mismo tipo; estos últimos dos casos muestran una relación de generalización/especialización, como en una jerarquía de clases, por ejemplo *Pediatritian is-a Doctor*.

Yu propone dos nuevos modelos, diferentes y complementarios, que usará para representar a los actores, sus elementos, y las dependencias entre actores. Esta propuesta fue hecha en [24]:

- *Strategic dependency (SD) model*: es un modelo a alto nivel que describe un proceso que permite ver a los agentes en términos de las dependencias directas entre ellos; este modelo ayuda a analizar las **relaciones externas entre agentes**;
- *Strategic rationale (SR) model*: describe los problemas y preocupaciones (*concerns*) que los agentes tienen sobre los procesos existentes y alternativas propuestas, y cómo lograrlas; a diferencia del modelo SD este es un modelo que mira hacia adentro de cada actor para entender qué es lo que está pasando internamente, cómo los procesos se componen de elementos intencionales y cómo estos elementos contribuyen al propósito general. Es un modelo que se focaliza en las **relaciones internas de los agentes**.

Presentamos a continuación un ejemplo de cada modelo, explicando sus elementos principales; ambos ejemplos fueron extraídos de la tesis doctoral de Yu y se basan en la programación de reuniones en modo manual.

La figura 2.5 muestra el modelo SD. Allí observamos a diferentes actores (en este caso **MeetingInitiator**, **MeetingParticipant** e **ImportantParticipant**) y diferentes tipos de elementos que los conectan (como **Assured(AttendsMeeting(ip,m))** o **Proposed-Date(m)**); estos últimos serán los *elementos intencionales* (*intentional elements*).

Los posibles elementos intencionales de un actor son *objetivo* (*goal*), en el sentido de la definición de Yu; *objetivo blando* (*softgoal*), con la definición presentada en el *NFR Framework*, *tarea* (*task*) que es un procedimiento ya definido o un *recurso* (*resource*); en la figura

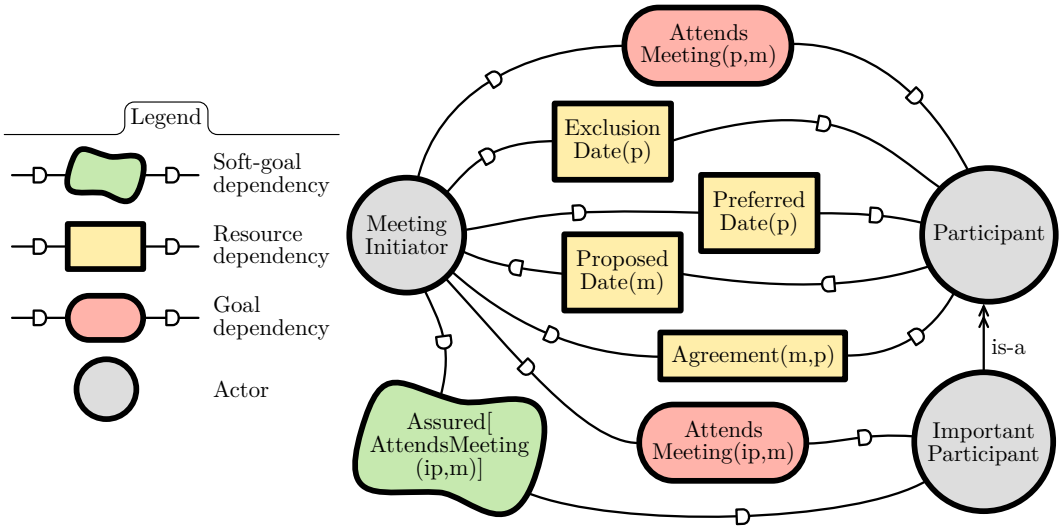

Fig. 2.5: Modelo SD de la organización de una reunión utilizando sin *scheduler*, figura de tomada de [24, fig. 4.1]

podemos observar varios de ellos, como por ejemplo el objetivo **AttendsMeeting(p,m)**, el objetivo blando **Assured(AttendsMeeting(ip,m))** o el recurso **ProposedDate(m)**.

Los distintos actores se relacionan entre sí a través de *dependencias* (*dependencies*): el concepto de dependencia es que un actor (*depender*) depende de otro (*dependee*) para lograr un objetivo (*dependum*) que por sí mismo no podría lograr. Hay una dependencia por cada tipo de *dependum* o elemento intencional: *dependencia de objetivo* (*goal dependency*), *dependencia de objetivo blando* (*softgoal dependency*), *dependencia de tarea* (*task dependency*) y *dependencia de recurso* (*resource dependency*). Ejemplos de las mismas son la dependencia de objetivo con *depender* **MeetingInitiator**, *dependum* **AttendsMeeting(p,m)** y *dependee* **MeetingParticipant**, donde **MeetingInitiator** depende de la respuesta del **MeetingParticipant p** para saber si participará en la reunión propuesta **m**; o la dependencia de recurso con *depender* **MeetingParticipant**, *dependum* **ProposedDate(m)** y *dependee* **MeetingInitiator**, en la cual **MeetingParticipant** propone posibles fechas para la reunión **m** al **MeetingInitiator**.

En cambio en la figura 2.6 puede verse un ejemplo del modelo SR; aquí vemos representados en detalle y por actor todos los elementos que están bajo su responsabilidad. Cada actor tiene un *área de control* (*boundary*) que marca su límite de acción; lo que escapa del aŕea de control se debe lograr con la colaboración de otros actores a través de dependencias, en cambio lo que está dentro de esta región puede realizarlo en autonomía valiéndose solamente de elementos intencionales. En este modelo puede verse cómo se relacionan los elementos intencionales a través de *links de elementos intencionales* (*intentional element links*), que pueden ser *descomposición* (*decomposition*), que descompone una tarea en uno o más elementos intencionales; *contribución* (*contribuition*) que indican qué tipo de contribución tiene un elemento hacia un *objetivo blando* (en el mismo sentido usado en el *NFR Framework*); *medio-fin* (*means-end*), que indica una relación entre un *fin* (como un objetivo a lograr, objetivo blando a satisfacer, tarea a cumplir o recurso a producir) y un *medio* a través del cual lograrlo. Por lo general el medio es una tarea, ya que justamente una tarea expresa cómo lograr algo. Las *reglas* (*rules*) son una aplicación concreta de una relación medio-fin. En la misma figura 2.6 podemos encontrar ejemplos de estos *links*: se

puede observar una descomposición entre la tarea **OrganizeMeeting** que se descompone en el objetivo duro **MeetingBeScheduled** y los objetivos blandos **Quick** y **Effort**; una contribución positiva puede verse entre los objetivos blandos **Quality[ProposedTime]** y **Convenient[m,d]** (donde **d** es una fecha y **m** una reunión) que indica que cuanto más precisas sean las propuestas de fechas recibidas, más simple será coordinar una fecha para la reunión; un *link* medio-fin en cambio puede verse entre la tarea **IntersectAvailableDates** y el objetivo **FindAgreeableSlot**, indicando que el contar con las fechas disponibles de los participantes ya organizadas es un medio para encontrar una horario y fecha disponible para los participantes, aunque otro posible medio para este mismo objetivo podría ser la tarea **SendSlot** que indica que el actor **MeetingInitiator** envía una propuesta de horario sin tener en cuenta la disponibilidad de los participantes. Por último, podemos observar dos reglas distintas para la relación medio-fin **FindAgreeableSlot**: **SendSlot** y **IntersectAvailableDates**.

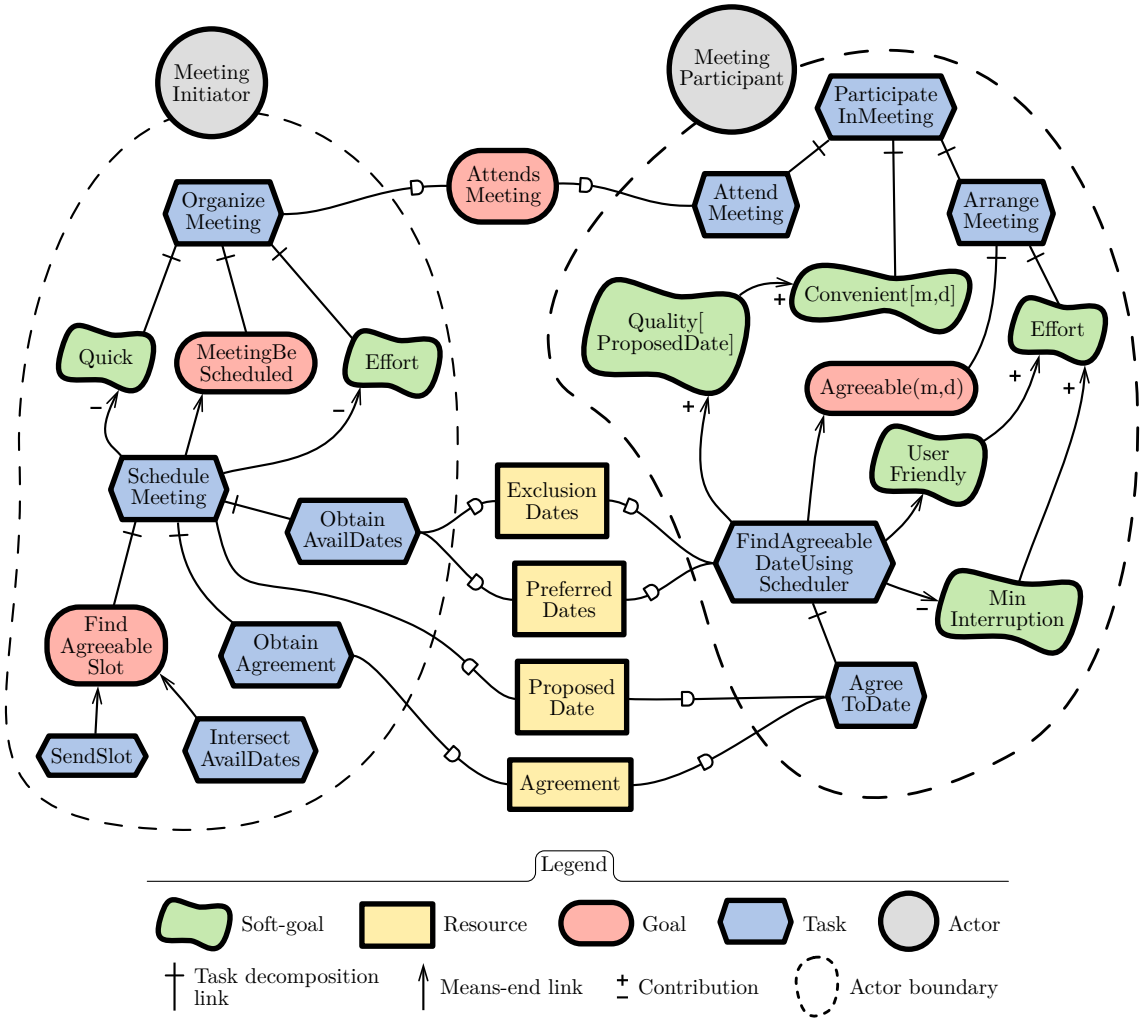

Fig. 2.6: Modelo SR de la organización de una reunión sin *scheduler*, versión modificada de [24, fig. 4.4]

En cuanto a los elementos del framework que encontramos representados en cada mo-

delo, vemos que en el modelo SD aparecen solamente los agentes, elementos intencionales y dependencias entre agentes, mientras que en el modelo SR figuran los agentes con su área de control y todos los elementos presentes dentro de ella, es decir elementos intencionales, links de elementos intencionales y contribuciones; si el agente tiene alguna dependencia con otros, esto se visualizará también en el modelo SR a través de dependencias entre agentes, pero estarán fuera de su área de control.

Un ejemplo de los modelos SD y SR pueden verse en las figuras 2.5 y 2.6.

Ambos modelos pueden ser usados junto con la estrategia *Business Process Reengineering*[11] (BPR) para proponer caminos alternativos. Si se toma como punto de partida el modelo SD vemos que allí están presentes los objetivos: teniendo en cuenta que estos no son explícitos, entonces puede existir más de un modo de cumplirlos; para cada objetivo además existirá una regla (es decir una relación concreta medio-fin) que especifica cómo lograrlo, y nada impide que pueda existir más de una para el mismo medio y fin. La estrategia BPR en el marco del *i* Framework* está fuera del alcance de este trabajo, ya que consideramos que el mayor aporte del framework son los modelos *Strategic dependency* y *Strategic rationale* ya presentados.

Limitaciones y ventajas

El *i* framework* logró convertirse en un estándar en el 2008, llamado *User Requirements Notation* (URN)[12]. Además de ello se realizan workshops internacionales y cuenta con su propia página wiki[13].

El framework, influenciado por el *NFR Framework* y sus autores, cambia el modo de pensar la Ingeniería de Requerimientos hasta el momento. Se enfoca en ésta como un todo, abarcando en su análisis desde el principio tanto requerimientos funcionales como no funcionales y presentando modelos comunes para ambos. No se trata por ende de una mera integración entre el mundo funcional y el no funcional sino en un modo de considerarlos juntos desde la Ingeniería Temprana de Requerimientos.

Yu et al. presentan dos modelos complementarios para visualizar los elementos de su framework, *Strategic Dependency* (SD) y *Strategic Rationale* (SR), además de una semántica para modelar los conceptos presentados. Al ofrecer una visión distinta del proceso de modelado de software, difiere también en sus objetivos y características. Acepta el cambio en el diseño de software y permite rediseñarlo a través de su integración con BPR así como también estudiar distintas alternativas, llevando una versión más moderna al diseño del software para la época. Podemos deducir que considera al software como algo móvil, dinámico y en constante evolución[14], y los expresa a través de sus modelos con la integración entre SD, SR y la estrategia BPR.

La resolución de conflictos entre objetivos es un tema no mencionado en la tesis doctoral de Yu; esto surge recién en el 2014 donde, como dicen Franch et al. en [26], Yu propone un algoritmo donde define el nivel de satisfacción de sus elementos intencionales [28] entre cuyos resultados se encuentra el *conflicto*.

Por otro lado, si bien la propuesta del framework resultó muy interesante para el

[11] *Business Process Reengineering* es un enfoque integral que se centra en examinar el flujo de trabajo y los procesos dentro de una organización, con el fin de mejorar la eficiencia, la eficacia y la adaptabilidad. https://en.wikipedia.org/wiki/Business_process_re-engineering

[12] https://www.cs.toronto.edu/km/istar/

[13] http://istarwiki.org

[14] "The framework provides features to help describe processes ("modelling") and to guide change ("re-engineering")." [24, Sección 1.5]

mundo académico, presentó algunas limitaciones. En primer lugar el mismo Yu en [24] menciona que la escalabilidad y factibilidad del framework deberían ser estudiadas más profundadamente; esto se reafirma luego con el trabajo de Estrada Esquivel [29, pag. 64], quien realizó un análisis empírico del framework, con modelos de distintos tamaños durante un período total de 9 meses, y confirma el problema de la falta de mecanismos para hacer del *i* framework* una herramienta escalable en problemas de gran tamaño y complejidad dado que la cantidad de elementos que habría que representar en un solo modelo haría el análisis inviable. Estrada Esquivel también estudio otras características del framework como la modularidad y reusabilidad, tampoco soportadas; la modularidad presenta el mismo problema que la escalabilidad (el framework tiene una estrategia de representación monolítica con todos los elementos relativos a un actor, sin posibilidad de armar una jerarquía); la reusabilidad, dado que no existe modularidad, hace difícil la reutilización de las partes.

Otro problema de la representación del *i* Framework* es la dificultad para reconocer *concerns transversales* [30]: dada la falta de escalabilidad, en problemas de gran tamaño puede ser difícil identificarlos incluso dentro del mismo modelo, y como veremos en 2.2.4 en general estos tipos de *concerns* suelen ser NFR.

Con el paso de los años surgieron trabajos inspirados en el *i* Framework* que intentaron resolver los problemas anteriormente mencionados, sobre todo el de la modularidad que es considerado el más serio, como por ejemplo [30] y [31].

2.2.3. NFR integrados a diagramas funcionales (2004)

Uno de los problemas más conocidos de los NFR es cómo integrarlos a los requerimientos funcionales. Para ello Cysneiros et al. desarrollaron un proceso que ayuda a elicitarlos, analizar conflictos entre ellos, así como también guía al programador en su integración a los modelos conceptuales tradicionales, que tendrán de esta manera información sobre los NFR y viceversa [8].

El proceso propuesto por los autores es *goal-oriented*, *process-oriented*; se asumen como herramientas ya conocidas el *Language Extended Lexicon* (LEL)[15] como vocabulario común para ambos mundos funcional y no funcional, y el *NFR Framework* para crear los SIGs, de la manera propuesta por Chung et al. que vimos en 2.2.1. Es, además, un proceso general, en el sentido de que no se focalizará en ningún NFR en particular.

Esta estrategia propone un método para integrar los NFR elicitados en casos de uso, escenarios, diagramas de clases, diagramas de secuencia y diagramas de colaboración; puede ser utilizado en etapas tempranas o tardías del proyecto, o incluso en ambas. También se puede usar para validar proyectos ya empezados, en el sentido de que si el modelo conceptual ya fue desarrollado aún es posible construir la estrategia solo para los NFR e integrarlos al resto [8, pag. 1]; se puede por lo tanto concluir que funciona sea como *process-oriented approach* que como *product-oriented approach*, dependiendo de las necesidades del usuario.

La propuesta puede verse en 2.7. Como precondiciones los autores asumen que:

[15] Modelo semi-formal que contiene el vocabulario de un Universo del Discurso (UofD).
UofD se refiere al conjunto de entidades, conceptos y relaciones que son relevantes para el dominio del problema y define el alcance y los límites del modelo.
https://en.wikipedia.org/wiki/Domain_of_discourse

- se utilizará alguna estrategia ya conocida para elicitar el LEL;
- la notación de la estrategias funcional y no funcional utilizarán el mismo vocabulario que el LEL;
- el usuario conoce estrategias de elicitación de requerimientos funcionales y no funcionales.

Como primer paso se arma el LEL (paso 1), con el conocimiento del UofD y una estrategia de elicitación. Teniendo al LEL ya definido se pueden construir en paralelo los pasos 2 y 3, que toman como entrada común el lenguaje LEL resultado del primer paso; la salida de la construcción funcional es un modelo de objetos (para su construcción los autores no dan detalles, ya que está fuera del alcance de esta herramienta) que debe cumplir la condición de que el vocabulario utilizado sea el mismo que el utilizado para el LEL; para la construcción no funcional en cambio, se utiliza la estrategia del *NFR Framework* y el resultado son los SIGs junto con el LEL modificado para agregarle los símbolos no funcionales (NFR-LEL). El resultado de la construcción no funcional puede haber generado conflictos entre los NFR que hay que resolver antes de poder integrar ambas estrategias (paso 4). Estos conflictos pueden ser resueltos de 3 modos: comparar SIGs que tienen el mismo *tipo*[16], utilizar una base de conocimiento para identificar SIGs con NFR usualmente conflictivos, o comparación de a pares para los SIGs que no cumplen las dos condiciones anteriores. En el paso 5, último paso, se integran ambas construcciones; en este paso se espera que pueda haber un ida y vuelta hacia algunos de los pasos previos antes de llegar al resultado final, ya sea porque por ejemplo faltan símbolos o alias en el lenguaje LEL (lo que llevaría a agregarlo y comenzar nuevamente desde el paso 1) o porque se encuentran *discrepancias* que tienen que ser resueltas (lo que nos lleva a rehacer los pasos 2 y 3). La integración puede hacerse como se dijo antes en etapas tempranas o tardías del proyecto (o ambas), y de ello también dependerá con cuáles tipos de diagramas funcionales se realice; los autores dan estrategias para integrar NFR con casos de uso, escenarios, diagramas de clases, diagramas de secuencia y diagramas de colaboración.

Limitaciones y ventajas

La estrategia de Cysneiros et al. hace foco en uno de los problemas clásicos de la Ingeniería de Requerimientos: la integración entre requerimientos funcionales y no funcionales. No desarrollan nuevas técnicas para elicitar ni diseñar ninguno de los dos mundos sino que se basan por un lado en la vasta variedad de opciones existentes para los requerimientos funcionales y por otro en el exitoso *NFR Framework* para los requerimientos no funcionales; en este sentido se puede decir que el aporte de los autores está en los pasos 4 y 5 del SADT.

Como resultados de los casos de estudio analizados, los autores concluyen que si se aplica esta estrategia se debería obtener un modelo conceptual más completo, con ambos tipos de requerimientos. Además, los tiempos de salida al mercado pueden ser menores dado que se evitan varios errores derivados de la falta de análisis de los NFR [8, pag. 2].

Podemos decir que es una herramienta global en el sentido de que puede ser utilizada con cualquier NFR y eso es un punto a favor de los autores, ya que muchas veces los NFR son analizados uno por uno lo que dificulta el método en sí y aumenta la complejidad

[16] tipo tomado según la definición de Chung et al. en [1].

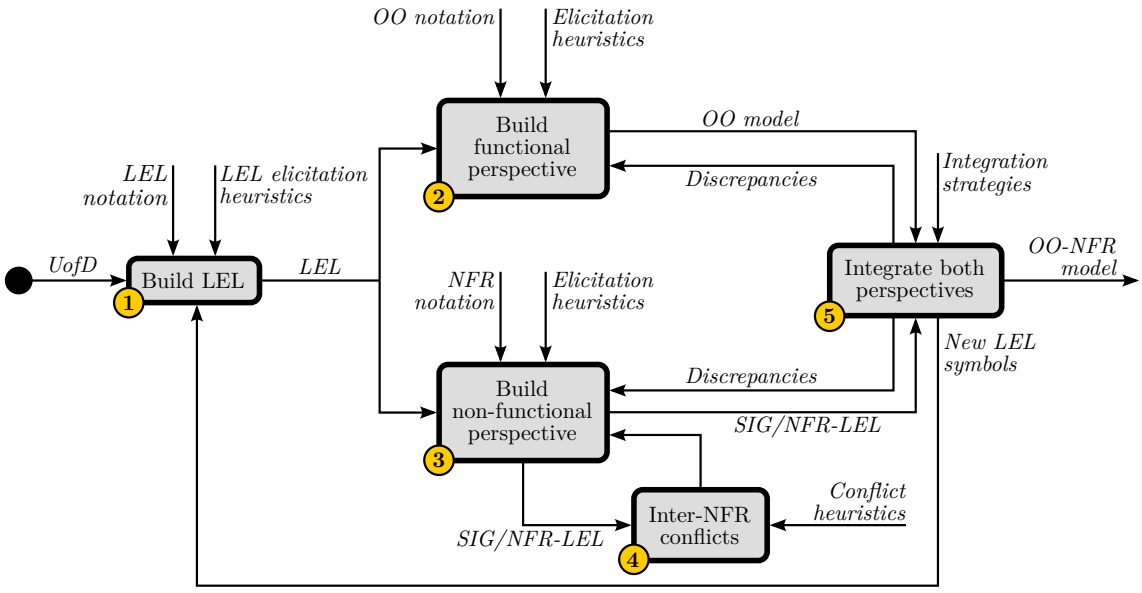

Fig. 2.7: SADT con la propuesta de Cysneiros et al., versión modificada de [8, fig. 1]

para el usuario; de todas maneras se obtendrán mejores resultados si los NFR involucran acciones que afectan al diseño del software [8, pag. 21].

Se utiliza el mismo lenguaje entre el mundo funcional y el no funcional, lo que facilita la integración y la correspondencia entre los requerimientos. Por otro lado, podemos concluir que la integración con los requerimientos funcionales, si bien cubre los diagramas más importantes, no es completa, y el usuario debe atenerse a las opciones dadas por los autores o desarrollar una propia.

Una limitación en cuanto a las estrategias de resolución de conflictos entre NFR, mencionada por los mismos autores, es que la heurística de comparar SIGs de a pares puede ser inaplicable si el número de SIGs es muy grande; cuál es el límite no está especificado.

Los autores mencionan como un punto en contra la falta de automatización entre el lenguaje y la construcción de los SIGs; esta tarea es manual, con lo cual consume mucho tiempo, además de que no se tiene mucha seguridad sobre la exactitud del proceso. La escalabilidad es un tema aún a tratar y analizar [8, pag. 21].

Si bien se describen heurísticas para resolver conflictos entre NFR e integrar FR y NFR, que son definitivamente dos puntos claves en el diseño del software, algo que creemos falta en esta estrategia son heurísticas para la resolución de conflictos entre FR y NFR.

2.2.4. Classpects como requerimientos no funcionales (2006)

Basándose en el *NFR Framework* [20] [1], Marew et al. proponen una táctica para sintetizar *classpects* desde un SIG y luego integrarlo con las clases (existentes o no) de nuestro sistema [7].

Los *aspects* son conocidos por capturar *concerns transversales* entre clases; muchas veces estos *concerns* son luego identificados como NFR, con lo cual los *aspects* podrían ser usados para representar NFR. Un *classpect*, en cambio, es una construcción que surge de

la unión entre las capacidades de una clase y un *aspect*: un *classpect* se puede comportar como una clase (según la definición de OOP) pero también puede comportarse como un *classpect* en el sentido de que exhibe el comportamiento de los *aspects* capturando los *concerns transversales*.

Como precondición para poder aplicar la táctica, los autores asumen que se cuenta con un análisis de los requerimientos funcionales junto con un diagrama de clases y casos de uso. La integración planteada para integrar NFR y FR cuenta con los siguientes 4 pasos:

1. creación del SIG;
2. sintetización de *classpects* desde el SIG;
3. descubrir clases desde los *classpects*;
4. integración de clases y *classpects*.

El paso 1, creación del SIG, se asume conocida y dada por [1]; en este punto se tratan solamente NFR.

Para el paso 2, tomando en cuenta que en el SIG cada nodo representa a un NFR con un tipo y un tópico (el argumento sobre el que opera el NFR), se analizan 4 reglas posibles para crear los *classpects* derivados del SIG. Luego de aplicar estas reglas nos encontraremos con una primera versión de código generado con varios *classpects*, donde cada classpect representa uno de los NFR del SIG, con las siguientes dos características:

- es esquelético: este paso se limita a insertar mensajes basados en las operacionalizaciones. Los argumentos finales y el cuerpo de los métodos aún no están definidos, así como tampoco la implementación;
- se preserva la estructura jerárquica del SIG: cada regla es fiel a la jerarquía del SIG, por lo tanto la estructura resultado de *classpects* lo refleja. Esto permite una mejor mantenibilidad, ya que es muy fácil trazar una relación entre SIG y *classpects*.

Para el paso 3, se cuenta ya con los *classpects* sintetizados para los NFR identificados en el SIG; ahora se necesita ver cómo se conectan con las clases ya existentes de los FR[17]. Para este momento se pueden encontrar distintas situaciones: que el proceso de elicitación de FR y NFR haya sido hecho con el mismo vocabulario (no es lo más usual); partiendo de los casos de uso, analizar con qué clase(s) colabora(n) para implementarlo y luego tratar de identificar esas mismas clases en el diagrama de secuencias; usar el documento de requerimientos y tratar de encontrar NFR junto a las entidades que representan. Cabe notar que en este paso solamente se identifican puntos posibles de conexión, sin profundizar en ello.

En el último paso, se vuelve a los *classpects* esqueléticos generados por el paso 2 y se trata de completar las partes faltantes. Por ejemplo, las operacionalizaciones fueron generadas como mensajes de texto, en este paso se busca encontrar a qué clase y métodos (si existen) corresponden. Para esto se va a necesitar el diagrama de clases. Dado que los *classpects* logran capturar detalles de implementación que no están presentes en la documentación funcional, es posible incorporarla en este punto.

Limitaciones y ventajas

[17] En este paso intervienen por primera vez los requerimientos funcionales.

La integración entre FR y NFR a través de *classpects* propuesta por [7] presenta una táctica con la intención de unir dos mundos divididos, el de los FR y los NFR. Toma como punto de partida el SIG del *NFR Framework*, que sentó las bases para muchos casos de estudio y técnicas posteriores, y el concepto de *aspects* presentado en [23] e intenta dar una táctica para lograr esta unión.

De la presentación hecha no se puede inferir cómo logran este objetivo: en particular en el punto 4 donde tratan la integración de clases y *classpects* falta estructura sobre cómo logran ciertos puntos, como por ejemplo: cómo se agrega una clase si no existe, cómo se agregan operaciones o atributos, posible reestructuración del código luego de finalizar los pasos. Tampoco se tratan en detalle los conflictos entre NFR, o entre NFR y FR y cómo resolverlos.

El hecho de que la integración entre el mundo funcional y el no funcional, según esta propuesta, puede suceder solamente una vez que se tienen los diagramas funcionales ya desarrollados (paso 3 y 4), hace pensar que los autores están esperando a un estadío tardío del proyecto para la integración con los atributos de calidad; en particular, para completar los *classpects* (paso 4) se necesita el diagrama de clases ya desarrollado, de lo cual se concluye que se están tratando los NFR una vez que se cuenta ya con los FR del proyecto.

Por otro lado, creemos que es interesante el uso de los aspects para representar NFR, ya que por su naturaleza afectan transversalmente a los distintos módulos de un programa y esto es un punto de encuentro con los atributos de calidad. Sin embargo falta más análisis y casos de estudio para poder concluir si es una técnica verdaderamente efectiva, y en cuáles circunstancias.

Cabe notar además que los autores no presentan ningún caso concreto que respalde la táctica, como así tampoco mencionan si es una técnica general para cualquier NFR o no.

2.3. Comparación de las metodologías pre-2013

En la sección 2.2 presentamos distintas metodologías para incluir los NFR en el desarrollo del software previas al año 2013.

De todas ellas, hay dos que constituyen una innovación completa: el *NFR Framework* y el *i* Framework*, mientras las dos restantes toman como punto de partida para la elicitación y el análisis de los requerimientos no funcionales el trabajo hecho por Chung et al. y se enfocan en otros aspectos de la Ingeniería de Requerimientos, como ser la integración de los NFR con los FR (como Marew et al. y Cysneiros et al.) o en heurísticas para tratar conflictos entre NFR (como Cysneiros et al.). Incluso en el *i* Framework*, si bien es una metodología novedosa, podemos ver la influencia del *NFR Framework* que fue cronológicamente anterior, en conceptos como objetivos blandos, links de contribución o links de descomposición.

Como diferencia principal entre el *i* Framework* y las demás metodologías presentadas podemos decir que el trabajo de Yu et al. es un framework para la Ingeniería de Requerimientos, mientras los demás trabajos están focalizados principalmente en los NFR.

Todas las técnicas presentadas utilizan el *process-oriented approach* (que involucra a los NFR desde el comienzo del análisis) lo cual presume mejores resultados, salvo el *i* Framework* donde esta distinción no tiene sentido ya que es un framework donde se tratan todo tipo de requerimientos focalizándose en una elicitación prematura a través de la Ingeniería Temprana de Requerimientos. Respecto a la propuesta de Cysneiros et al,

como mencionamos en 2.2.3, es una técnica que podría ser también utilizada de acuerdo al *product-oriented approach*, dependiendo del momento en el cual se encuentre el proyecto de software cuando se aplica.

Analizando solamente los *NFR Framework* y *i* Framework*, podemos ver que ambos utilizan el *goal-oriented approach*; si bien el *NFR Framework* usa un *process-oriented approach* que el *i* Framework* no, los dos coinciden en la importancia de la elicitación, análisis y documentación temprana de los requerimientos que el *i* Framework* desarrolla en su Ingeniería Temprana de Requerimientos. Ambos trabajos proponen sus propios modelos para describir sus objetivos y definen una semántica donde se desarrollan los conceptos introducidos. Una de las diferencias más importantes entre estos trabajos es que Chung et al. se focalizan en los NFR mientras que Yu et al. analizan todo tipo de requerimientos conjuntamente (aunque como vimos en la sección 2.2.1 los autores afirman que el *NFR Framework* puede utilizarse para todo tipo de requerimientos en realidad no se detalla cómo ni se dan casos de estudio que respalden esta afirmación). Chung et al. necesitan dar una semántica para cada tipo de NFR mientras que esto carece de sentido en el trabajo de Yu et al. El hecho de que en el *NFR Framework* se proponga un SIG y una semántica por cada NFR lo hace modular, en cambio el *i* Framework* tendrá siempre como resultado un solo modelo SD y uno SR sin importar la complejidad y tamaño del problema tratado, lo cual en problemas grandes dificultará el análisis del resultado. Una parte importante del *NFR Framework* es la resolución de conflictos mientras que el *i* Framework* propone para esto el *forward evaluation algorithm* definido en el año 2014 por Yu et al. [26, pag. 16] que se utiliza para analizar el nivel de satisfacción de los *elementos intencionales*, con lo cual si bien en la tesis doctoral de Yu este tema no es mencionado eventualmente se incluyó y se transformó en una librería del *i* Framework*[18]. Por último, en la bibliografía leída del *NFR Framework* no se considera la posibilidad de cambios (los autores mencionan que debería ser una extensión), mientras que el *i* Framework* presenta en la estrategia BPR que permite pensar en nuevas soluciones y rediseñar el proceso en caso de ser necesario.

En cambio entre los trabajos de Cysneiros et al. y Marew et al. una diferencia es que el primero pide utilizar un lenguaje común para todos los requerimientos mientras que el último deja que cada perspectiva use el lenguaje que prefiera y luego busca integrarlos; esto debe ser bien analizado por el usuario antes de decidirse por una de las dos técnicas, ya que si bien elegir lenguajes distintos para FR y NFR da más flexibilidad seguramente genera más trabajo en etapas sucesivas del proyecto. Otra diferencia es que Cysneiros et al. cubren un espectro más amplio del mundo funcional: permiten integrar los NFR en etapas tempranas, tardías o ambas y dan heurísticas para integrarlos en 5 modelos conceptuales entre los cuales están contemplados los más importantes; en cambio Marew et al. proponen que la integración propiamente dicha entre los NFR (sintetizados como *classpects*) y los FR, se realice tomando como documento principal el diagrama de clases, y creemos que el hecho de tener que contar con un diagrama funcional de estadío tan avanzado de proyecto para completar los templates de los *classpects* puede interferir con la calidad de la integración resultante.

Comparando las cuatro metodologías, podemos observar que Marew et al. y Yu et al. no dan heurísticas propias para tratar los conflictos entre NFR o en el caso de Yu et al. para conflictos entre requerimientos en general, en cambio las metodologías restantes proponen sus propios métodos para este paso. De todas maneras, dado que Marew et al. utilizan el *NFR Framework* para crear los SIGs puede ser que en ese paso traten los

[18] https://austria.omilab.org/psm/content/istar

conflictos, aunque no lo explicitan.

Por último, ninguna de las metolodologías aquí analizadas, da una opción para tratar la resolución de conflictos entre NFR y FR o la aparición de nuevos requisitos funcionales (y su posterior integración con los no funcionales), puntos importantes de la Ingeniería del Software; de este último punto queda excento el *i* Framework* ya que da una propuesta para rediseñar el software continuamente mediante el BPR.

Si pensamos en las distintas fases de los requerimientos no funcionales y su integración en un proyecto de software tratadas por las metodologías de la sección 2.2 podemos decir que cada una se enfoca en detalle en las presentadas en la tabla 2.1:

	NFR Framework	i* Framework	Classpects	Modelos conceptuales
Análisis NFR	✓	✓	✓	✓
Lenguaje NFR-FR		✓		✓
Integración NFR-FR		✓	✓	✓
Conflictos NFR-NFR	✓		✓	✓
Conflictos NFR-FR				
Nuevos FR		✓		

Tab. 2.1: Comparación entre metodologías de acuerdo su enfoque de NFR e integración con FR

3. ESTADO DEL ARTE DESDE EL 2013 HASTA LA ACTUALIDAD

En este capítulo analizaremos como primer paso, qué sucedió con las metodologías presentadas en 2.2. A continuación presentaremos algunas metodologías de desarrollo surgidas posteriormente al 2012, con los criterios explicados en 1.2.

3.1. Análisis cuantitativo y cualitativo pre-2013

Para comprender la evolución del uso de los entornos y vigencia de los artículos y ejemplos usados haremos un análisis cuantitativo de los trabajos de referencia [5] y [4] junto los ejemplos y las metodologías analizados en el capítulo 2: *NFR Framework*, *i* Framework*, *NFR integrados a diagramas funcionales* y *Classpects como requerimientos no funcionales*.

Para la búsqueda utilizamos el sitio Google Scholar, filtrando por nombre completo de artículo y tomando la cantidad de citas en 3 momentos temporales diferentes para evaluar la evolución a lo largo del tiempo: a partir del 2013, 2020 y 2023 (todos hasta junio del 2024).

Trabajo	#Citas desde 2013	#Citas desde 2020	#Citas desde 2023
Bajpai et al.	54	26	8
Mairiza et al.	186	75	22
NFR Framework (libro)	1510	473	130
NFR Framework (paper)	538	119	26
i* Framework (libro)	877	185	45
i* Framework (paper)	69	50	11
Modelos conceptuales	150	36	6
Classpects	4	1	
Therac-25	887	261	78
ION	9	3	1
LAS	97	31	8
Pentium FDVI	8		

Tab. 3.1: Análisis cuantitativo de los trabajos analizados en la sección 2

De los trabajos de referencia, el de Bajpai et al. no ha tenido mucha repercusión, lo cual puede inferirse las 54 citas con las que cuenta desde el 2013 hasta junio del 2024 y las sólo 8 que tiene desde el 2023, mientras que los autores Mairiza et al. fueron más citados en estos años, con 186 citas totales desde el 2013, y siguen siéndolo, con 22 citas en el último año y medio.

Entre las metodologías, podemos ver que de las 4 analizadas hay 2 que se destacan por cantidad de citas, el *i* Framework* y el *NFR Framework*: en ambos el libro ha sido más popular que el paper, si bien los dos se mantienen con una cantidad de referencias

alta a lo largo del tiempo. Por otro lado, podemos ver que las metodologías de los *Modelos conceptuales* tenía 36 referencias 10 años atrás (que si por ejemplo lo comparamos con las 50 del *i* Framework* (paper) no son despreciables) pero se ha ido perdiendo con los años: ya desde el 2020 se presentan 36 citas y desde el 2023, sólo 6. La metodología de los *Classpects* nunca tuvo demasiada repercusión, con sólo 4 citas en los últimos 10 años.

Se realizó asimismo un análisis cualitativo de las citas más recientes de estos trabajos (es decir desde el 2023), para evaluar además qué tipo de aporte están realizando; esta investigación se hizo tomando en cuenta sólo los trabajos presentes en los sitios Google Scholar, Research Gate, IEEE y ACM, y en casos donde la cantidad de citas es muy grande (como el *NFR Framework*, con 156 en total) no se revisaron todos.

De los trabajos de referencia, el de Bajpai et al. en la actualidad no se utiliza y cuando se hace, se toma solo como trabajo de referencia; tal es el caso de [32], [33] y [34]. En cambio el trabajo de Mairizia et al. ha sido levemente más popular y con aportes más significativos: su detallada categorización de dominios de aplicación y tipos de sistemas se utiliza en el framework propuesto por [32] así como en la ontología de [35]; también es citado como trabajo de referencia en [33], [36] y [37].

En cuanto a las metodologías, para el *NFR Framework* se observa que las referencias que se hacen son en general positivas: trabajos como [38] lo consideran un buen método para tratar a los NFR, mientras que otros como [39] y [40] lo usan para armar su metodología, en [41] usan el concepto de satisfacible así como el de SIG, [42] lo usa para construir el catálogo de los NFR mientras que [43] lo utiliza como un paso del método propuesto; interesante de estos dos últimos trabajos es que utilizan tanto el *NFR Framework* como el *i* Framework*, viendo así que estas metodologías tan exitosas pueden además complementarse. Por otro lado, [44] pone en duda que ambos frameworks, *NFR* y *i**, hayan sido empíricamente validados, afirmación que inferimos puede ser verdadera: en el caso del *i* Framework* los problemas que presenta, como falta de escalabilidad, modularidad y reusabilidad (y que ya mencionamos en 2.2.2), lo hacen poco atractivo para la industria; en cambio hablando del *NFR Framework* creemos que los conocimientos previos que demanda, el hecho de que no esté desarrollado para todos los NFR y la dificultad para aceptar cambios lo hacen poco interesante para los proyectos de hoy en día.

La metodología de los *Modelos conceptuales* ha tenido un aporte más discreto: el trabajo es citado por [45] y [46], mientras que [47] advierte sobre la dificultad de incluir al usuario final en el proceso dada la complejidad de la notación utilizada.

Por último, de nuestro análisis de las referencias del *i* Framework* se puede inferir que se cita en modo positivo además de seguir en vigencia para la construcción de nuevas técnicas: ejemplos de esto son [48] donde el metamodelo que construyen se basa en el *i* Framework*, [49] donde los autores pretenden crear un chatbot que ayude en el desarrollo de nuevas extensiones del *i* Framework*, en [50] lo usan para crear un framework de simulación de comportamiento social, [51] lo utiliza en el primer paso de la construcción de su modelo, y finalmente [52] se basa en el *i* Framework* para modelar sus ecosistemas. Es interesante también observar que además de seguir en vigencia se está utilizando con algunas de las últimas tendencias en el área: IoT [48], LLMs [49], Machine Learning [50]. Se observan por otro lado algunas referencias no tan positivas, como el caso de [53] que explicita que el framework no puede capturar orden temporal, o [54] que advierte sobre la dificultad del *i* Framework* así como también del *NFR Framework* de detectar omisiones o ambigüedades en los requerimientos que surgen del contexto en el proceso de diseño.

3.2. Tipos de metodologías de gestión de NFR

En esta sección analizaremos técnicas surgidas con fecha posterior al trabajo de Bajpai et al. [5], es decir luego del 2012 y hasta el 2024; los criterios para la elección de las propuestas fueron detallados en 1.2.

3.2.1. Scrum para Big Data y Cloud (2017)

Este trabajo tiene como último autor a Lawrence Chung, uno de los creadores del *NFR Framework*. La propuesta se centraliza en proyectos que estén desarrollados con Big Data y Cloud usando Scrum, y busca incluir los NFR a éste último [55].

En la figura acá figura del ciclo de Scrum puede verse el ciclo tradicional de Scrum. En Scrum los requerimientos del sistema se representan con *user stories*, requerimientos que en general son funcionales ignorando los requerimientos no funcionales completamente o hasta un punto demasiado avanzado del desarrollo del proyecto (como sucedía en las metodologías de la sección 2.2, Sachdeva y Chung piensan que Scrum no es la excepción).

Conocemos ya la importancia de los NFR y de incluirlos lo antes posible en cualquier proyecto de software; los autores consideran que esto tiene particular relevancia en los proyectos que incluyen Big Data y Cloud, particularmente los requerimientos de *security* y *performance*, porque *security* es crítico para cualquier proyecto que utiliza Cloud donde suele haber problemas de *leaks*, y la *performance* es un requerimiento que debe estudiarse conjuntamente con *security* ya que uno suele ir en detrimento del otro. Dado además que los cambios a Scrum que proponen los autores deben hacerse de a un requerimiento no funcional por vez, proponen en este trabajo dos modificaciones, una para incluir *security* y la otra para *performance*; en ambos casos es importante que estos cambios se introduzcan en la fase inicial del desarrollo de software.

Para *security* el procedimiento es el siguiente:

1. se crea un *feature* (es decir un conjunto de *user stories* con características en común) llamado *security*;
2. se divide el feature *security* en *user stories* junto al *Product Owner* y los *stakeholders*; en este paso es importante que los escenarios del *acceptance criteria* incluyan un mapeo a *security*;
3. las *user stories* del punto 2 se incluyen en el *backlog* y se priorizan, de manera que sean incluídas en un *sprint* lo antes posible;
4. una vez que estamos en un *sprint* cuyo *backlog* incluye una de las *user stories* en 2, se sigue el ciclo normal de Scrum y cuando todas las *user stories* cumplen su *acceptance criteria* se genera un nuevo *product increment*;
5. a esta altura, en vez de iniciar inmediatamente con el nuevo *sprint*, se analizan los impactos de las medidas de seguridad en las restantes *user stories* (consideramos que un buen momento para hacerlo sería en la *retrospective* aunque los autores no lo explicitan). En caso que haya un impacto de *security*:

- si el cambio es menor, se arregla el problema;
- sino, se genera una o más *user stories* que se agregan al *backlog* y deberán ser priorizadas.

En cambio para *performance* los autores proponen:

1. se inicia con una *spike* (una *user story* de investigación) con el objetivo de cuantificar la *performance* y definir un umbral para la misma;
2. como resultado de lo anterior:
   - si es necesario se crean nuevas *user stories*;
   - se agrega el resultado del umbral al *acceptance criteria* de cada *user story* del feature inicial (punto 1);
   - se agrega el resultado del umbral a la *definition of done*;
3. se comienza un nuevo *sprint* y se define un *sprint backlog*;
4. se realizan tests para medir la *performance* en cada *user story* del *sprint*, donde tenga sentido hacerlo. Si alguno de estos tests falla:
   - si el impacto es grande se debe crear una nueva *user story* para resolverlo;
   - si en cambio no tiene un gran impacto, se resuelve el problema y se continúa.

En este trabajo los autores quieren estructurar el manejo de los NFR en proyectos que incluyen Big Data y IoT desde un estadío temprano del desarrollo del software utilizando una metodología Agile; Agile resulta interesante para proyectos que incluyen IoT dada su flexibilidad y manejo de las necesidades del cliente. Sachdeva y Chung se focalizan en proyectos que utilizan el framework Scrum por ser una de los más populares del mundo Agile, y lo modifican para que incluya al inicio del proyecto la elicitación y análisis de los requerimientos no funcionales *security* y *performance*, por considerarse dos de los más críticos y de mayor costo para el proyecto si no se incluyen en la etapa inicial del desarrollo del software; por ejemplo si estos dos NFR no se toman en cuenta pueden afectar a los requerimientos *maintainability* y *scalability* demasiado tarde en el proyecto [55, pag. 6].

Este método fue aplicado en un caso de estudio en la industria con buenos resultados, pero dado que esto, como admiten los mismos autores, puede deberse a otros factores, sería interesante realizar nuevos estudios con una metodología más precisa para poder sacar conclusiones fehacientes. Los autores no especifican cómo resolver conflictos entre ambos requerimientos no funcionales pero en el caso de estudio el conflicto surgió y debieron hacerlo, aunque no en un estadío temprano lo cual llevó a un rediseño del software; una posible mejora a la técnica, podemos deducir, sería anteponer el manejo de conflictos al del diseño.

La propuesta debería entonces ser aplicada a nuevos casos para poder evaluar su efectividad y escalabilidad, esto último dado que no contamos con información sobre el tamaño del caso de estudio utilizado. Para poder considerarla una propuesta más completa consideramos también que los autores deberían introducir otros de los NFR más populares al framework Scrum, así como también el modo de resolver los conflictos surgidos entre NFR o entre NFR y FR, ambos no especificados. Por último, si bien el trabajo es relevante,

está pensado para proyectos tradicionales de software y no ha sido testado en proyectos complejos de IoT que incluyen otras disciplinas además del software, como indican los autores Prasher et al. [56].

3.2.2. Intencionalidad en la elicitación de objetivos (2017)

Este trabajo [27] tiene entre sus autores a Prado Leite y Cysneiros, ambos partícipes de la propuesta NFR anterior al 2013 analizada en 2.2.3. En este caso, los autores buscarán dar un método de elicitar la intencionalidad; para esto se necesita identificar a los actores sociales en la organización, y una vez hecho esto, elicitar y poder expresar esa intencionalidad, es decir los intereses y motivaciones de los actores. Usarán parte de las estrategias vistas en el *i* Framework* junto con el método *Actor Goals from Lexicon* (AGFL) que mostraremos a continuación. Se usará como referencia el trabajo preliminar de los mismos autores sobre AFLG [57], donde presentan algunos conceptos de la técnica.

La estrategia AGFL sirve para elicitar objetivos duros y blandos[1] e involucra fuertemente la partipación del Ingeniero de Requerimientos. Pueden identificarse 3 pasos principales como puede verse en 3.1.

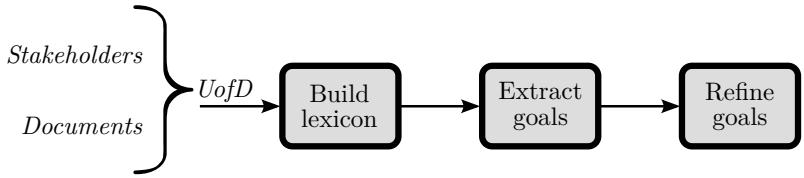

Fig. 3.1: Estrategia *AGFL* presentada en el trabajo de Oliveira et al.

En el primer paso, **Build Lexicon**, se utiliza el lenguaje LEL[2] para ayudar en la construcción de un vocabulario del problema; el LEL capturará distintos símbolos a los que clasificará como *sujeto* (alguien que hace una acción), *objeto* (algo que se ve afectado por una acción), *verbo* (una acción) o *estado* (el resultado de una acción). Oliveira et al. proponen además una estructura para estos símbolos que contiene: un *nombre*, una *descripción*, la *clasificación* antes descripta y una o más *respuestas de comportamiento*; estas están asociadas al símbolo y se entienden como acciones que tendrán lugar cada vez que el símbolo esté presente, y deberían responder a un "por qué". En la figura 3.2 podemos ver un ejemplo de un tipo de símbolo LEL de clasificación sujeto con dos respuestas de comportamiento.

LEL Symbol

Name:	Authorized driver
Notion:	Driver or vehicle that belongs to toll road administration
Clasification:	Subject
Behavioral response:	1- Driver is authorized to travel the road
	2- Driver hopes to have a good trip

Fig. 3.2: Símbolo LEL para el sujeto "authorized driver" de un sistema de peajes.

[1] Objetivos blandos en el mismo sentido del *NFR Framework*.
[2] *Language Extended Lexicon*, presentado en la metodología 2.2.3

Del resultado del primer paso nos interesan particularmente las respuestas de comportamiento, ya que a partir de ellas lograremos extraer los objetivos que son la finalidad del segundo paso, **Extract Goals**. Las respuestas de comportamiento contienen acciones (verbos), que pueden ser divididas en dos tipos: acciones que cambian un estado en otro (*acción concreta*) o acciones que agregan un atributo de calidad a un estado (*acción flexible*) mientras el estado no cambia. En el símbolo LEL de la figura 3.2 se pueden ver dos respuestas de comportamiento, de las cuales la 1 se elicitará como un objetivo duro (verbo "autorizar") mientras la 2 (verbo "esperar") como un objetivo blando.

La estrategia AGFL usa otro tipo de construcción llamada *marco* para ayudar en la elicitación de objetivos desde las respuestas de comportamiento, basándose para ello siempre en la pregunta "por qué". Cada símbolo del LEL tendrá un marco que dependerá de su clasificación; dentro del marco tendremos las respuestas de comportamiento junto a las respuestas de los "por qué" expresados con una sintaxis específica (dependiendo del verbo):

- verbo concreto: porque + sujeto + verbo en pasado + sujeto (*depender*)
- verbo flexible: porque + tipo + tópico + objetivo duro + sujeto[3]

Esto es el último paso antes de la elicitación de objetivos.

Una respuesta de comportamiento genera (al menos) un objetivo; si el objetivo generado depende de un segundo actor se crea un nuevo objetivo de tipo *reflexivo*, sea duro o blando,[4] y la elicitación continúa hasta llegar a un objetivo que no dependa de ningún sujeto; todos los objetivos reflexivos se deben agregar al marco. Los objetivos blandos deben tener una correspondencia con un objetivo duro.

El último paso **Refine Goals**, para el cual es fundamental la intervención del Ingeniero de Requerimientos, consiste en tomar todos los objetivos, duros y blandos, y ordenarlos con el siguiente criterio:

- se agrupan los objetivos de tipo sujeto por actor: todo objetivo que involucre al actor como principal debe estar en la lista;
- si hay objetivos repetidos, se eliminan;
- se da un orden temporal a cada objetivo: los de largo término se colocan al final.

En este trabajo Oliveira et al. quieren presentar una herramienta para ayudar en el proceso de elicitación de objetivos en general tomando en cuenta la intencionalidad de los autores involucrados (el por qué); dado que se trata de una tarea compleja involucran en el método al Ingeniero de Requerimientos como una figura esencial. Usarán los objetivos de la misma manera que son presentados en el *i* Framework*.

En resumen, en este trabajo, que fue expuesto en la Conferencia i* 2017, introducen la herramienta desarrollada para la estrategia propia *Actor Goals from Lexicon* (AGFL) que sirve para formalizar la elicitación; para ello utilizan como ejemplo un sistema de peajes. Cuentan los 3 pasos principales de la estrategia, que son la construcción del léxico (**Build Lexicon**), extracción de objetivos (**Extract Goals**) y refinamiento de objetivos (**Refine**

[3] La construcción de un objetivo blando como *tipo[tópico]* forma parte del *NFR Framework*.
[4] Esto se corresponde con la construcción *dependee-dependum-depender* del *i* Framework*.

Goals); usan en estos pasos conceptos y metodologías ya introducidos antes, como el lenguaje LEL (que sostienen es de fácil comprensión incluso para personas sin conocimientos técnicos), *NFR Framework* y *i* Framework*. Para evitar la representación de objetivos en estilo libre se adoptan marcos pre-definidos que sirven para guiar la representación de la intencionalidad de los *stakeholders*.

Aunque aquí los autores no se explayan sobre ciertos pasos del AGFL, sí especifican que se busca solucionar el problema de escalabilidad del *i* Framework* a través de una nueva estrategia de diseño de los objetivos mediante diagramas separados, llamados *Paneles de Intencionalidad* (*intentionality panels*), que tienen como función modularizar el modelo SR; los paneles a diferencia del modelo son diagramados individualmente separando por intencionalidad[5] mientras que no se representan tareas o recursos que aparecen en el modelo SR. Las relaciones entre objetivos son semánticamente los mismos que los adoptados por el modelo SR [24].

La herramienta AGFL fue desarrollada en PHP, Javascript y MySQL y hoy en día forma parte de la wiki i*. El AGFL se transformó luego en parte de un lenguaje de los mismos autores llamado ERi*c.

Si bien AGFL utiliza diferentes tipos de marcos para visualizar los símbolos LEL y objetivos, podemos deducir de la descripción de los autores que el proceso de elicitación es completamente manual (no utiliza otras herramientas disponibles como podrían ser los catálogos NFR o NFR más usuales por dominio de software), haciendo que toda la responsabilidad en el reconocimiento de NFR esté a cargo del Ingeniero de Requerimientos.

Para poder analizar la efectividad de la misma consideramos que debería aplicarse la herramienta en casos de estudio donde pueda analizarse si el problema de escalabilidad del *i* Framework* fue efectivamente resuelto, como era el propósito de Oliveira et al. También pensamos que sería interesante aplicar la estrategia en diferentes situaciones y dominios para obtener evidencia de los beneficios de aplicar el enfoque en casos reales.

3.2.3. Elicitación en Agile con Cloud (2020)

En esta última propuesta presentamos la metodología de Younas et al. para elicitar NFR en proyectos Agile que utilizan servicios Cloud [58]. Agile no es la excepción al resto de las metodologías en cuanto a que los NFR son ignorados mientras que los FR son tratados con prioridad; basta ver que las *user stories* en general elicitan requerimientos funcionales. Los motivos para esta falta de interés en los NFR son los mismos que en los proyectos tradicionales de software: el desconocimiento de parte del clientes y del equipo, la intrínseca dificultad de su elicitación y la falta de estándares que expliquen cómo incorporar NFR en la elicitación. Las consecuencias, no sorprende, son las mismas: proyectos que incorporan demasiado tarde los NFR y terminan por costar mucho más del estimado inicial, proyectos cancelados o fallidos [58, pag. 1].

Con el propósito, entonces, de incluir la fase de elicitación de NFR en Agile, los autores proponen una metodología que se basa en técnicas ya existentes para elicitar requerimientos no funcionales, a la que llamarán *NFRElicit*; usarán para facilitar la comunicación y colaboración servicios Cloud (como Google Doc, Dropbox, Team Foundation Server entre otros).

[5] Define tipos de relaciones como: correlación, con el mismo significado que el *NFR Framework*; contribución, con el mismo significado que el *i* Framework*; dependencia, con el mismo significado que el usado en los modelos SD.

Podemos ver los ocho pasos de la propuesta en la figura 3.3. Se comienza con una fase inicial de recolección de requerimientos con la colaboración de los *stakeholders*, que puede realizarse de distintos modos: con entrevistas, observación y análisis social, casos de uso, escenarios, reuniones, brainstorming, focus group [59, pag. 1-2]. Esto lleva a los requerimientos preliminares escritos en forma de texto libre (paso **Requirement statements**). Para descubrir NFR como ya hemos visto antes ayuda conocer el tipo de software que se quiere construir [4] así como los catálogos de NFR [1]: así surgen el paso **Software Context** que a partir de los requerimientos preliminares intentará analizar el tipo de software y como consecuencia los NFR más usuales en él, y el paso **Glossary** que utilizará los catálogos de NFR para ayudar a una elicitación más completa. Se utilizarán también herramientas automáticas que se basan en el reconocimiento y extracción de NFR a partir del texto libre (**Extraction approach**). Por último, los programadores pueden contar con la ayudar de un experto del dominio en caso de tener dudas o necesitar ayuda con el reconomiento de NFR (ya que como se ha dicho precedentemente, muchas veces los mismos programadores no tienen el conocimiento necesario sobre este tipo de requerimientos); este es el paso **Consult expert**. A esta altura, con los resultados de los pasos anteriores, los programadores analizarán problemáticas referidas a los NFR con la información resultante hasta el momento: es importante la participación de los *stakeholders* para responder dudas y del experto del dominio para preparar de manera correcta las preguntas a hacer sobre los requerimientos involucrados,[6] que constituye el paso **Issues identification**; cada pregunta debería tener su posible NFR asociado junto con sus atributos, dependencias y prioridad que deberán ser discutidos con los *stakeholders*. Una vez que las problemáticas son resueltas, se obtiene una lista con NFR candidatos que es discutida junto a *stakeholders* y expertos (paso **Candidate NFRs**); en caso que se encuentren conflictos o interdependencias entre elementos de la misma se vuelve al paso **Issues identification** y se itera hasta que no se presenten más conflictos. Cuando esto sucede es porque obtuvimos la lista final de NFRs elicitados y podemos avanzar en etapas sucesivas del proyecto de software (paso **NFRs ready**).

La metodología presentada fue evaluada en un proyecto de la Unión Europea llamado eProcurement (un sistema online para contrataciones públicas) y para evaluarlo se tomó como punto de referencia una clasificación hecha de manera manual utilizando otra metodología llamada NORMAP: de los 88 NFRs originales, 81 fueron identificados por *NFRElicit*, 5 tuvieron alguna diferencia con el NFR identificado por NORMAP y 1 no se identificó.

Este trabajo fue presentado en el año 2020 y es interesante ver cómo las problemáticas respecto a los NFRs continúan siendo las mismas: metodologías que ignoran la elicitación e integración de NFR (en este caso, Agile), FR considerados como requerimientos primarios mientras los NFR son olvidados o atendidos solamente en etapas de diseño e implementación, desconocimiento de este tipo de requerimientos por parte de los *stakeholders* y falta de estándares.

Los autores dan una metodología semi-automática para la elicitación de NFR en Agile, la cual tiene como objetivo ayudar a los programadores y *stakeholders* en el reconocimiento de los mismos. Se utilizan para ello un cúmulo de otras técnicas presentadas con anterioridad, como los catálogos de NFR, lista de NFR por tipo de sistema, técnicas de recolección

[6] Es importante en este punto, pero en general siempre, que los requerimientos estén escritos en el lenguaje del cliente.

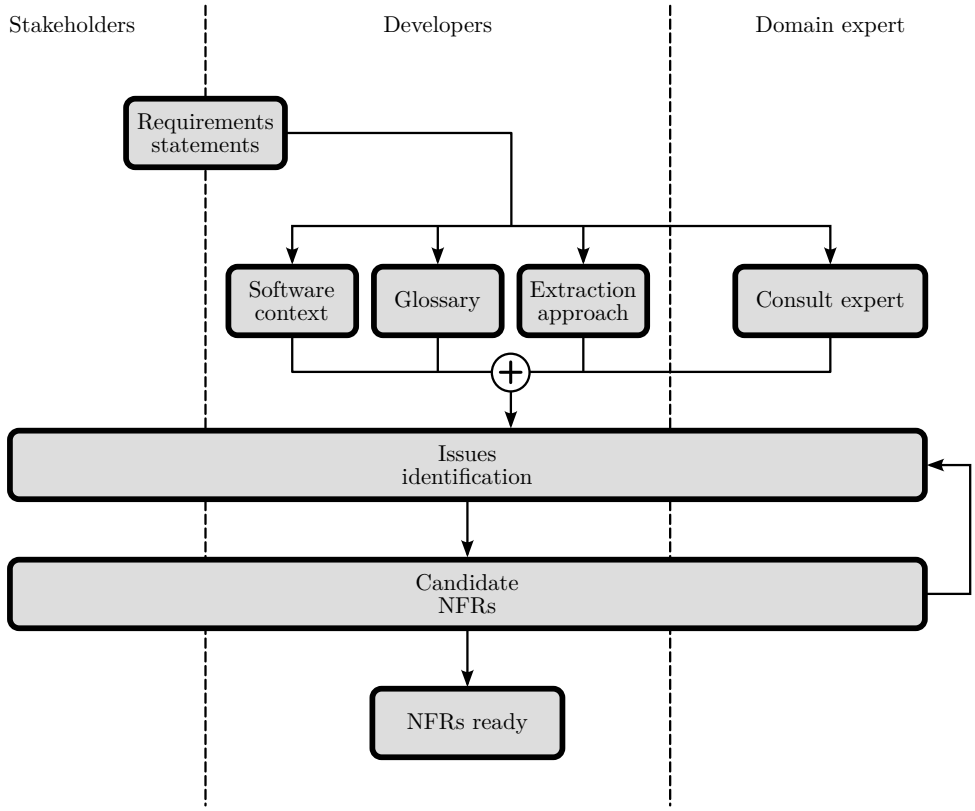

Fig. 3.3: Metodología *NFRElicit*

de requerimientos, entre otras. La técnica fue utilizada en un proyecto real en la industria, con buenos resultados.

NFRElicit propone una herramienta concreta para abordar el problema de los NFR en Agile y está respaldada por un estudio real en la industria del software. Todos sus pasos se basan en métodos/frameworks ya existentes y utilizados precedentemente, con lo cual se esperarían buenos resultados de esta nueva metodología. Creemos que en realidad *NFRElicit* podría ser utilizada en cualquier tipo de proyecto de software, siempre que se tenga una colaboración fluída con los *stakeholders*, ya que de los pasos presentados no está estrechamente ligada a la metodología Agile; el hecho de usar servicios Cloud puede ser interesante para las empresas que decidan utilizarla dado que permitiría reducir costos. La herramienta presentada se focaliza en la elicitación de NFR, no especificando cómo funcionaría la integración de los NFR con los FR, o cómo se podrían resolver conflictos surgidos; este punto podemos concluir que podría ser una desventaja de la misma. La técnica demostró tener buenos resultados en el caso real presentado por los autores, aunque consideramos que para sacar mejores conclusiones sobre su efectividad debería ser testeada más ampliamente, en diferentes tipos de proyectos de software e industrias.

3.3. Comparación de las metodologías post-2013

En la sección 3.2 analizamos algunas metodologías surgidas a partir del año 2013, elegidas según los criterios ya explicados.

Podemos observar que las tres técnicas analizadas se focalizan solamente en la etapa de elicitación de NFR sin ahondar en etapas sucesivas importantes como el análisis, negociación, diseño, etc.

Sachdeva y Chung [55] proponen una modificación a Scrum para proyectos con Cloud y IoT; la modificación al framework es específica por cada requerimiento no funcional (como sucedía en el *NFR Framework*) ya que utilizan fuertemente distintos conceptos de Scrum por cada requerimiento incorporado; así es que en el trabajo presentado analizan solamente dos, *security* y *performance*, por considerarse de los más críticos en proyectos con tecnologías Cloud y BigData, y que como suelen generar conflictos entre ellos se suelen analizar en conjunto. Son los únicos autores que presentarán una técnica para elicitar únicamente dos tipos de NFR.

Si tomamos la otra técnica que utiliza metodologías Agile y Cloud, es decir la de Younas et al. [58], vemos que sirve para tratar cualquier tipo de NFR sin necesidad de dar una estrategia particular para cada tipo; también se advierte que usa Cloud pero solo como servicio, para facilitar la colaboración entre los distintos actores del proyecto y no está así orientada a proyectos en Cloud. La primera diferencia que observamos con respecto al trabajo de Sachdeva y Chung es que, como ya dijimos, la propuesta es general en cuanto a los requerimientos, en el sentido de que sirve para cualquier tipo de NFR. Dado que Sachdeva y Chung modifican el ciclo de Scrum para incorporar *security/performance* a Scrum desde su unidad más básica, la *user story*, los autores están automáticamente integrando los NFR con los FR así como también tratando posibles conflictos que surjan entre estos tipos de requerimientos mediante la posibilidad de agregar nuevas *user stories* al *backlog*, aunque no tratan explícitamente los conflictos surgidos entre *performance* y *security* que consideramos un punto faltante de la metodología; esto presenta una diferencia con respecto a Younas et al. ya que no hablan de requerimientos funcionales en su estrategia, concentrándose solamente en elicitar los NFR y en los conflictos surgidos internamente (paso **Issues identification**). Vemos además que Younas et al. utilizan metodologías/técnicas previamente existentes (como los catálogos NFR o listados de NFR por tipo de sistema) pero sin modificar ninguno de ellos, solamente uniéndolos de manera estratégica para crear un nuevo método, en cambio Sachdeva y Chung realizan cambios concretos a un framework para incluir la elicitación de NFR. En *Scrum para Big Data y Cloud* el equipo entero de Scrum está involucrado en la elicitación (ya que la incorporación de los NFR comienza desde antes de crear las *user stories*) en cambio en *Elicitación en Agile con Cloud* pareciera que la responsabilidad recae principalmente en los programadores con algunas intervenciones del experto del dominio.

Por otro lado, Oliveira et al. [27] presentan la estrategia AGFL para elicitar objetivos duros y blandos con intencionalidad, basándose así en el trabajo de Yu y el *i* Framework*; la estrategia AGFL formó parte con el tiempo de un lenguaje de los mismos autores llamado ERi*c. Oliveira et al. tienen en común con los demás autores que también se focalizan solamente en la etapa de elicitación de los NFR, aunque utilizarán para ello viejas metodologías y lenguajes conocidos, como el *i* Framework*, *NFR Framework* y lenguaje LEL. La propuesta de los autores involucra fuertemente el conocimiento del equipo de programadores, al igual que la de Younas et al., y también como ellos da una estrategia general para cualquier tipo de NFR; a diferencia de Younas et al. cuya propuesta es semi-automática, Oliveira et al. están proponiendo un método manual con lo cual podemos deducir que requiere un mayor consumo del tiempo de los programadores; otra diferencia que podemos inferir en cuanto a la formación del equipo es que Oliveira et al. no hacen

mención a ninguna otra figura más allá de los programadores.

Entre Oliveira et al. y el resto de los autores cambia la manera de ver el mundo, así como cambiaba entre Yu y el resto de los autores de las metodologías anteriores al 2013: los primeros tomarán la visión del *i* Framework* y un mundo con intencionalidad, viendo así objetivos en vez de requerimientos y tratando de entender el por qué de la existencia de cada uno; en cambio a las otras dos técnicas no les interesará la intencionalidad en la elicitación y se basarán en requerimientos en vez de objetivos. Oliveira et al. son los únicos que tratarán todos los tipos de objetivos a la vez (duros y blandos), con lo cual desde este punto de vista es la más completa.

De las tres metodologías, Oliveira et al. son los únicos que no presentan un caso de estudio de un caso real donde se haya aplicado su técnica, por ende no hay información concreta respecto a su efectividad.

Por último, podemos notar que, a diferencia de lo sucedido con anterioridad al año 2013, las técnicas aquí discutidas no son completamente innovativas sino que se basan en otras ya existentes y hacen modificaciones a las mismas, mientras en cambio en 2.2 analizamos dos frameworks que introdujeron técnicas con nuevos conceptos y enfoques (*NFR Framework* y *i* Framework*) y que siguen influenciando la Ingeniería de Requerimientos como vimos en 3.1.

4. CONCLUSIONES

En este trabajo analizamos la importancia de los requerimientos no funcionales para la industria del software, sus principales problemáticas y estudiamos algunas de las metodologías surgidas en distintos momentos de la historia del software. Para esto dividimos nuestro análisis en dos momentos temporales: hasta el año 2012 incluído (año del *survey* realizado en el trabajo [5]) y desde el 2013 hasta el 2024. Presentamos también, con casos concretos cómo ignorar los requerimientos no funcionales ha llevado a desastres económicos en la industria (como los switches de AT&T o el bug FDIV de Pentium) o humanos (como el Therac-25, el TPS del Instituto Oncológico Nacional de Panamá o la Ambulancia de Londres).

En total se presentaron 7 metodologías para analizar diferentes etapas del ciclo de vida de los NFR de las cuales 4 son pre-2013 y las 3 restantes post-2013. No todas ellas se enfocan en todas las etapas de la Ingeniería de Requerimientos, y no todas funcionan por igual para cualquier tipo de requerimiento no funcional sino que es necesario proponer una técnica diferente para cada uno. Algunas se olvidan de la existencia de los requerimientos funcionales y por ende no afrontan la integración entre los diferentes tipos de requerimientos, otras en cambio no detallan un problema inherente a los NFR que son los conflictos inter-NFR.

Cuando de requerimientos no funcionales se trata, pareciera que no se puede tenerlo todo. Sin embargo, en algo coinciden los autores de las distintas metodologías y es en que los NFR son de vital importancia para cualquier proyecto complejo de software y que la integración de los mismos en los proyectos debe ocurrir más temprano que tarde.

La filosofía GORE tuvo un éxito no menor en el área de Ingeniería de Requerimientos y ha llevado a la creación de muchos métodos que la utilizan: ya en el año 2001, año del trabajo de Lamsweerde, se estaba utilizando en la industria en proyectos de distintos dominios, distintos tipos de proyectos y de distintos tamaños [12, pag. 11]. Dos de las metodologías más innovativas presentadas en este trabajo, *NFR Framework* y *i* Framework*, basan su desarrollo en el concepto de objetivos así como también aquellas que utilizan algunos de estos dos frameworks como un paso intermedio; entre ellas se encuentra *Intencionalidad en la elicitación de objetivos* del año 2017, lo cual demuestra que GORE sigue vigente hasta el día de hoy.

Comparando las metodologías pre-2013 con aquellas pos-2013 vemos que las primeras se basan en proyectos más tradicionales de software, seguramente por una cuestión de época. Las más recientes, en cambio, analizan la integración de los NFR en proyectos con tecnologías específicas, como Agile, Big Data o IoT. De todas maneras podemos observar que las técnicas de la vieja escuela siguen presentes: *Intencionalidad en la elicitación de objetivos* (3.2.2) es una metodología basada en los conceptos del *i* Framework*, *Elicitación en Agile con Cloud* (3.2.3) usa los catálogos de NFR como hace el *NFR Framework*, o *Scrum para Big Data y Cloud* (3.2.1) que usa la técnica de elicitar de a un NFR por vez (también como el *NFR Framework*).

Es interesante ver que aunque algunos de los trabajos más antiguos que analizamos ya señalaban la problemática de los NFR, como por ejemplo Boehm en su trabajo del año 1996 donde menciona la necesidad de incluir los atributos de calidad en el análisis de requerimientos para lograr obtener productos finales menos vulnerables a fallas [11, pag.1],

o Mylopolous et al. en 1992 que hablan de la falta de atención hacia los NFR [20, pag.1]; la situación, en esencia, parece no haber cambiado en más de tres décadas. Ejemplos de esta afirmación en el año 2017 son Sachdeva y Chung que explicitan que los NFR en general son ignorados o incluídos demasiado tarde en el ciclo de vida del desarrollo del software [55, pag.1], o Younas et al. que en el 2020 mencionan la falta de madurez de las metodologías de elicitación de los NFR [58, pag.1]. El problema, podemos decir, sigue vigente y sin soluciones claras que se adopten y usen en la industria.

No escapan a esta realidad de los NFR las metodologías más usadas en la actualidad, como puede ser Agile, que si bien surgió a inicios del 2000 es muy popular al día de hoy y fue aplicada por el 71 % de las compañias en Estados Unidos en el 2023[1], sin embargo en el presente los problemas presentados por los proyectos que utilizan metodologías Agile respecto a los que usan un desarrollo tradicional de software como puede ser waterfall, en cuanto a requerimientos no funcionales se refiere, expresan el mismo conjunto de preocupaciones [55] [58].

El modo de pensar los NFR no pareciera haber cambiado esencialmente desde el 2013 hasta la fecha. De las metodologías presentadas observamos más originalidad en las pre-2013: *i* Framework* propone una nueva estrategia completamente original de ver la Ingeniería de Requerimientos y presenta nuevos modelos para representar objetivos y sus relaciones con los agentes, junto a la estrategia BPR; el *NFR Framework* hace lo propio para los requerimientos no funcionales, en particular este último introduce el SIG, que fue utilizado luego en las propuestas sucesivas 2.2.3 y 2.2.4; incluso de las metodologías pre-2013 podemos decir que *Classpects como requerimientos no funcionales*, si bien no tuvo éxito posterior, da una propuesta original utilizando *aspects* para capturar *concerns transversales*. En cambio de las post-2013 no vemos desarrollos tan importantes o innovativos: de las propuestas analizadas todas están basadas en metodologías anteriores, hacen modificaciones a frameworks ya existentes para lograr elicitar los NFR, o buscan unir un cúmulo de propuestas no propias para formular una nueva.

Vale decir que incluso estas dos metodologías presentan problemas no menores que dificultan su integración en la industria, como las ya mencionadas falta de escalabilidad, modularidad y reusabilidad del *i* Framework* o el gran conocimiento de lógica requerido, el hecho de que la semántica de cada NFR deba ser dada individualmente y la dificultad para aceptar cambios del *NFR Framework*.

La falta de atención hacia los NFR sigue causando problemas en la industria: tal es el caso de los trenes en Polonia que no arrancaban luego de tener un servicio técnico [60], donde había dispositivos abiertos a conexión remota con el tren demostrando una grave falla de *security*, o las coordenadas de los centros de reparación, los números de serie de los componentes prefijados directamente en el código y la instrucción de uno de los trenes de "romperse" al llegar al millón de kilómetros que reflejan fallas de *mantainability* y *testability*; o el muy reciente incidente de CrowdStrike [61] que afectó a diversos tipos de industrias con un daño financiero estimado en mil millones de dólares, donde una actualización defectuosa de software que esperaba 21 parámetros de entrada en vez de los 20 recibidos calculados con una *regex* mostró serios problemas en cuanto a *robustness*, *maintainability* y *operability*, pudiendo también citar otros NFR no presentes en la tabla 3 de [4] como *upgradability*, lo que provoca un inevitable *déjà vu* al caso de los switchess 4ESS de AT&T analizado en 2.1.5, más de 30 años después.

Las temáticas tratadas, a pesar del paso del tiempo, parecen ser las mismas: los FR

[1] https://techreport.com/statistics/business-workplace/how-many-companies-use-agile/

son elicitados antes que los NFR si no exclusivamente, NFR que solamente aparecen en etapas de diseño o desarrollo del software, conflictos entre requerimientos, falta de conocimiento de los mismos, falta de estándares [58]. Los proyectos de software son cada vez más complejos, mucho más de lo planteaban Bajpai et al. en el 2012, pero la situación esencialmente no cambió. Si bien hay técnicas que afrontan problemas particulares con características específicas, no vemos una solución general a la problemática. Seguimos sin ver una metodología que se haya impuesto para tratar transversalmente a los NFR en las etapas que presentan mayor complejidad.

Bibliografía

[1] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.

[2] Christof Ebert. Putting requirement management into praxis: dealing with nonfunctional requirements. *Information and Software technology*, 40(3):175–185, 1998.

[3] Nishant Jha and Anas Mahmoud. Mining non-functional requirements from app store reviews. *Empirical Software Engineering*, 24:3659–3695, 2019.

[4] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM symposium on applied computing*, pages 311–317, 2010.

[5] Vikas Bajpai and Ravi Prakash Gorthi. On non-functional requirements: A survey. In *2012 IEEE Students' Conference on Electrical, Electronics and Computer Science*, pages 1–4. IEEE, 2012.

[6] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46, 2000.

[7] Tegegne Marew and Doo-Hwan Bae. Using classpects for integrating non-functional and functional requirements. In *IASTED International Conference on Software Engineering, as part of the 24th IASTED International Multi-Conference on APPLIED INFORMATICS*, pages 142–147. IASTED, 2006.

[8] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Nonfunctional requirements: From elicitation to conceptual models. *IEEE transactions on Software engineering*, 30(5):328–350, 2004.

[9] Martin Glinz. On non-functional requirements. In *15th IEEE international requirements engineering conference (RE 2007)*, pages 21–26. IEEE, 2007.

[10] NS Rosa, PRF Cunha, and GRR Justo. ProcessNFL: A language for describing non-functional properties. in proc. In *35th Hawaii International Conference on System Sciences*, 2002.

[11] Barry Boehm and Hoh In. Identifying quality-requirement conflicts. *IEEE software*, 13(2):25–35, 1996.

[12] A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pages 249–262, 2001.

[13] W Eric Wong, Xuelin Li, and Philip A Laplante. Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures. *Journal of Systems and Software*, 133:68–94, 2017.

[14] Nancy G Leveson and Clark S Turner. An investigation of the Therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[15] International Atomic Energy Agency. *Investigation of an accidental exposure of radiotherapy patients in Panama*. International Atomic Energy Agency Vienna, 2001.

[16] Anthony Finkelstein and John Dowell. A comedy of errors: the london ambulance service case study. In *Proceedings of the 8th international workshop on software specification and design*, pages 2–4. IEEE, 1996.

[17] José Carlos Soares Brandão and Alan Mercer. The multi-trip vehicle routing problem. *Journal of the Operational research society*, 49:799–805, 1998.

[18] Mark Janeba. The pentium problem. *The Pentium Problem*, 1995.

[19] Dennis Burke. All circuits are busy now: The 1990 AT&T long distance network collapse. *California Polytechnic State University*, 1995.

[20] John Mylopoulos, Lawrence Chung, Brian Nixon, et al. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on software engineering*, 18(6):483–497, 1992.

[21] Luiz Marcio Cysneiros and JC Sampaio do Prado Leite. Integrating non-functional requirements into data modeling. In *Proceedings IEEE International Symposium on Requirements Engineering (Cat. No. PR00188)*, pages 162–171. IEEE, 1999.

[22] Zhang Lin-lin, Ying Shi, Ni You-cong, Wen Jing, Zhao Kai, and Ye Peng. Towards multi-dimensional separating of nfrs in software architecture. In *2008 International Conference on Computer Science and Software Engineering*, volume 2, pages 104–107. IEEE, 2008.

[23] Hridesh Rajan and Kevin J Sullivan. Classpects: unifying aspect-and object-oriented language design. In *Proceedings of the 27th international Conference on Software Engineering*, pages 59–68, 2005.

[24] Eric S. K. Yu. Modeling strategic relationships for process reengineering. In *Social Modeling for Requirements Engineering*, 1995.

[25] Eric Yu, Paolo Giorgini, Neil Maiden, and John Mylopoulos. Social modeling for requirements engineering: An introduction. 2010.

[26] Xavier Franch, Lidia López, Carlos Cares, and Daniel Colomer. The i* framework for goal-oriented modeling. *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, pages 485–506, 2016.

[27] Antonio de Padua Albuquerque Oliveira, Julio Cesar Sampaio do Prado Leite, Luiz Marcio Cysneiros, and Wellington Gabriel Sampaio Da Silva. Eliciting goals and softgoals-how to perceive the intentionality at the beginning of the journey. In *iStar*, pages 31–36, 2017.

[28] Jennifer Horkoff and Eric Yu. Interactive goal model analysis for early requirements engineering. *Requirements Engineering*, 21:29–61, 2016.

[29] Hugo Estrada Esquivel. *A service-oriented approach for the i* framework*. PhD thesis, Universitat Politècnica de València, 2008.

[30] Fernanda Alencar, Jaelson Castro, Ana Moreira, João Araújo, Carla Silva, Ricardo Ramos, and John Mylopoulos. Integration of aspects with i* models. In *Agent-Oriented Information Systems IV: 8th International Bi-Conference Workshop, AOIS 2006, Hakodate, Japan, May 9, 2006 and Luxembourg, Luxembourg, June 6, 2006, Revised Selected Papers*, pages 183–201. Springer, 2008.

[31] Xavier Franch. Incorporating modules into the i* framework. In *Advanced Information Systems Engineering: 22nd International Conference, CAiSE 2010, Hammamet, Tunisia, June 7-9, 2010. Proceedings 22*, pages 439–454. Springer, 2010.

[32] Ezeldin Sherif, Waleed Helmy, and Galal Hassan Galal-Edeen. Proposed framework to manage non-functional requirements in agile. *IEEE access*, 11:53995–54005, 2023.

[33] Mert Ozkaya, Deniz Akdur, Etem Cetin Toptani, Burak Kocak, and Geylani Kardas. Practitioners' perspectives towards requirements engineering: A survey. *Systems*, 11(2):65, 2023.

[34] Kamaljit Kaur and Parminder Kaur. Sabdm: A self-attention based bidirectional-rnn deep model for requirements classification. *Journal of Software: Evolution and Process*, 36(2):e2430, 2024.

[35] Roberto Monaco, Xiufeng Liu, Teresa Murino, Xu Cheng, and Per Sieverts Nielsen. A non-functional requirements-based ontology for supporting the development of industrial energy management systems. *Journal of cleaner production*, 414:137614, 2023.

[36] Arif Nurwidyantoro, Mojtaba Shahin, Michel Chaudron, Waqar Hussain, Harsha Perera, Rifat Ara Shams, and Jon Whittle. Integrating human values in software development using a human values dashboard. *Empirical Software Engineering*, 28(3):67, 2023.

[37] Larissa Costa, Jaelson Castro, Judith Kelner, Bruno Jeronimo, Maria Lencastre, and Óscar Pastor. On the quest of trust requirements for socially assistive robots. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, pages 252–261. IEEE, 2023.

[38] José Miguel Horcas, Mónica Pinto, and Lidia Fuentes. Empirical analysis of the tool support for software product lines. *Software and Systems Modeling*, 22(1):377–414, 2023.

[39] Daniel De Pascale, Giuseppe Cascavilla, Mirella Sangiovanni, Damian A Tamburri, and Willem-Jan van den Heuvel. Internet-of-things architectures for secure cyber–physical spaces: The visor experience report. *Journal of Software: Evolution and Process*, 35(7):e2511, 2023.

[40] Hemanth Gudaparthi, Nan Niu, Yilong Yang, Matthew Van Doren, and Reese Johnson. Deep learning's fitness for purpose: A transformation problem frame's perspective. *CAAI Transactions on Intelligence Technology*, 8(2):343–354, 2023.

[41] Edgar Sarmiento-Calisaya and Julio Cesar Sampaio do Prado Leite. Early analysis of requirements using nlp and petri-nets. *Journal of Systems and Software*, 208:111901, 2024.

[42] Ana Moreira, João Araújo, Catarina Gralha, Miguel Goulão, Isabel Sofia Brito, and Diogo Albuquerque. A social and technical sustainability requirements catalogue. *Data & Knowledge Engineering*, 143:102107, 2023.

[43] Rohith Sothilingam and SK Eric. Toward a goal-oriented argumentation approach for fair ml measures using i. In *iStar*, pages 4–9, 2023.

[44] Thomas Olsson, Séverine Sentilles, and Efi Papatheocharous. A systematic literature review of empirical research on quality requirements. *Requirements Engineering*, 27(2):249–271, 2022.

[45] Thiago Menezes. A review to find elicitation methods for business process automation software. *Software*, 2(2):177–196, 2023.

[46] Sandro Bimonte, Hassan Badir, Pietro Battistoni, Houssam Bazza, Amina Belhassena, Christophe Cariou, Gerard Chalhoub, Juan Carlos Corrales, Adrian Couvent, Jean Laneurit, et al. Data-centric uml profile for agroecology applications: Agricultural autonomous robots monitoring case study. *Computer Science and Information Systems*, 20(1):459–489, 2023.

[47] Yeshica Isela Ormeño and Jose Ignacio Panach. Mapping study about usability requirements elicitation. In *International Conference on Advanced Information Systems Engineering*, pages 672–687. Springer, 2013.

[48] Luca Sabatucci, Massimo Cossentino, Claudia Di Napoli, and Angelo Susi. A model for automatic selection of iot services in ambient assisted living for the elderly. *Pervasive and Mobile Computing*, 95:101845, 2023.

[49] Erlânio Freire, Enyo Gonçalves, Marcos Oliveira, Luiz Guilherme Moreira Leite, and Sabrina Ketlen Colares Santos. Prisebot-a chatbot to assist in the development of istar. In *Proceedings of the 20th Brazilian Symposium on Information Systems*, pages 1–10, 2024.

[50] George Sidiropoulos, Chairi Kiourt, and Lefteris Moussiades. Crowd simulation for crisis management: The outcomes of the last decade. *Machine learning with applications*, 2:100009, 2020.

[51] Irina Rychkova, Eddy Kiomba Kambilo, Nicolas Herbaut, Oscar Pastor, Rene Noel, and Carine Souveyet. Technology-aware enterprise modeling: Challenging the model-driven architecture paradigm. In *International Conference on Business Process Modeling, Development and Support*, pages 388–396. Springer, 2024.

[52] István Koren, Matthias Jarke, Frank Piller, and Hoda ElMaraghy. Sustainability and resilience in alliance-driven manufacturing ecosystems: A strategic conceptual modeling perspective. 2023.

[53] Mandira Roy, Souvick Das, Novarun Deb, Agostino Cortesi, Rituparna Chaki, and Nabendu Chaki. Correlating contexts and nfr conflicts from event logs. *Software and Systems Modeling*, 22(6):1987–2010, 2023.

[54] Michiyo Wakimoto. Detecting context-dependent defects and measuring review quality for software reviews. 2023.

[55] Vaibhav Sachdeva and Lawrence Chung. Handling non-functional requirements for big data and iot projects in scrum. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*, pages 216–221. IEEE, 2017.

[56] Vikram Singh Prasher and Stephen Onu. The internet of things (iot) upheaval: overcoming management challenges. *The Journal of Modern Project Management*, 8(2), 2020.

[57] Antonio de Padua Albuquerque Oliveira, Julio Cesar Sampaio do Prado Leite, Luiz Marcio Cysneiros, and Claudia Cappelli. Eliciting multi-agent systems intentionality: from language extended lexicon to i* models. In *XXVI International Conference of the Chilean Society of Computer Science (SCCC'07)*, pages 40–49. IEEE, 2007.

[58] Muhammad Younas, Dayang Norhayati Abang Jawawi, Muhammad Arif Shah, Ahmad Mustafa, Muhammad Awais, Muhammad Kamran Ishfaq, and Karzan Wakil. Elicitation of nonfunctional requirements in agile development using cloud computing environment. *IEEE access*, 8:209153–209162, 2020.

[59] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pages 308–313. IEEE, 2003.

[60] Adam Haertle. Some heavyweight train hardware hacking. https://badcyber.com/dieselgate-but-for-trains-some-heavyweight-hardware-hacking/, 2023.

[61] CrowdStrike. Crowdstrike incident. https://www.crowdstrike.com/wp-content/uploads/2024/08/Channel-File-291-Incident-Root-Cause-Analysis-08.06.2024.pdf/, 2024.