



CHAPTER 4 (OBJECT)

Cs 303



JUNE 8, 2025
AHMED KHAMIS
Faculty of Science

4.3 Requirements Elicitation Concepts

في الجزء ده، الكاتب يشرح المفاهيم الأساسية لتجميع المتطلبات (elicitation)، اللي هي خطوة من خطوات تحليل النظام، وبيوضح الأقسام اللي هيكلت عنها:

المفاهيم الرئيسية اللي هيغطيها:

✓ 4.3.1 Functional Requirements .1 (المتطلبات الوظيفية) – سشن (هنشرحها حالاً)

4.3.2 Nonfunctional Requirements .2 (المتطلبات غير الوظيفية) – سشن

4.3.3 – Completeness, Consistency, Clarity, and Correctness .3 سشن

4.3.4 – Realism, Verifiability, and Traceability .4 سشن

4.3.5 – Greenfield Engineering, Reengineering, and Interface Engineering .5 سشن

4.3.1 Functional Requirements

"Functional requirements describe the interactions between the system and its environment independent of its implementation."

المتطلبات الوظيفية بتوصف التفاعل بين النظام والبيئة اللي حواليه، بعض النظر عن هو معمول إزاي أو بالتقنولوجيا إيه.

يعني بذكر على إيه اللي المفروض يحصل مش إزاي هيحصل، وده بيساعدنا نكتب متطلبات واضحة يقدر أي حد يفهمها سواء كان مطور أو محلل أو صاحب مشروع.

"The environment includes the user and any other external system with which the system interacts."

البيئة هنا المقصود فيها المستخدم وأي نظام خارجي بيتعامل مع النظام ده.

يعني أي حاجة برا النظام نفسه ويتتفاعل معها – سواء مستخدم بيضغط على زرار، أو سيرفر خارجي بيعته بيانات – دي اسمها "البيئة".

"For example, Figure 4-2 is an example of functional requirements for SatWatch, a watch that resets itself without user intervention."

الصورة 4-2 بتوضح مثال على متطلبات وظيفية لساعة SatWatch، وهي ساعة بتنظم نفسها أوتوماتيكي من غير ما المستخدم يتدخل.

ودي أهم نقطة، إن كل الحاجات اللي بتعملها الساعة بتحصل من غير تدخل بشري، وده بالضبط اللي بندور عليه في المتطلبات الوظيفية: نظام بيتصرف بناءً على تفاعله مع البيئة.

شرح المتطلبات الوظيفية في المثال (SatWatch):

"SatWatch is a wrist watch that displays the time based on its current location. SatWatch uses GPS satellites to determine its location and internal data structures to convert this location into a time zone."

الساعة SatWatch بتعرض الوقت حسب المكان اللي هي موجودة فيه دلوقتي. بتسخدم أقمار صناعية (GPS) علشان تحدد مكانها، وبعد كده بتحوله تلقائيًا لمنطقة زمنية مناسبة.

المعنى: المستخدم مش محتاج يقول الساعة هي فين. هي بتعرف وتتصرف.

"The information stored in SatWatch and its accuracy measuring time is such that the watch owner never needs to reset the time."

الدقة اللي شغالة فيها الساعة عالية جدًا لدرجة إن المستخدم مش هيحتاج يضبطها خالص.

المعنى: دي وظيفة النظام، إنه يفضل دايماً مطبوط، ودي متطلب وظيفي مهم.

"SatWatch adjusts the time and date displayed as the watch owner crosses time zones and political boundaries."

لما المستخدم يسافر من بلد للتانية أو من منطقة زمنية لتنانية، الساعة بتضبط نفسها أوتوماتيك.

المعنى: النظام بيفهم البيئة حواليه وينصرف بناء عليها.

"For this reason, SatWatch has no buttons or controls available to the user."

بما إن كل حاجة تلقائية، فالساعة مش فيها زرار ولا حاجة يقدر المستخدم يتتحكم فيها.

المعنى: ده بيأكيد إن كل التصرفات جوا النظام نفسه، مش معتمدة على أوامر من برا.

"During blackout periods, SatWatch assumes that it does not cross into a new time zone or a political boundary."

لو فقد الاتصال بالأقمار الصناعية (زي blackout)، الساعة بتثبت على المنطقة الزمنية اللي كانت فيها لحد ما الشبكة ترجع.

المعنى: النظام لازم يتصرف بذكاء في حالة فشل خارجي مؤقت.

"SatWatch has a two-line display showing, on the top line, the time (hour, minute, second, time zone) and on the bottom line, the date (day, date, month, year)."

الشاشة فيها سطرين: واحد للوقت بكل تفاصيله، والثاني للتاريخ.

المعنى: من ضمن وظائف النظام إنه يعرض المعلومات دي بشكل منظم.

"The display technology used is such that the watch owner can see the time and date even under poor light conditions."

الشاشة مصممة إمّا تبقى واضحة حتى لو الإضاءة مش كوبسّة.

المعنى: ده متطلّب وظيفي متعلّق بسهولة الاستخدام.

"When political boundaries change, the watch owner may upgrade the software of the watch using the WebifyWatch device..."

لو حصل تغيير في حدود الدول أو المناطق الزمنية، ممكن يعمل تحديث للساعة من خلال تطبيق ويّب أو جهاز معين.

المعنى: دي وظيفة لنظام — إنه يقدر يتّحدّث ويتكامّل مع أدوات خارجية.

4.3.2 Nonfunctional Requirements

"Nonfunctional requirements describe aspects of the system that are not directly related to the functional behavior of the system."

المتطلبات غير الوظيفية بتوصّف حاجات في النّظام مش مرتبطة بشكل مباشر بالسلوك أو الوظيفة الأساسية اللي النّظام بيعملها.

يعني بدل ما نقول "النّظام بيحسب كذا"، بنقول "النّظام لازم يكون سهل الاستخدام" ، أو "لازم يستغل بسرعة معينة" ، أو "لازم يكون متاح طول الوقت".

"Nonfunctional requirements include a broad variety of requirements that apply to many different aspects of the system, from usability to performance."

المتطلبات غير الوظيفية بتغطي جوانب كثيرة في النّظام، من أول سهولة الاستخدام حتّى الأداء والتّوافر والأمان.

الهدف منها ضمان إن النّظام يستغل بكفاءة، ويكون عملي وآمن وسهل ومناسب للناس اللي هتسخدمه.

"The FURPS+ model used by the Unified Process provides the following categories of nonfunctional requirements:"

فيه نوجّه اسمه FURPS+ بيساعّدنا نصنّف المتطلبات دي لمجموعات مفهومة.

:FURPS+ تقسيم الـ 

1. Usability – سهولة الاستخدام

"is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component."

هي سهولة استخدام النّظام من ناحية التّعلم، إدخال البيانات، وفهم النّتائج. زي مثلاً يكون الزرار واضح، الألوان متناسقة، والشرح بسيط في الواحّدة.

2. Reliability – الاعتمادية

"is the ability of a system or component to perform its required functions under stated conditions for a specified period of time."

هي قدرة النظام إنه يشتغل بشكل سليم وثابت لفترة معينة تحت ظروف معينة. تشمل مفاهيم زي:

• **Dependability**: النظام موثوق فيه و نتيجته سلية.

• **Robustness**: النظام يشتغل حتى مع مدخلات غلط أو في ظروف صعبة.

• **Safety**: النظام ميعملش كوارث ولا يتسبب في أضرار.

3. Performance – الأداء

"are concerned with quantifiable attributes of the system, such as response time, throughput, availability, and accuracy."

بنقيس فيها حاجات رقمية زي:

• **Response Time**: وقت الاستجابة.

• **Throughput**: كمية الشغل اللي يخلصها في وقت معين.

• **Availability**: مدى توفر النظام دايماً.

• **Accuracy**: الدقة في النتائج.

4. Supportability – سهولة الدعم والتعديل

"are concerned with the ease of changes to the system after deployment."

يعني النظام سهل يتعدل عليه بعد ما يتبني، سواء علشان يصلح مشكلة أو يضيف ميزة. تشمل:

• **Adaptability**: يشتغل في بيئات مختلفة.

• **Maintainability**: يصلح ويتطور بسهولة.

• **Installability**: سهل يتسطب.

• **Internationalization**: يشتغل بلغات وسوق مختلفة.

▪ **FURPS+ Plus** في إضافية: يتضمن متطلبات

■ Implementation:

"constraints on the implementation of the system"

يعني مثلاً لازم نكتب الكود بلغة معينة أو نستخدم أدوات معينة.

■ **Interface:**

"constraints imposed by external systems"

زي إن النظام لازم يتعامل مع نظام تاني بلغة معينة أو تنسيق بيانات معين.

■ **Operations:**

"constraints on the administration and management of the system"

زي مين يقدر يدخل ويعدل، ومين عنده صلاحيات.

■ **Packaging:**

"constraints on the actual delivery of the system"

زي متطلبات التحريم أو طريقة تسليمisoft وبر.

■ **Legal:**

"concerned with licensing, regulation, and certification issues"

زي إن النظام لازم يلتم بقوانين معينة أو معايير دولية زي ISO.



Figure 4-3 أمثلة لمتطلبات غير وظيفية على – SatWatch:

النوع	المطلب
Usability	أي مستخدم يقدر يستخدم الساعة ويفهم الرموز الدولية بسهولة
Reliability	لازم يحصل فشل صريح في نظام الـ GPS عشان يحصل خطأ في الساعة
Performance	الساعة لازم تعرض الوقت في خلال 5 دقائق من رجوع GPS
Supportability	لازم تقدير تعمل أبديت للساعة باستخدام تطبيق Webify



قيود تنفيذية (Constraints):

النوع	المتطلب
Implementation	السوفتوير لا يكتب بلغة Java
Interface	الساعة لا تلزم بمعايير

Section 4.3.3: Completeness, Consistency, Clarity, and Correctness

"Requirements are continuously validated with the client and the user. Validation is a critical step in the development process..."

المتطلبات دايماً بنراجعها مع العميل والمستخدم. ودي خطوة محورية جداً قبل التنفيذ.

الهدف إنك تتأكد إن كل اللي مكتوب فعلاً هو اللي العميل عايزه، ومفيش حاجة ناقصة، ومفيش حاجة غلط.

"...checking that the specification is complete, consistent, unambiguous, and correct."

بنراجع المواصفات ونتأكد إنها كاملة، منطقية، واضحة، وصحيحة.

كل كلمة من دول ليها معنى مهم في متطلبات المشروع، تعال نفهمهم واحدة واحدة.



1. Completeness

"It is complete if all possible scenarios through the system are described, including exceptional behavior."

يعني لازم يكون كل السيناريوهات اللي ممكن تحصل موصوفة، حتى الحالات النادرة أو الغريبة (زي فقد الاتصال).

لو حصل حاجة مش متفطية في المواصفات، دي مشكلة incompleteness

: مثال في الجدول (Table 4-1)

"The SatWatch specification does not specify the boundary behavior when the user is standing within GPS accuracy limitations of a state's boundary."

يعني لو المستخدم واقف على حدود بين ولايتين، والـ GPS مش دقيق جدًا، الساعة مش بتحدد هي في أي منطقة زمنية بالضبط.

الحل: تضيف متطلب بيقول إن الساعة مش لازم تغير المنطقة الزمنية أكثر من مرة كل 5 دقائق.



2. Consistency (مفيش تعارض)

"It is consistent if it does not contradict itself."

يعني مفيش متطلب بيقول حاجة، وبعدها متطلب تاني بيقول العكس.

مثال:

"A watch that does not contain any software faults and must provide an upgrade mechanism for downloading new versions of the software."

التناقض هنا إنك بتقول الساعة مفيهاش أي مشكلة، بس بتقول لازم يكون فيها نظام تحديث. طب ليه تحدثها لو مفيهاش مشاكل؟

الحل: صياغة المتطلبات بشكل يسمح بوجود أخطاء محتملة قابلة للتصحيح.

3. Unambiguous – غير غامض (واضح)

"A specification is unambiguous if exactly one system is defined (i.e., it is not possible to interpret the specification two (or more) different ways)."

يعني المتطلب ميتفهمش بكتذا معنى. لازم أي حد يقرأ يفهم نفس الحاجة بالضبط.

مثال:

"The SatWatch specification refers to time zones and political boundaries. Does the SatWatch deal with daylight saving time or not?"

العبارة مش واضحة، هل الساعة بتتبع التوقيت الصيفي ولا لا؟

الحل: تكتب نص صريح يقول "الساعة تدعم التوقيت الصيفي تلقائياً"، أو لا.

4. Correctness – الصحة (الدقة)

"It is correct if it represents accurately the system that the client needs and that the developers intend to build."

يعني الكلام اللي في المتطلبات يعبر فعلاً عن اللي العميل عايزه، ويقدر المطور ينفذه زي ما هو.

مثال:

"The watch must not contain any faults, as it is not verifiably anyway."

دي عباره ملهاش معنى تطبيقي، ومش ممكن تتأكد منها.

الحل: صياغة المتطلبات بحيث تكون قابلة للتحقق فعلاً – زي "يجب ألا يتتجاوز معدل الخطأ مرة واحدة في الأسبوع".

آخر فقرة ف السكشن:

يتكلم إنك ممكن تحتاج نموذج مبدائي (mock-up) علشان تعرض فكرة الساعة وتأخذ رأي المستخدمين.

مثال:

ممكن واحدة من المستخدمين تقول "أنا عايزه الساعة تعرض التاريخ بالطريقة الأمريكية والأوروبية مع بعض."

يعني لازم تفكّر دايماً في آراء المستخدمين من بدرى.

Section 4.3.4 – Realism, Verifiability, and Traceability

"Three more desirable properties of a requirements specification are that it be realistic, verifiable, and traceable."

فيه 3 صفات مهمة لازم تتوفر في أي مواصفات لمتطلبات المشروع: إنها تكون واقعية، قابلة للتحقق، ويمكن تتبعها.



1. Realistic – واقعية

"The requirements specification is realistic if the system can be implemented within constraints."

يعني المتطلبات تكون ممكن تنفيذها فعلاً في الظروف المتأحة (ميزانية – وقت – موارد – أدوات).

مثال مش واقعي:

"متوسط وقت الفشل (Mean Time To Failure) للساعة لازم يكون 100 سنة".

ده كلام صعب جداً يتنفذ، مين اللي هيجرب 100 سنة؟ دي متطلبات خيالية مش عملية.

 لازم تكتب المتطلبات بشكل يتناسب مع الواقع اللي انت شغال فيه.



2. Verifiable – قابلة للتحقق

"The requirements specification is verifiable if, once the system is built, repeatable tests can be designed..."

يعني بعد ما تبني النظام، تقدر تعمل عليه اختبار وتبت إن المتطلبات اتحققت فعلاً.

المطلب يكون **verifiable** يعني يفع نكتبه بطريقة تخليناقدر نختبره ونتأكد إن النظام فعلاً حققه.

يعني:

- أقدر أبني اختبار واضح.
- أقدر أكر الاختبار أكثر من مرة بنفس النتائج.

- أقدر أقول بثقة "المطلب ده حصل" أو "ماحصلش."

ليه المطلبات الغامضة مش **verifiable** 

- "The product shall have a good user interface."

كلمة "good" دي subjective، يعني كل واحد هيشفوها بطريقة.

- واحد يقول لازم تكون فيها animation
- واحد تاني شايف إن الأبيض والأسود كفاية.
- مفيش مقياس واضح للـ "good"

يعني لو سأناك: هل الواجهة جيدة؟ هتقول حسب رأي الشخصي... وده مش اختبار علمي ولا objective

 علشان نخلها verifiable، ممكن نقول:

- "The product shall comply with the WCAG 2.1 accessibility standard."
- "The product shall be error free."

ده معناه إن مفيش ولا Bug واحدة في النظام.

تعال فكر: إزاي هتثبت إن مفيش أي Error؟

- لازم تختبر كل الحالات المحتملة.
- عدد الحالات المحتملة في أي نظام كبير جداً (أحياناً لا نهائي).

يعني دي مش متطلب قابل للتحقق لأنه يستلزم اختبار مستحيل التنفيذ بالكامل.

 ممكن نكتبه بدل كده:

- "The system shall have no known critical defects at time of release."
- أو "The system shall pass 100% of test cases in the final test suite."
- "The product shall respond to the user within 1 second for most cases."

"most cases" مش عدد، مش نسبة، مش وقت... يعني إيه بالضبط؟ %90؟ %60؟ مرة من كل اتنين؟ مين يقرر؟

 لازم نحط نسبة محددة، وتقول هنجرب ده إزاي.

 الحل الصح زي الجملة دي:

"The system shall respond in less than 1 second for 95% of requests."

دي بقى متطلب **verifiable** جدًا، ليه؟

1. فيها مقياس واضح: أقل من ثانية.
2. فيها نسبة واضحة: 95% من الطلبات.
3. أقدر أكتب test case request1000 ، وأقيس الزمن، وتأكد إن 950 منهم استجابوا تحت 1 ثانية.
4. ده ينفع يتتقاس، يتسجل، ويتراجع عليه تاني.

تجربة عملية:

لو عندك API ، وحيث تقيس زمن الاستجابة (response time) :

- تبعت 1000 طلب.
- تقيس الزمن بناءً كل واحد.
- تحسب النسبة اللي استجبت أقل من ثانية.

لو طلعوا 1000/960 →  المتطلب اتحقق

لو طلعوا 1000/900 →  المتطلب ما تحققش لأن أنا قايل فوق أن 95% من الطلبات يتنفذ في أقل من ثانية
وده الفرق بين الكلام الإنساني والكلام القابل للاختبار فعلياً.

3. Traceable – قابلة للتتبع

"A requirements specification is traceable if each requirement can be traced throughout the software development..."

يعني كل متطلب مكتوب في مستند المتطلبات (Requirements Document) ، ممكن نعرف:

1. هو اتنفذ فين في الكود أو التصميم أو الواجهة.
2. ولو الكود اتغير، نعرف المتطلب اللي لازم نراجعه أو نعدله.

أهميتها:

- بتساعدك في كتابة الاختبارات (عشان تعرف كل وظيفة أصلها إيه).
- لو حصل تعديل، تعرف إيه اللي هيتأثر (تحليل الأثر – impact analysis)
- لو عميل غير رأيه، تقدر بسرعة تشووف إيه اللي هيتغير بناءً عليه.

مثال عملی:

لو عندك متطلب: "النظام يعرض الساعة والتاريخ"، تقدر تروح لكلاس TimeDisplay.java وتقول "آه هو ده الجزء اللي بينفذ المتطلب ده".

4.3.5 – Greenfield Engineering, Reengineering, and Interface Engineering

أولاً: Greenfield Engineering

"In greenfield engineering, the development starts from scratch—no prior system exists..."

يعني المشروع بيبدأ من الصفر تماماً، مفيش نظام قديم، ولا كود، ولا أي حاجة مبنية قبل كده.

المتطلبات بتيجي منين؟

من المستخدمين والعملاء مباشرة.
بنقعد معاهن نسألهم عايزين إيه، ونتخيل كل حاجة من البداية.

مثال:

شركة جديدة قررت تبني أول تطبيق لها لإدارة المخازن.
مفيش حاجة قدية ترجع لها، فكل المتطلبات هتتجمع من الناس اللي هتشتغل على النظام.

ساعة SatWatch اللي في الكتاب مثال على Greenfield Project

لأها اختراع جديد في سوق جديد، مكنش فيه نظام مشابه قبل كده.

ثانياً: Reengineering

"A reengineering project is the redesign and reimplementation of an existing system..."

يعني عندك نظام شغال فعلًا، بس قديم أو تقنيًا ما بيساعدش، فبتعيد بناءه وتطوره من جديد.

المتطلبات بتيجي منين؟

من النظام الحالي نفسه (نشوف يعمل إيه) + من المستخدمين (نشوف محتاجين إيه كمان)

مثال:

بنك شغال على برنامج قديم جدًا بلغة Cobol، وعايز يطوره بـ Java وواجهة حديثة.
بس الوظائف الأساسية هي: فتح حساب - تحويل فلوس - إصدار كروت.

ثالثاً: Interface Engineering

"An interface engineering project is the redesign of the user interface of an existing system."

يعني النظام شغال زي الفل، لكن الشكل والواجهة بناعاته بقت قديمة أو مش مريحة.

إاحنا بنعيد تصميم الواجهة فقط — لكن الوظائف اللي وراها بتنفصل زي ما هي.

المتطلبات بييجي منين؟

من مراجعة واجهة الاستخدام الحالية + آراء المستخدمين + تحسينات تجربة المستخدم (UX)

مثال:

نظام بريد إلكتروني شغال بشكل متاز، لكن الواجهة بناعاته شكلها 2005. فبنخلها تشبه Gmail العصري مثلاً، من غير ما نغير أي وظيفة في الكود الخلفي.

ملحوظة مهمة:

"This type of project is a reengineering project in which the legacy system cannot be discarded without high cost."

يعني لو النظام القديم مهم جداً ومش هينفع نرميه (لأنه غالى أو معقد)، ساعات بتعمل **Interface Engineering** كحل وسط: نسيب كل حاجة زي ما هي، ونغير بس الشكل الخارجي.

في كل الأنواع الثلاثة:

"Developers need to gather as much information as possible from the application domain..."

يعني سوا بتبدأ من الصفر أو بتعدل على القديم، لازم تعمل:

- مقابلات مع المستخدمين
- تقرأ الـ manuals القديمة
- ت Shawf الشيئات والملاحظات اللي بيستخدموها الناس
- تسأل الأسئلة الصعب

لو معرفتش تسأل صعب؟ مش هتجمع المتطلبات صعب، وهتغلط في تنفيذ النظام.

4.4 Requirements Elicitation Activities

In this section, we describe the requirements elicitation activities. These map a problem statement into a requirements specification that we represent as a set of actors, scenarios, and use cases.

في الجزء ده، هنشرح الأنشطة اللي بنستخدمها عشان نجمع متطلبات النظام من الناس. الهدف هو إننا نحول المشكلة اللي عند العميل إلى مواصفات واضحةقدر نشتغل عليها، وده بيتم عن طريق:

- الممثلين (Actors)
- السيناريوهات (Scenarios)
- حالات الاستخدام (Use Cases)

الأنشطة اللي سكشن ده يغطيها:

- Identifying Actors
- Identifying Scenarios
- Identifying Use Cases
- Refining Use Cases
- Identifying Relationships Among Actors and Use Cases
- Identifying Initial Analysis Objects
- Identifying Nonfunctional Requirements

كل واحدة من دول هتكلّم عنها سكشن سكشن، وهنبدأ دلوقتي بأول واحدة:

4.4.1 Identifying Actors

 أول حاجة: يعني إيه "Actor" ؟

"Actors represent external entities that interact with the system. An actor can be human or an external system."

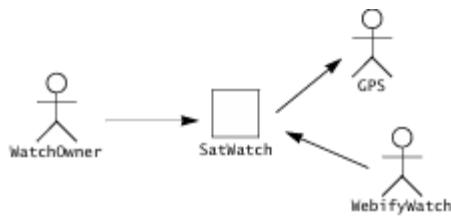
الممثل (Actor) هو أي كيان خارجي بيتفاعل مع النظام:

- يدخل له بيانات
- يستقبل منه بيانات
- يؤثر على سلوكه
- يتطلب منه تنفيذ حاجة

وال Actor ده ممكن يكون:

- إنسان (زي المستخدم أو الموظف)
- جهاز (زي الأتمار الصناعية أو الموبايل)

- نظام تابي (زي سيرفر أو قاعدة بيانات)



مثال 1: نظام SatWatch (من Figure 4-4) ✓

في الساعة الذكية، عندنا 3 Actors رئيسين:

Actor	وصف التفاعل
WatchOwner	صاحب الساعة - يلبسها، يتحرك بها، ويبيحوف الوقت
GPS Satellites	تبعد للساعة إحداثيات الموقع عشان تحديد المنطقة الزمنية
WebifyWatch Server	يحدث إعدادات الساعة، زي التوقيت الصيفي والتعديلات الرسمية

الشكل (Figure 4-4) يوضح العلاقات:

- الأسماء بتبيين اتجاه التفاعل.

- كل Actor يتفاعل مع SatWatch بشكل مختلف:

WatchOwner ○

GPS ○

WebifyWatch ○

* ملاحظة مهمة من المقررة:

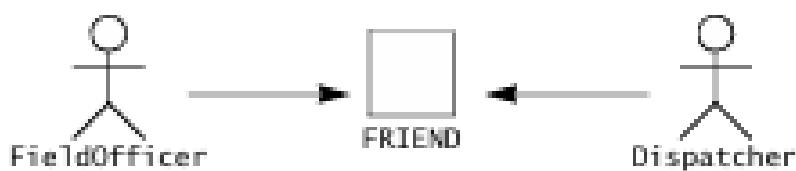
"They all exchange information with the SatWatch. Note, however, that they all have specific interactions with SatWatch..."

يعني: الثلاثة يتفاعلوا مع SatWatch، لكن كل واحد ليه دور مختلف.

مثال 2: نظام الطوارئ (Figure 4-5) ✓

"FRIEND, a distributed information system for accident management..."

ده نظام يستخدمه أكثر من شخص لإدارة الحوادث:



Actor	الوظيفة
FieldOfficer	ظابط مطافي أو شرطة في موقع الحدث
Dispatcher	يرد على مكالمات الطوارئ ويوزع الموارد

* الاثنين دول بيتفاعلوا مع نفس النظام، لكن:

- FieldOfficer بيستخدم موبايل أو جهاز محمول.

- Dispatcher يشتغل من جهاز مكتبي أو workstation

الشكل (Figure 4-5) بيؤي إن كل واحد عنده واجهة استخدام مختلفة → وده معناه إنهم Actors منفصلين.

لية بنفصل بينهم حتى لو بيستخدموا نفس النظام؟ 

"Actors are role abstractions and do not necessarily directly map to persons."

يعني: Actor مش شرط يكون شخص حقيقي، لكنه "دور".
ممكن شخص واحد يشغل كـ FieldOfficer في الصباح، و Dispatcher بالليل.
بس عشان يقوم بوظيفتين مختلفتين، نعتبرهم Actors منفصلين.

طب ليه هم نحدد الـ Actors في الأول؟ 

"The first step of requirements elicitation is the identification of actors."

لأن ده:

- يحدد نطاق النظام (مين بيتفاعل معاه ومين لا).
- يكشف وجهات النظر المختلفة اللي لازم نراعيها في التصميم.
- يساعدنا نبدأ نكتب Use Cases لكل Actor (وده اللي هنشرحه في 4.4.3).

أداة مساعدة: الأسئلة اللي تساعدك تحدد الـ Actors 

Questions for identifying actors:

- أي مجموعات مستخدمين بيستخدموا النظام في شغفهم؟
- مين بيشغل الوظائف الأساسية للنظام؟
- مين مسؤول عن الصيانة أو الإدارة؟

- النظام هيكل مع أي أجهزة أو أنظمة ثانية؟

:FRIEND  أمثلة على Actors في نظام

لو بصيت كويس في النص، هتلaci أمثلة كثير لاتجاهات التفكير:

Actor محتمل	وظيفته في النظام
Firefighter	يستجيب لحوادث
Dispatcher	ينسق الموارد
EPA Database	يوفر معلومات عن المواد الخطرة
Mayor / Governor	ممكن يراقب النظام أو يطلع تقارير
Technical Admin	مسؤول عن صيانة النظام
External Tool	يتكامل مع النظام من بزا

لكن مش لازم تعتبر كل دول Actors مختلفين.

"We need to consolidate this list into a small number of actors..."

يعني لازم تبص مين اللي ليه وظائف متشابهة، وتضمهم في Actor واحد.

Section 4.4.2 – Identifying Scenarios

"A scenario is 'a narrative description of what people do and experience as they try to make use of computer systems and applications' (Carroll, 1995)."

السيناريو هو وصف "قصصي" يبحكي المستخدم بيعمل إيه وبيشوف إيه وهو بيتفاعل مع النظام.

يعني: بنحكي قصة قصيرة من وجهة نظر مستخدم بيستخدم النظام في موقف معين.

"A scenario is a concrete, focused, informal description of a single feature of the system from the viewpoint of a single actor."

السيناريو عبارة عن وصف واقعي، بسيط، ومركز على ميزة واحدة من النظام، من وجهة نظر Actor واحد فقط.

 يعني مش تحليل نظري أو عام، ده موقف عملي بيحصل فعلًا.

الفرق بين السيناريو وال Use Case 

"Scenarios cannot (and are not intended to) replace use cases, as they focus on specific instances and concrete events (as opposed to complete and general descriptions)."

السيناريوهات مش بديلة لـ Use Cases، لأن السيناريو يركز على حالة معينة حصلت فعلاً،
لكن الـ Use Case يوصف كل الحالات الممكنة اللي ممكن تحصل مع الـ Actor

"However, scenarios enhance requirements elicitation by providing a tool that is understandable to users and clients."

السيناريوهات مفيدة جدًا في تحليل المتطلبات لأنها واضحة وسهلة الفهم للمستخدمين والعملاء.

Figure 4-6 مثال عملی في ✅

سيناريو حالة حريق في مخزن warehouseOnFire

Scenario name	warehouseOnFire
Participating actor instances	bob, alice:FieldOfficer john:Dispatcher
Flow of events	<ol style="list-style-type: none">Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the "Report Emergency" function from her FRIEND laptop.Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene, given that the area appears to be relatively busy. She confirms her input and waits for an acknowledgment.John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.Alice receives the acknowledgment and the ETA.

Figure 4-6 warehouseOnFire scenario for the ReportEmergency use case.

1. Bob وهو سائق في الشارع يشوف دخان طالع من مخزن Alice → بتفعل خاصية "Report Emergency" من تطبيقها.
2. Alice بتدخل عنوان الحريق، ووصف للمكان، وتطلب فرق إطفاء ووحدات إسعاف.
3. John (Dispatcher) يوصله إنذار على جهازه، يقرأ البيانات، ويبعث فرقاً ووحدتين إسعاف للموقع.
4. Alice يجيئها تأكيد + الوقت المتوقع للوصول (ETA)

🎯 ده سيناريو واقعي، محدد، ويشرح نقطة دخول معينة للنظام، من منظور Actor واحد أو اثنين، مش لازم يغطي كل الحالات الممكنة.

"It does not attempt to describe all possible situations in which a fire incident is reported."

يعني مش لازم السيناريو يشمل كل أنواع الحريق — هو مجرد مثال حالة واحدة.

"To describe the outcome of a decision, two scenarios would be needed, one for the 'true' path, and another one for the 'false' path."

لو عايز توصف نظام بيقرر بناءً على اختيار، هحتاج أكثر من سيناريو (واحد للطريق الصحيح، واحد لو القرار اترفض مثلاً).

✓ من Carroll, 1995): أنواع السيناريوهات

النوع	المعنى
As-is	بيوصف الوضع الحالي بدون تغيير، يعني إزاي النظام شغال دلوقتي فعلياً
Visionary	بيوصف حاجة لسه في الخطة، هدف مستقبلي عايزين نوصله
Evaluation	بيوصف سيناريوهات اختبارية نستخدمها لقياس تجربة المستخدم
Training	بنستخدمه لتدريب المستخدمين الجدد على النظام

أسئلة تساعدك تبني سيناريو: 

Questions for identifying scenarios:

- ما المهام التي ينفذها الـ Actor باستخدام النظام؟
- أي بيانات يتعامل معها؟ مين ينشئها؟ مين يعدلها؟
- إزاي النظام يعرف إن في تغير خارجي؟ مين اللي بيبلغه؟ إزاي وامتنى؟
- إزاي النظام يبلغ المستخدم؟ وبأي سرعة؟

أدوات المساعدة في كتابة السيناريو: 

"Developers use existing documents about the application domain to answer these questions..."

زي:

- كتيبات التشغيل القديمة
- خبرات المستخدمين
- مقابلات مع العملاء
- شيتات فيها ملاحظات من العمل

* والمهم إنك ما تكتبش السيناريو من دماغك، بل من واقع الاستخدام الحقيقي.

أمثلة على سيناريوهات إضافية من FRIEND: 

السيناريو	الوصف
warehouseOnFire	حريق في مخزن - ضباط يكتشفوه ويرسلوا موارد
fenderBender	حادث بسيط - الشرطة تسجل الحادث وتدير المرور

catInATree	قطة محشورة – تدخل بسيط يتأخر في الوصول
earthquake	زلزال كبير – النظام ينسق موارد متعددة وأطراف مختلفة

✓ Section 4.4.3 – Identifying Use Cases

A scenario is an instance of a use case; that is, a use case specifies all possible scenarios for a given piece of functionality.

السيناريو عبارة عن حالة واحدة فقط من **Use Case** أما **Use Case** فهو يوصف كل السيناريوهات المحتملة اللي بتحصل حواليين ميزة معينة.

مثال:

Use Case: Report Emergency •

Scenarios: warehouseOnFire – fenderBender – catInATree •

A use case is initiated by an actor. After its initiation, a use case may interact with other actors, as well.

يعني اللي بيبدأ Use Case دايماً هو Actor معين. لكن أثناء تنفيذ الخطوات ممكن يكون فيه تفاعل مع Actors تانين.

رزي:

يبدأ البلاغ FieldOfficer •

يتفاعل ويكل الاستجابة Dispatcher •

✓ شرح Figure 4-7 – Use Case: ReportEmergency

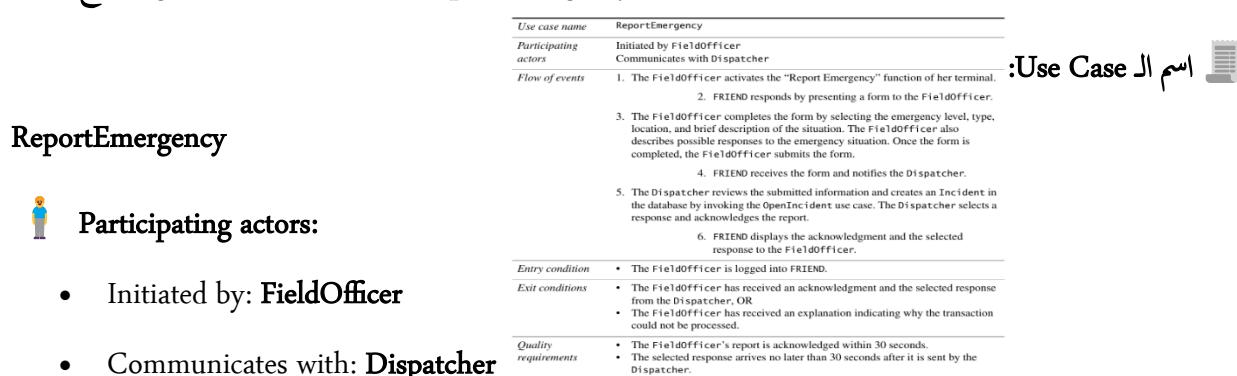


Figure 4-7 An example of a use case, ReportEmergency. Under ReportEmergency, the left column denotes actor actions, and the right column denotes system responses.

Flow of Events (تسلسل الأحداث)

"Report Emergency" يفعل خاصية **FieldOfficer** .1

2. النظام FRIEND يعرض فورمة بلاغ.
3. FieldOfficer يكتب التفاصيل (الموقع - النوع - الحالة - الاستجابات المطلوبة).
4. Dispatcher يبعث البلاغ لـ FRIEND
5. Dispatcher يراجع، ينفذ الخطوات، ويعت تأكيد.
6. FieldOfficer يعرض الرد للـ FRIEND

الخطوات من 1 إلى 3 يبدأها الد Actor (FieldOfficer)

الخطوات من 4 إلى 6 يكملها النظام و Actors تأكيد

Entry Conditions:

The FieldOfficer is logged into FRIEND.

يعني لازم يكون داخل على النظام الأول.

Exit Conditions:

البلاغ تم استقباله واترد عليه •
أو

النظم يشرح ليه الطلب اترفض •

Invariants:

مثلاً، لازم يتم الرد خلال 30 ثانية - ولو أتأخر، يتم تسجيل المشكلة.

أهمية الد Use Case

• يشرح كل السيناريوهات المرتبطة بوظيفة واحدة.

• يحدد:

◦ مين يبدأها؟

◦ إمّي تخلص؟

◦ إيه الحالات اللي تنجح أو تفشل فيها؟

◦ إيه الشروط اللي لازم تتتوفر؟

- التفاعلات اللي بتحصل جوه النظام.

ملحوظة مهمة:

The name of a use case should be a verb phrase denoting what the actor is trying to accomplish.

اسم الـ Use Case لازم يوضح الهدف من وجهة نظر الـ Actor
يعني قول Emergency Report مش ReportEmergency

مش اسم الميزة

لكن الفعل اللي بيعمله المستخدم

Use Case – Figure 4-8

"Use Case Writing Guide"

نصائح كتابة الـ Use Case:

القاعدة	الشرح
Use verb phrases	استخدم أفعال توضح العمل مش الكيان (ReportEmergency) EmergencyReport)
Be specific	خليك محدد في الأسماء والوظائف
Focus on system behavior	ما تكتبش الواجهة، اكتب سلوك النظام (مش الزرار، لكن الفعل اللي بيعمله الزرار)
Include exceptions separately	لو في فشل أو شرط خاص، أكتبه في قسم منفصل
Avoid UI-specific terms	ما تقولش "يضغط على زر X"، لكن قل "يرسل البلاغ"

مثال توضيحي من Figure 4-9

بيقارن بين كتابة صح وغلط:

Good name	Bad name
ReportEmergency	What is the user trying to accomplish?

| Good: "FieldOfficer reports the accident."

| Bad: "An ambulance is dispatched."

ليه؟ لأنه بيشرح حاجة بتحصل، لكن مش بيحدد مين عمل إيه ومين بدأ



Section 4.4.4 – Refining Use Cases

Use case name	ReportEmergency
Participating actors	Initiated by FieldOfficer Communicates with Dispatcher
Flow of events	<ol style="list-style-type: none">1. The FieldOfficer activates the "Report Emergency" function of her terminal.2. FRIEND responds by presenting a form to the officer. <i>The form includes an emergency type menu (general emergency, fire, transportation) and location, incident description, resource request, and hazardous material fields.</i>3. The FieldOfficer completes the form by specifying minimally the emergency type and description fields. The FieldOfficer may also describe possible responses to the emergency situation and request specific resources. Once the form is completed, the FieldOfficer submits the form.4. FRIEND receives the form and notifies the Dispatcher by a pop-up dialog.5. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. <i>All the information contained in the FieldOfficer's form is automatically included in the Incident. The Dispatcher selects a response by allocating resources to the Incident (with the AllocateResources use case) and acknowledges the emergency report by sending a short message to the FieldOfficer.</i>6. FRIEND displays the acknowledgment and the selected response to the FieldOfficer.
Entry condition	• ...

Figure 4-10 Refined description for the ReportEmergency use case. Additions emphasized in *italics*.

Figure 4-10 is a refined version of the ReportEmergency use case.

الشكل 4-10 هو نسخة متطرفة (أو محسنة) من Use Case اسمه ReportEmergency اللي شرحناه قبل كده.

It has been extended to include details about the type of incidents known to FRIEND and detailed interactions indicating how the Dispatcher acknowledges the FieldOfficer.

النسخة دي أضافت معلومات أكثر:

- أنواع الحوادث اللي النظام FRIEND يعرفها.
- وطريقة التفاعل بين Dispatcher و Field Officer ، خصوصاً إزاي ال Dispatcher بيبعد تأكيد إنه استقبل البلاغ.



Figure 4-10 Breakdown – Refined ReportEmergency

العنصر	التفاصيل
Use case name	ReportEmergency
Participating actors	Initiated by FieldOfficer, Communicates with Dispatcher

Flow of Events (تفاصيل أكثر):

.1 FieldOfficer يشغل وظيفة "Report Emergency" من جهازه.

.2 FRIEND يعرض فورمة بلاغ فيها تفاصيل زي:

◦ نوع الطوارئ (حريق، إصابة، تسرب كيماوي...)

◦ الموقع

◦ وصف الحادث

◦ الموارد المطلوبة

3. يملأ الفورم ويكتب: **FieldOfficer**

◦ نوع الطوارئ بالتحديد

◦ وصف الحالة

◦ الموارد المطلوبة (دي حاجة جديدة مضافة في النسخة المحسنة)

4. FRIEND يستقبل البيانات، ويظهر إشعار (Pop-up) لـ **Dispatcher**

5. Dispatcher يراجع البلاغ، ويستخدم **OpenIncident** Use Case عشان يسجل البلاغ في قاعدة البيانات.

◦ كان بيختار استجابة مناسبة من النظام (باختيار الموارد)

◦ ويعتبر تأكيد لـ **FieldOfficer** على شكل رسالة قصيرة

6. FRIEND يعرض الرد لـ **FieldOfficer**

✓ الفرق بين النسخة العادي والمحسنة:

النسخة المحسنة أضافت:

• تفاصيل دقيقة عن أنواع الحوادث والموارد.

• آلية لإرسال التأكيد من **Dispatcher**

• تفاعل أكبر بين الطرفين → وده يعكس الواقعية.

✓ ليه بنعمل Use Case Refinement لـ ؟

Refined scenarios can be used to define the functionality of the system...

يعني لما نبدأ بكتابة Use Case، غالباً بكتتها بشكل عام، لكن مع الوقت وإضافة التفاصيل:

• نبدأ نوضح أكثر إيه اللي يحصل

• ونضيف شروط، استثناءات، حالات تفاعل معقدة، إلخ

❖ دا مهم جداً عشان:

- نحدد حدود النظام بدقة
- نوضح التفاعلات بالتفصيل
- نكمل المتطلبات اللي السيناريوهات ما غطتهاش

كان بيقول: 

The refinement of use cases yields increasingly more details about the features provided by the system and the constraints associated with them.

كل ما نحسن الـ Use Case، بنفهم أكثر:

- إيه الميزات اللي لازم النظام يوفرها
- وايه القيود (زي الوقت، الموارد، الصلاحيات)

وأخيرًا بيقول في آخر العمود: 

The following aspects of the use cases, initially ignored, are detailed during refinement:

إيه اللي بيضاف في عملية التحسين؟ 

1. **Data elements:** زي الفورم في المثال – إيه الحقول المطلوبة؟ مين يكتبيها؟
2. **Actor-level interaction:** تفاصيل من بيرد على مين؟ ويايه؟
3. **Access rights:** هل ال Actor ده ليه صلاحية ينفذ Use Case معين؟
4. **Preconditions/Postconditions:** إيه لازم يكون موجود قبلها؟ وايه المفروض يحصل بعدها؟
5. **Alternative flows:** لو حصل فشل أو اختيار تاني؟ إيه اللي يحصل؟

مثال مبسط: 

"Place Order" اسمه Use Case
في الأول شكتبه كده:

- المستخدم يضيف منتجات، يضغط "طلب"، يظهر تأكيد.

لما نعمل refinement:

- نضيف أنواع المنتجات، خصومات، الكمية، طريقة الدفع، حالة المخزون
 - ونوضح إيه يحصل لو الدفع فشل؟ أو لو المنتج خالص؟
- نفس اللي حصل مع ReportEmergency هنا.

Section 4.4.5 – Identifying Relationships among Actors and Use Cases

Even medium-sized systems have many use cases. Relationships among actors and use cases enable the developers and users to reduce the complexity of the model and increase its understandability.

حتى الأنظمة المتوسطة فيها Use Cases كثير. عشان كده، ربط العلاقات بين الـ actors والـ use cases يدخل المفهوم أبسط وأسهل للفهم.

❖ فيه ٣ أنواع من العلاقات هننشر حم دلوقتي:

Communication Relationships .1

Extend Relationships .2

Include Relationships .3

أول نوع: 

Communication relationships between actors and use cases represent the flow of information during the use case.

يعني ببساطة:

• مين اللي بيبدأ الـ use case ؟

• ومنين بيتفاعل معاه بس مش بيبدأ ؟

 Figure 4-11 مثال واضح على العلاقات –

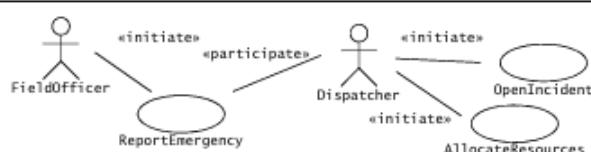


Figure 4-11 Example of communication relationships among actors and use cases in FRIEND (UML use case diagram). The FieldOfficer initiates the ReportEmergency use case, and the Dispatcher initiates the OpenIncident and AllocateResources use cases. FieldOfficers cannot directly open an incident or allocate resources on their own.

في النظام :FRIEND

العنصر	معناه
FieldOfficer ← «initiate» ← ReportEmergency	الـ FieldOfficer هو اللي بيبدأ البلاغ

Dispatcher ← «participate» ←
ReportEmergency

الـ Dispatcher يتفاعل مع البلاغ بعد ما يوصله، لكن مش
يبدأه

وده يساعدنا نفهم:

- مين ليه حق البدء؟
- ومين بس بيتلقى ويشارك في باقى الخطوات؟

النوع الثاني: Extend Relationships

A use case extends another use case if the extended use case may include the behavior of the extension under certain conditions.

يعني فيه Use Case أساسي، وفيه واحد تاني بيكل عليه أو بيشتغل لو حصلت حالة معينة.



مثال من الحياة – Figure 4-12:



Figure 4-12 Example of use of extend relationship (UML use case diagram). ConnectionDown extends the ReportEmergency use case. The ReportEmergency use case becomes shorter and solely focused on emergency reporting.

لو الـ FieldOfficer بيعت بلاغ، وخفأة دخل نفق فقد الاتصال → بدأ Use Case اسمه ConnectionDown

العنصر	المعنى
ReportEmergency	الحالة الأساسية – تقديم البلاغ
ConnectionDown	الحالة الخاصة – بتشتغل بس لو الاتصال اقطع

* الهدف من العلاقة دي:

- تفصل الحالات الخاصة عن السيناريو العام.
- تخلي الكود والموديل أبسط وأسهل للصيانة.

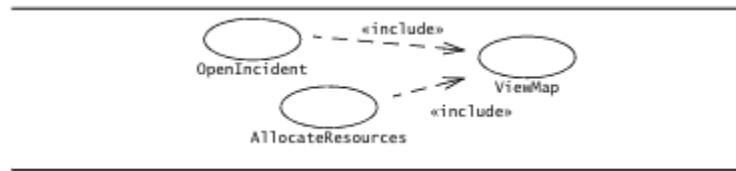
النوع الثالث: Include Relationships

Redundancies among use cases can be factored out using include relationships.

يعني لو في خطوات مشتركة بين Use Cases كتير → نطلعها في Use Case منفصل، و"تضمنها" لما تحتاج.



مثال عملی – Figure 4-13



في النظام، الـ Dispatcher يمكن يحتاج يشوف خريطة المدينة سواء في:

- **OpenIncident**

- أو **AllocateResources**

بدل ما نكرر خطوات عرض الخريطة في الاثنين، نعمل Use Case جديد اسمه **ViewMap** ونكتب إن الـ Use Cases الثانية تشمل **include** عليه.

كده وفرنا وقت، وخلينا التصميم modular

Include and extend are similar constructs, and initially it may not be clear to the developer when to use each one.

العلاقتين شبه بعض في الشكل (خط منقطع + سهم)،
بس بخدموا أهداف مختلفة تماماً.

الفرق الأساسي:

For include relationships, the event triggering the target use case is described in the flow of event of the source use case.

يعني: في include، الحدث الأساسي نفسه يستدعي الـ use case الثاني من جوه الخطوات.
يعني الخطوات بتاعة use case (أ) تشمل خطوة بتنفذ use case (ب).

For extend relationships, the event triggering the source (i.e., extending) use case is described in the source use case as a precondition.

أما في extend، الـ use case الثاني يتكلّم الأساسي لو حصل شرط معين.
يعني مش دايماً بيشتغل، بس بيشتغل في ظروف خاصة.

الفكرة العملية:

العلاقة	بنستخدمها لما...	مثال
Include	الوظيفة بتحصل دايماً، ومشتركة في أكثر من use case	OpenIncident في included ViewMap و AllocateResources
Extend	الوظيفة بتحصل في ظروف خاصة فقط	ConnectionDown extends ReportEmergency الاتصال انقطع



Figure 4-14 شرح



في المثال ده، نفس use case **ConnectionDown**

- في الرسم اللي على اليمين: معمول له **extend**

ييشغل بس لما يحصل "فقدان اتصال" أثناء ال ReportEmergency

- في الرسم اللي على الشمال: معمول له **include**

يعني وظيفته الأساسية مشتركة وجزء من الخطوات المعتادة

* أدق وأوضح في الحالات دي عشان بتفصل الطبيعي عن الاستثنائي.

ملحوظة مهمة:

A behavior that is strongly tied to an event and that occurs only in a relatively few use cases should be represented with an included relationship.

لو فيه سلوك بيحصل قليل، لكن بيظهر في كذا → include use case الأفضل

لو بيحصل قليل وفي use case واحد → الأفضل extend

الخلاصة على لسان الكتاب:

In summary, the following heuristics can be used for selecting an extend or an include relationship.

جدول Heuristics (القواعد الذكية):

القاعدة	متى نستخدمها؟	مثال
= Extend = استثناءات أو حالات نادرة أو اختيارية	زي لو حصل عطل أو فشل	عطل في سيارة إطفاء

Include = خطوات متكررة في أكثر من use case	زي فتح خريطة - تسجيل بيانات - إرسال تقرير	ViewMap
متداخلش في التقسيم	لو قسمنا كل use case لـ 10 صغيرين، الدنيا هتبقى فوضى	اعمل شوية Use Cases كبار مفصلين أفضل

الهدف الأساسي:

The purpose of adding include and extend relationships is to reduce or remove redundancies...

الغرض من العلاقات دي:

- نوفر وقت ومنع التكرار.
- نفصل الحالات العاديّة عن الحالات الخاصة.
- نخلّي النموذج أنظف وأسهل في الفهم والصيانة.

Section 4.4.6 – Identifying Initial Analysis Objects

One of the first obstacles developers and users encounter when they start collaborating with each other is differing terminology.

أول مشكلة بتحصل لما المطورين يبدأوا يستغلوا مع المستخدمين هي إن كل واحد بيستخدم مصطلحات مختلفة لنفس الحاجة.

Misunderstandings result from the same terms being used in different contexts and with different meanings.

يعني مثلاً: المستخدم يقصد بكلمة "incident" حاجة، والمبرمج فاهم حاجة تانية خالص → يحصل سوء تفاهم.

طيب نحلها إزاي؟

To establish a clear terminology, developers identify the participating objects for each use case.

لازم نبدأ نحدد ونوحد المفاهيم.
يعني:

- كل حاجة (object) بتشارك في الـ use case → نعرفها باسم واضح
- ونعمل Glossary فيها الأسماء والتعرifات الموحدة

:Glossary الـ

The glossary is included in the requirements specification and, later, in the user manuals.

يعني هنضيغها في مستندات المتطلبات، وكان في كتيبات الاستخدام.

فائدتها: 

- كل مطور جديد يفهم نفس التعريفات.
- الكلمة تستخدم دايماً معناها الرسمي فقط.
- تمنع اللبس بين اسم مستخدم واسم في الكود.

طب ليه بنسميم  ? Initial Analysis Objects

The identification of participating objects results in the initial analysis object model.

لما نحدد الحاجات اللي بتظهر في الـ use cases (زى Dispatcher ، Incident)،
يبقى كده بدأنا نكون نموذج كائنات تحليلي – Object Model

مثال تطبيقي: 

لو عندك اسمه ReportEmergency، هتلقي فيه عناصر بتتكرر:

Object	وظيفته في الـ Use Case
FieldOfficer	يبدأ البلاغ
Dispatcher	يرد عليه
Incident	نوع البلاغ اللي يتم تسجيله
EmergencyType	حريق؟ حادث؟ مواد خطرة؟

كل واحدة من دول تعتبر  Analysis Object

الجملة الختامية المهمة: 

The complete analysis model is usually not used as a means of communication between users and developers, as users are often unfamiliar with object-oriented concepts.

يعني المستخدم مش لازم يفهم UML والكائنات،
لكن محظوظ إن المطورين ببنوا المموج ده لأنفسهم على الأقل كبداية للنظام.

However, the description of the objects (i.e., the definitions of the terms in the glossary) and their attributes are visible to the users and reviewed.

لـكن المهم جداً إن المستخدم يشوف الـ glossary – يعرف كل اسم بيـشير لـيه.

إزاـي تحدد الـ analysis objects 

Many heuristics have been proposed in the literature for identifying objects.

يعني في قواعد ذكـة (Heuristics) بـتساعدك تحدد إـيه اللي يـعتبر object مـهم في الجدول في الصورة، أدـاك شـوية منهـم:

جدول Heuristics (قواعد لـاختيار الـ objects) 

القاعدة	معناها
Terms that developers or users must clarify	الكلـيات اللي بيـكون فيها لـبس لـازم تـحدد معـناها (زي Incident)
Recurring nouns in the use cases	أـي اسم (اسم كـيان) بيـتكرـر كـثير – غالـباً مـهم (زي Incident)
Real-world entities	الـكـيانـات الواقعـية اللي بـتحـثـلـها جـوهـنـظـامـ (Resource)
Real-world processes	أـي عمـليـة بـتحـصل فـي الواقع وـعـابـينـ نـثـثـلـها كـوـدـ (EmergencyOperationsPlan)
Use cases	لو اـسـم use case بيـتضـمنـ كـيانـ – غالـباً دـهـ object (ReportEmergency)
Data sources	زي الطـابـعـة أو مـلـفـ – بـرضـو مـكـنـ تكونـ (Printer)
Application domain terms	مـصـطـلـحـاتـ المـسـتـخـدـمـينـ أوـ الشـرـكـةـ – لو مـحـمـةـ نـخـفـظـ بـيـهاـ كـ object

بعد ما حـددـتـ الـ objectsـ، اـعملـ consolidation 

If two use cases refer to the same concept, the corresponding object should be the same.

يعـنيـ:

- لو فيه 2 use cases يـستـخدـمـوا نفسـ الحاجـةـ، لـازـمـ يـتـسمـواـ بـنفسـ الـاسمـ
- لو كانـ فيهـ اختـلافـ فـيـ التـسـمـيـةـ، نـخـتـارـ اـسـمـ وـاحـدـ موـحدـ وـنـوـضـخـ معـناـهـ

دهـ بـيـنـ اللـبـسـ بـيـنـ المـطـورـينـ وـبعـضـهـمـ 

مثالـ منـ الجـدولـ (Table 4-2) 

Table 4-2 Participating objects for the ReportEmergency use case.

Dispatcher	Police officer who manages Incidents. A Dispatcher opens, documents, and closes incidents in response to EmergencyReports and other communication with FieldOfficers. Dispatchers are identified by badge numbers.
EmergencyReport	Initial report about an Incident from a FieldOfficer to a Dispatcher. An EmergencyReport usually triggers the creation of an Incident by the Dispatcher. An EmergencyReport is composed of an emergency level, a type (fire, road accident, other), a location, and a description.
FieldOfficer	Police or fire officer on duty. A FieldOfficer can be allocated to at most one Incident at a time. FieldOfficers are identified by badge numbers.
Incident	Situation requiring attention from a FieldOfficer. An Incident may be reported in the system by a FieldOfficer or anybody else external to the system. An Incident is composed of a description, a response, a status (open, closed, documented), a location, and a number of FieldOfficers.

Object	التعريف
Dispatcher	مسؤول يبرد على بلاغات Incident، يكتب تقارير EmergencyReport
EmergencyReport	بلاغ طارئ يبدأ FieldOfficer، وبيؤدي لإنشاء Incident
FieldOfficer	موظف ميداني يعمل بلاغات، كل واحد له ID
Incident	موقع طارئ يستدعي تدخل من FieldOfficer – يتكون من: وصف + موقع + حالة + FieldOfficers

كل دول حدناهم ك ReportEmergency :Use Case في analysis objects 

:cross-check  بعد ما تحدد هم؟ اعمل

Once participating objects are identified and consolidated, the developers can use them as a checklist...

يعني بعد ما نطلعهم، نسأل نفسنا شوية أسئلة عشان نراجع الموجز.

جدول الأسئلة الذكية (من الـ Heuristics box) 

السؤال	الهدف منه
أي object يخلعوا الـ use cases ؟	نعرف مصدر البيانات
أي actors يقدروا يشوفوا البيانات دي ؟	نحدد الصالحيات
مين يقدر يعدل أو يمسح الـ object ؟	نراجع حالات التعديل والحذف
مين يقدر بيبدأ الـ use case ده ؟	نعرف الفاعلين
هل الـ object فعلاً مطلوب ؟	لو مفيش use case يعتمد عليه، يعني نحذفه



4.4.7 – Identifying Nonfunctional Requirements

Nonfunctional requirements describe aspects of the system that are not directly related to its functional behavior.

المتطلبات غير الوظيفية (nonfunctional) هي خصائص النظام اللي ملهاش علاقة مباشرة بالوظائف اللي بيقدمها، زي:

- الشكل وسهولة الاستخدام
- الأمان
- السرعة
- القابلية للصيانة

طيب ليه مهمة جدًا؟

Because they have as much impact on the development and cost of the system.

لأنها بتأثر بشكل كبير جدًا على:

- تكلفة المشروع
- قابلية النظام للاستخدام
- جودته فعليًا عند التنفيذ

مثال تطبيقي واضح من النص:

A mosaic display system displays data from radars to track planes...

لو بتبني نظام يعرض حركة الطيارات:

- فيه متطلبات غير وظيفية زي:
 - عدد الطيارات اللي يقدر يعرضهم
 - سرعة التحديث (refresh rate)
 - حجم الشاشة
 - دقة البيانات

كل ده مش "وظيفة" واضحة، لكنه مهم جدًا عشان النظام يستغل صح في الواقع.

 مثال ثانٍ:

In the SatWatch, one requirement says the time never needs to be reset, and another says the battery should last a long time.

في SatWatch :

- لو عايز الساعة ما تتأخرش أبداً → محتاج نظام دقيق (تكلفة أعلى)
 - لو عايزها تشتعل سنين بدون بطارية → محتاج تقليل العمليات
- الاتنين متطلبين غير وظيفيين لكنهم يتعارضوا. فلازم نحل التناقض.

 طيب نحدد المتطلبات غير الوظيفية إزاي؟

Checklists باستخدام

 جدول 3-4 فيه أسئلة ذكية تساعدك:

- إيه المتطلبات غير الوظيفية المهمة؟
- مين هيدير النظام؟
- النظام سريع ولا لأ؟
- هل لازم يكون في portable version؟
- مستوى الأمان قد إيه؟

 محتويات جدول 3-4 (بساطة):

الفئة (Category)	أمثلة للأسئلة (من الجدول)
Usability	مستوى المستخدم؟ محتاج تدريب؟ شكل الواجهة؟
Reliability	لو حصل خطأ، إيه اللي يحصل؟ في system recovery؟
Performance	السرعة؟ حجم البيانات؟ أقصى ضغط؟
Supportability	مين هيصين النظام؟ هل ممكن ننقله؟
Implementation	هل في قيود على الأجهزة؟ نظام التشغيل؟
Interface	بيتكامل مع أنظمة تانية؟ بيدعم formats معينة؟

Operation	مَنْ يَشْغُلُ النَّظَامَ؟
Packaging	مَنْ هِيَ زَوْلُ النَّظَامَ؟ مُحْتَاجٌ إِلَى جَمَارٍ مَعِينٍ؟
Legal	النَّظَامُ مُحْتَاجٌ إِلَى تَرْخِيصٍ؟ فِي حُقُوقِ نَسْرٍ؟

طيب بعد ما نحدد هم، نعمل بيهم إيه؟ 

They can organize them into refinement and dependency graphs to identify conflicts.

يعني بنصنفهم ونشوف:

- هل في تعارض بين المتطلبات؟
- هل في متطلبات لازم تتنفذ قبل غيرها؟
- هل في متطلبات إضافية ظهرت واحتاجنا براجعة use cases؟

Section 4.5 – Managing Requirements Elicitation

4.5.1 – Negotiating Specifications with Clients: Joint Application Design (JAD)

- ◆ "Joint Application Design (JAD) is a requirements method developed at IBM at the end of the 1970s."

تصميم التطبيقات المشتركة (JAD) هو أسلوب لتجمیع المتطلبات تم تطويره في شركة IBM في أواخر السبعينيات.

يعني دي طريقة ابتكرتها IBM عشان تسهل الاتفاق بين المطوريين والعملاء والمستخدمين حوالين المتطلبات اللي النظام يحتاجها. 

- ◆ "Its effectiveness lies in that the requirements elicitation work is done in one single workshop session in which all stakeholders participate."

فعالية هذه الطريقة تكمن في أن كل عمل تجمیع المتطلبات يتم في جلسة ورشة واحدة يشارک فيها جميع أصحاب المصلحة.

يعني بدل ما كل واحد يستغل لوحده، كل الناس اللي ليهم علاقة بالمشروع يتقعدوا سوا في جلسة واحدة عشان يحددوا كل المتطلبات مع بعض. 

- ◆ "Users, clients, developers, and a trained session leader sit together in one room to present their viewpoints, listen to other viewpoints, negotiate, and come to a mutually acceptable solution."

المستخدمون، والعملاء، والمطوروون، وقائد للجلسة مدرب، يجلسون سوياً في غرفة واحدة ليعرضوا وجهات نظرهم، ويستمعوا لوجهات نظر الآخرين، ويتفاوضوا، ويتوصلوا إلى حل مقبول من الجميع.

الجلسة دي بتجمع كل الأطراف: كل واحد يقول رأيه، يسمع غيره، والآخر يطلعوا بحل متفق عليه.

- ◆ "*The outcome of the workshop, the final JAD document, is a complete requirements specification document that includes definitions of data elements, workflows, and interface screens.*"

نتيجة ورشة العمل هي مستند JAD النهائي، وهو وثيقة متكاملة لمواصفات المطلبات تحتوي على تعاريفات لعناصر البيانات، وتدفق العمل، وشاشات الواجهة.

يعني بعد الجلسة، يطلعوا بمستند رسمي مكتوب فيه كل حاجة: البيانات المطلوبة، خطوات النظام، وتصميمات الشاشات.

- ◆ "*Because the final document is jointly developed by the stakeholders (that is, the participants who not only have an interest in the success of the project, but also can make substantial decisions), the final JAD document represents an agreement among users, clients, and developers...*"

لأن المستند النهائي يتم تطويره بشكل مشترك بواسطة أصحاب المصلحة (يعني المشاركين اللي مش بس بيهم نجاح المشروع، لكن كان عندهم صلاحية اتخاذ قرارات مهمة)، فإن مستند JAD النهائي يمثل اتفاقاً بين المستخدمين، والعملاء، والمطورين...

يعني المستند اللي يطلع من الجلسة مش جاي من جهة واحدة، لكن معنول بموافقة كل الناس اللي القرار بإيديهم فعلاً، علشان كده يكون رسمي ومعقد.

- ◆ "... and thus minimizes requirements changes later in the development process."

... وبالتالي يقلل التغييرات في المطلبات في المراحل المتأخرة من تطوير النظام.

لأنهم اتفقوا من الأول، مفيش حد هيجي بعد شهر يقول "لا، غيرنا رأينا". فده يقلل التعديلات المتأخرة اللي بتبوظ المشاريع.

✓ Figure 4-15: Activities of JAD (UML Activity Diagram)

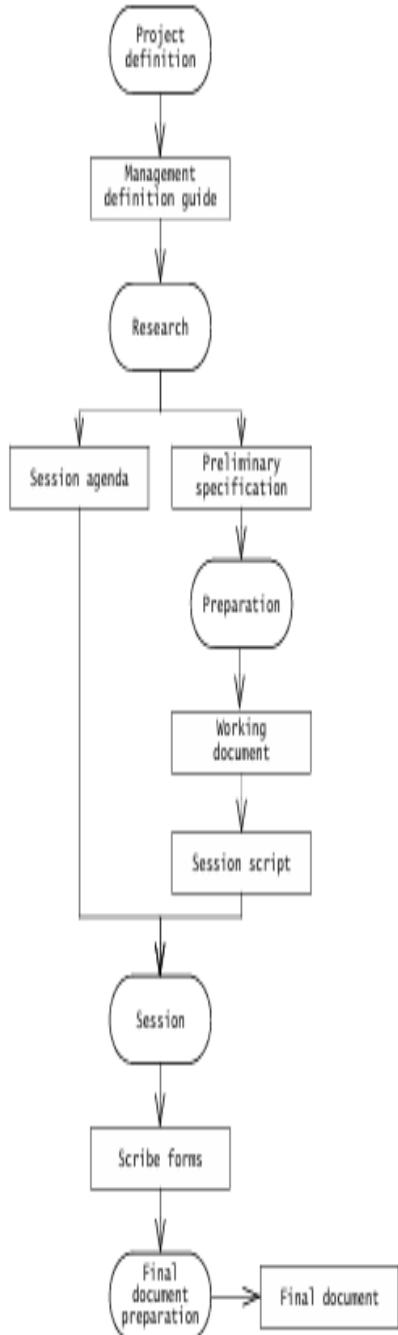
الرسمة دي بتوضح خطوات تنفيذ جلسة JAD من أول ما فكر في المشروع لحد ما نخرج بوثيقة نهائية رسمية.

وهيشي فيها واحد واحد

- ◆ Project Definition

تعريف المشروع

في الخطوة دي، يحدد هدف المشروع ومشكلته الرئيسية.
مدير الجلسة (facilitator) يعقد مع العميل ويبدأ يسأل:
"إحنا بنعمل النظام ده ليه؟ وإيه اللي محتاجينه يشتغل؟"



◆ Management Definition Guide

دليل تعريف الإدارة

ده مستند بيتعمل بعد المقابلات المبدئية، بيحظ فيه كل المعلومات الإدارية المهمة اللي تم الاتفاق عليها (زي الأهداف العامة – الأولويات – حدود النظام). يعني يعتبر الأساس اللي هنبدأ منه باقي الجلسة.

◆ Research

البحث والتحضير

هنا مدير الجلسة يجمع معلومات من المستخدمين – يشوف الأنظمة القديمة – يقرأ تقارير – يفهم البيئة اللي هيشتغل فيها النظام الجديد.

◆ Session Agenda

جدول الجلسة

بنحط جدول واضح للجلسة الأساسية:
إيه المواضيع اللي هنتكلم فيها؟
مين هيعرض إيه؟
وايه وقت كل جزء؟

◆ Preliminary Specification

المواصفات الأولية

بنبدأ نحط تصور مبدئي للمتطلبات اللي جمعناها لحد دلوقتي.
مش نهائي، بس هيبقى مرجع نبدأ منه المناقشة الفعلية.

◆ Preparation

التحضير

بنجهز أوراق الجلسة، العروض، المسودات، تصميم الشاشات الأولية، إلخ.
يعني كل حاجة نحتاجها عشان الجلسة تمشي سريعة ومن غير عشوائية.

◆ Working Document

Figure 4-15 Activities of JAD (UML activity diagram). The heart of JAD is the Session activity during which all stakeholders design and agree to a requirements specification. The activities prior to the Session maximize its efficiency. The production of the final document captures the decisions made during the Session.

المستند التشغيلي

ده عبارة عن المسودة الأساسية اللي هنشتغل فيها في الجلسة، فيها المتطلبات اللي جمعناها من البحث، واللي هنناقشها ونعدل عليها خلال الجلسة.

◆ Session Script

نص الجلسة

ده زي السيناريو أو السكريبت بتاع الجلسة – بيقول كل خطوة مين هيقدّمها، ومين يتكلم إمّي، واللي بعده هيكون مين... وهكذا. عشان الجلسة تمّشي منظمة.

◆ Session

الجلسة الرئيسية

أهم خطوة في JAD! كل الأطراف (المطور – العميل – المستخدم – المدير) يقعدوا مع بعض 3-5 أيام، يناقشو المتطلبات، يعدلوا، يتتفقوا، وكل القرارات بتتوافق.

◆ Scribe Forms

نماذج كاتب الجلسة

"Scribe" = كاتب الجلسة ده شخص بيكتب كل كلمة واتفاق حصل في الجلسة على نماذج جاهزة، عشان يتجمع منها التقرير النهائي.

◆ Final Document Preparation

تحضير الوثيقة النهائية

مدير الجلسة بيجمع كل اللي انتقال في الجلسة، ويكتبه في شكل وثيقة رسمية جاهزة للتتوقيع.

◆ Final Document

الوثيقة النهائية

ال المستند النهائي اللي اتفق عليه الكل -
هو اللي بنبني عليه النظام -
ويبقى فيه كل المتطلبات الرسمية.

- ◆ "JAD has been used by IBM and other companies."

تم استخدام JAD بواسطة شركة IBM وشركات أخرى.

يعني الطريقة دي مش نظرية، دي اتنفيذت فعلياً في شركات حقيقة وناجحة، مش مجرد كلام أكاديمي.

- ◆ "JAD leverages group dynamics to improve communication among participants and to accelerate consensus."

يعتمد JAD على ديناميكية المجموعة لتحسين التواصل بين المشاركين وتسرير الوصول إلى اتفاق.

يعني لأن كل الناس قاعدين مع بعض، بيبقى فيه تفاعل حي، فيفهموا بعض أسرع ويوصلوا لاتفاق بسهولة بدل من تبادل إيميلات أو مستندات.

- ◆ "At the end of a JAD session, developers are more knowledgeable of user needs, and users are more knowledgeable of development trade-offs."

في نهاية جلسة JAD ، يكون المطورون أكثر فهماً لاحتياجات المستخدم، والمستخدمون أكثر فهماً للتنازلات التقنية الخاصة بالتطوير.

المجملة دي مجمة جداً:

- المطور يفهم العميل عايز إيه فعلياً
- والعميل يفهم إيه اللي صعب أو غالٍ أو محتاج وقت فالاتنين بيحصل بينهم "تفاهم واقعي".

- ◆ "Additional gains result from a reduction of redesign activities downstream."

تكون هناك فوائد إضافية من تقليل أنشطة إعادة التصميم لاحقاً في المشروع.

يعني لما نوضح المتطلبات من الأول، مش هنضطر نرجع نعدل في الشغل بعد ما نكون عملناه غلط – وده بيوفر وقت ومجهد كتير.

- ◆ "Because of its reliance on social dynamics, the success of a JAD session often depends on the qualifications of the JAD facilitator as a meeting facilitator."

نظرًا لاعتماد JAD على التفاعل الاجتماعي، فإن نجاح الجلسة غالباً ما يعتمد على كفاءة مدير الجلسة كمنسق للاجتماع.

 يعني الشخص اللي بيدير الجلسة (facilitator) لازم يكون عنده شخصية قوية، يعرف يتحكم في الحوار، يوقف الجدال، يخليل الكل يشارك، ويوصل للاتفاق – لأنه لو فشل، الجلسة كلها تضيع.

4.5.2 – Maintaining Traceability

- ◆ *"Traceability is the ability to follow the life of a requirement."*

إمكانية التتبع (Traceability) هي القدرة على تتبع حياة المطلب.

 يعني تعرف المطلب بدأ منين، مين طلبه، اتحط فين في التصميم، اتنفذ في أي جزء من الكود، وتحتبره إزاي.

- ◆ *"This includes tracing to where the requirements came from (e.g., who originated it, which client need does it address)..."*

ده يشمل تتبع مصدر المطلبات (زي: مين اللي طلبه، واحتياج أي عميل بيغطيه)...

 بنسجل:

- العميل اللي طلب الميزة دي
- الغرض منها
- ومنين بدأت فكرتها

- ◆ *"... to which aspect of the system and the project it affects (e.g., which components realize the requirement, which test case checks its realization)."*

...ولأي جزء في النظام أو المشروع المطلب ده بيأثر (مثلاً: أي مكون ينفذ المطلب، أو أي اختبار بيتحقق منه).

 لازم نعرف:

- المطلب ده مرتبط بأي زر؟ أى شاشة؟ أى كود؟
- واتعمله test case فين؟
- لو اتغير، إيه اللي لازم يتغير معاه؟

- ◆ *"Traceability enables developers to show that the system is complete, testers to show that the system complies with its requirements, designers to record the rationale behind the system, and maintainers to assess the impact of change."*

التبغ يتيح للمطوريين إثبات أن النظام مكتمل، ولختري النظام إثبات توافقه مع المتطلبات، وللمصممين تسجيل منطق التصميم، وللمعنيين بالصيانة تقييم تأثير أي تغيرات.

يعني traceability يقييد كل الناس:

- المطور يعرف هو غطي كل المتطلبات ولا لا
- الختير يتتأكد من تحقق كل حاجة
- المصمم يشرح ليه اختار الطريقة دي
- المسؤول عن التعديل يقدر يشوف لو غير حاجة إيه اللي هيتأثر

◆ "Consider the SatWatch system we introduced at the beginning of the chapter."

خلينا نرجع لمثال SatWatch اللي بدأنا به الفصل.

النظام اللي بيعرض الوقت والتاريخ تلقائياً.

◆ "Currently, the specification calls for a two-line display that includes time and date."

حالياً، المواصفات بتقول إن الشاشة هتعرض سطرين: الوقت والتاريخ.

التصميم الحالي: التاريخ في سطر، والوقت في سطر ثاني.

◆ "After the client decides that the digit size is too small for comfortable reading, developers change the display requirement to a single-line display combined with a button to switch between time and date."

بعد ما العميل قرر إن حجم الأرقام صغير وصعب قراءته، المطوريين غيروا المتطلب ليعرض سطر واحد مع زر للتبديل بين الوقت والتاريخ.

العميل قال: "عايز الأرقام تكبر".

فالمطوريين قرروا يشيلوا سطر وينخلوه سطر واحد بس، وبزرار تغيير بين التاريخ والساعة.

◆ "Traceability would enable us to answer the following questions:"

التبغ هيساعدنا نجاوب على الأسئلة دي:

◆ "Who originated the two-line display requirement?"

مين اللي اقترح عرض الوقت والتاريخ في سطرين؟

نعرف أصل المطلب ده جاي منين.

◆ "Did any implicit constraints mandate this requirement?"

هل في قيود ضمنية فرضت المطلب ده؟ (زي حجم الشاشة، البطارية، التكلفة؟)

ممكن يكون السطرين كانوا عشان قيود تانية - فلازم نعرفها.

◆ "Which components must be changed because of the additional button and display?"

إيه الأجزاء اللي لازم تتعدل لما ضفنا زر جديد وعرض مختلف؟

هحتاج نغير الواجهة؟ ولا كان المعالج الداخلي؟ أو كود التحكم في العرض؟

◆ "Which test cases must be changed?"

إيه سيناريوهات الاختبار اللي لازم تتغير بعد التعديل ده؟

الاختبارات القديمة كانت على سطرين، دلوقتي لازم نراجعها.

◆ "The simplest approach to maintaining traceability is to use cross-references among documents, models, and code artifacts."

أسهل طريقة للحفاظ على التتبع هي استخدام روابط مرجعية بين المستندات، النماذج، والكود.

يعني مثلاً في مستند المطلبات نكتب: "ده بيتحقق في شاشة X وفي كود الملف Y"

◆ "Each individual element (e.g., requirement, component, class, operation, test case) is identified by a unique number."

كل عنصر في النظام (زي المطلبات أو الـ components أو الـ test cases) يأخذ رقم مميز.

الأرقام دي بتسهل التتبع، ونقدر نربط المطلب بالملف بالكود بالاختبار.

◆ "This approach is expensive in time and personpower, and it is error prone."

الطريقة دي بتاخد وقت ومجهد كبير، وسهل يحصل فيها أخطاء.

لو هتعمل ده يدوياً في Word أو Excel ، فده متعب جداً ويمكن تتلغبط أو تنسى حاجة.

◆ "Tool support can ease the burden."

البرمجيات المتخصصة ممكن تخفف العبء ده.

زي أدوات traceability (مثلاً: IBM DOORS)، تقدر تعمل كل ده تلقائياً.

- ◆ "Commercial tools like DOORS facilitate traceability dependencies at a more detailed level..."

أدوات تجارية زي DOORS بتساعد على إدارة العلاقات التبعية بمستوى تفصيلي أكبر...

يعني تقدر تشوف بدقة:

- المتطلب ده اتنفذ فين؟
- مرتبط بأي زر؟
- متغطي بأي test؟

- ◆ "...This reduces the cost of maintaining traceability, though not eliminating it."

ده يقلل تكلفة تتبع المتطلبات، لكنه ما ييشيلهاش تماماً.

يعني حتى بالأدوات، لازم تتبع، لكن بتتوفر وقت كبير عن الطريقة اليدوية.

✓ Section 4.5.3 – Documenting Requirements Elicitation

- ◆ "The results of the requirements elicitation and the analysis activities are documented in the Requirements Analysis Document (RAD)."

نتائج تجميع المتطلبات وأنشطة التحليل يتم توثيقها في مستند تحليل المتطلبات (RAD)

يعني كل اللي عملناه في جلسات JAD وجمع المتطلبات بنكتبه رسميًا في ملف اسمه RAD – ده هو المرجع الأساسي للنظام.

- ◆ "This document completely describes the system in terms of functional and nonfunctional requirements."

هذا المستند يصف النظام بالكامل من حيث المتطلبات الوظيفية وغير الوظيفية.

يعني إن RAD يعطي كل حاجة النظام بعملها (functional) + إزاي بيعملها (nonfunctional)

- ◆ "The audience for the RAD includes the client, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design)."

الجمهور المقصود من RAD يشمل العميل، المستخدمين، إدارة المشروع، محللي النظام (الذين يشتغلوا في جمع المتطلبات)، ومصممي النظام (الذين يصممون النظام نفسه).

يعني المستند ده لازم يكون واضح ومقرئه لأي حد له علاقة بالمشروع سواء بيشتغل في المتطلبات أو التصميم أو حتى الإدارة.

- ◆ "*The first part of the document, including use cases and nonfunctional requirements, is written during requirements elicitation.*"

الجزء الأول من المستند، اللي فيه Use Cases والمتطلبات غير الوظيفية، بيكتب أثناء مرحلة جمع المتطلبات.

يعني ده بيكتب من أول الجلسات، بناءً على اللي اتقال واتفقنا عليه.

- ◆ "*The formalization of the specification in terms of object models is written during analysis.*"

أما الجزء الرسمي اللي فيه النماذج (object models) بيكتب لاحظاً خلال مرحلة التحليل.

يعني في البداية بنكتب الكلام المتفق عليه، بعدين في مرحلة التحليل نحوله لنماذج تصميم UML.

- ◆ "*Figure 4-16 is an example template for a RAD used in this book.*"

الشكل 4-16 هو نموذج ل قالب RAD المستخدم في هذا الكتاب.

فيه كل الفصول والعناوين اللي RAD المفروض يحتويها.

شرح مكونات RAD من الشكل 4-16 : (Requirements Analysis Document) ✓

- ◆ 1. Introduction

يشمل:

- 1.1 Purpose
- 1.2 Scope
- 1.3 Objectives and success criteria
- 1.4 Definitions and acronyms
- 1.5 References
- 1.6 Overview

الغرض - نطاق المشروع - المصطلحات - مراجع قديمة - لغة عامة.

◆ 2. Current System

"The second section, *Current system*, describes the current state of affairs."

القسم الثاني يوصف النظام الحالي.

لو هنستبدل نظام قديم، بنووصف دلوقتي الناس شغالين إزاي، والمشاكل في الوضع الحالي.

◆ 3. Proposed System

"The third section, *Proposed system*, documents the requirements elicitation and the analysis model of the new system."

القسم الثالث يوثق المتطلبات والمفهوم التحليلي للنظام الجديد.

فيه كل اللي اتجمع من جلسات JAD + التحليل التقني اللي عملناه بعد كده.

3.1 Overview

شرح عام عن النظام الجديد

3.2 Functional Requirements

المتطلبات اللي بتحدد إيه النظام بيعمله

3.3 Nonfunctional Requirements

المتطلبات اللي بتحدد إزاي النظام بيشتغل

وفيها:

- 3.3.1 Usability
- 3.3.2 Reliability
- 3.3.3 Performance
- كل اللي شفناه في (FURPS+) ...

3.4 System Models

- 3.4.1 Scenarios
- 3.4.2 Use Case Model
- 3.4.3 Object Model (*during analysis*)

- 3.4.4 Dynamic Model (*during analysis*)
- 3.4.5 UI Mockups

فِيهَا الـ (UML) diagrams والشاشات المقترحة + النماذج اللي بتشرح سلوك النظام.

◆ 4. Glossary

قاموس المصطلحات – ضروري جدًا لمنع اللبس في الكلمات والتعریفات.

◆ "*The RAD should be written after the use case model is stable...*"

المفروض نكتب RAD بعد ما تكون خلصنا من ال Use Cases وما فيش تعديلات كتير بتحصل عليهم.

علشان المستند النهائي يكون دقيق، ومهكونش لسه بيتغير كل شوية.

◆ "*The RAD, once published, is baselined and put under configuration management.*"

بمجرد نشر RAD ، يتم اعتباره نسخة أساسية ويخضع لإدارة التكوين.

يعني بيتحفظ كنسخة رسمية، وأي تعديل عليه بعد كده لازم يكون مسجل ومعقد.

◆ "*The revision history section of the RAD will provide a history of changes...*"

قسم سجل التعديلات في RAD يعرض تاريخ التغييرات، من اللي عمل التعديل، وتاريخ التعديل، ووصف مختصر للتغيير.

زي log book – نعرف مين غير إيه وامتنى وليه.

