Software Safety

Requirements Engineering

Prof. Dr.-Ing. Patrick Mäder, MSc. Martin Rabe

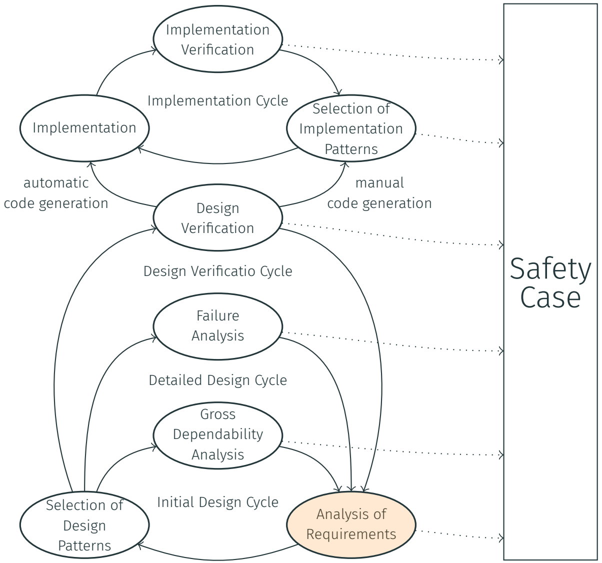
Copyright

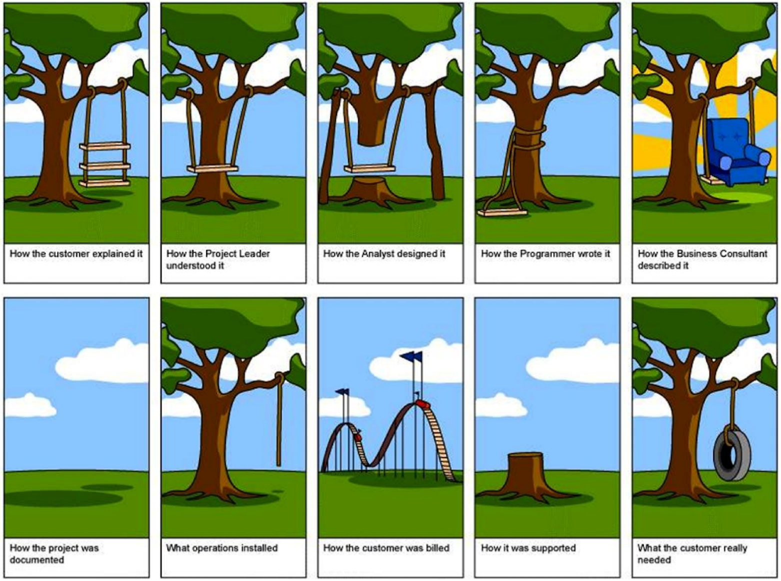
- This lecture is partly based on a book by Wiegers and Beatty [WiBe13] and partly inspired by RE/M courses by Lawrence Chung and Bernd Westphal.

Contents

Software Requirements

Software Lifecycle




How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Software Requirement

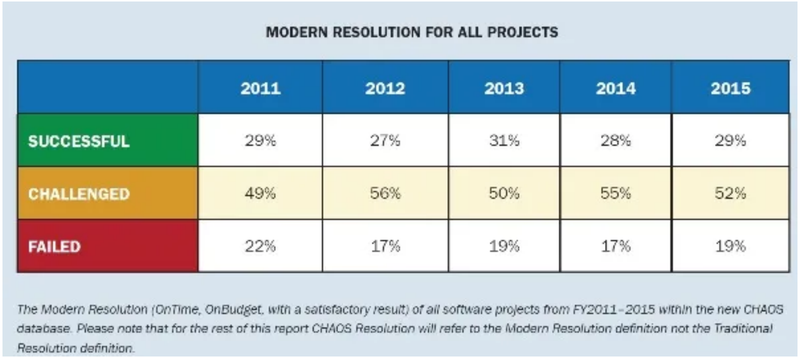
Software Requirement [IEEE 610.12:1990]

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
3. A documented representation of a condition or capability as in (1) or (2).

How big is the “erroneous specification” problem?

- Bell Labs and IBM studies
    - 80% of all defects are inserted in the requirements phase → improving RQs definition process reduces amount of testing and rework
- Voyager and Galileo spacecraft
    - of the 197 significant software faults found during integration and testing only 3 were programming errors → **vast majority were requirements problems** [Lutz93]

Software Project Success and Failure



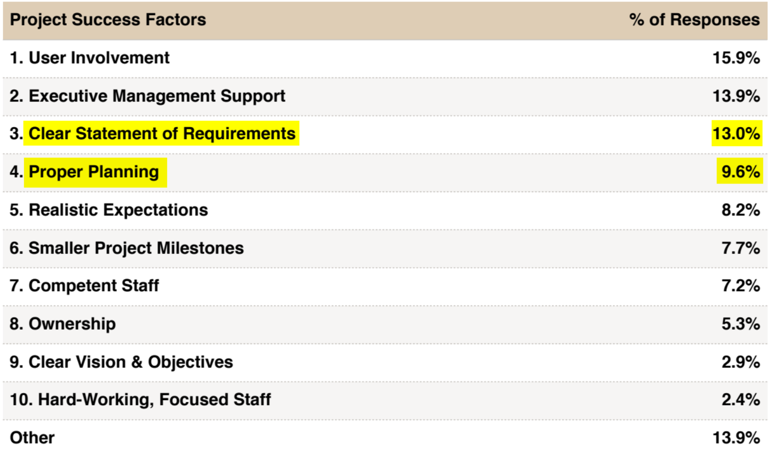
MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

[Stéphane Wojewoda and Shane Hastie: **Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch**, 2015]

Software Project Success Factors



Project Success Factors	% of Responses
1. User Involvement	15.9%
2. Executive Management Support	13.9%
3. Clear Statement of Requirements	13.0%
4. Proper Planning	9.6%
5. Realistic Expectations	8.2%
6. Smaller Project Milestones	7.7%
7. Competent Staff	7.2%
8. Ownership	5.3%
9. Clear Vision & Objectives	2.9%
10. Hard-Working, Focused Staff	2.4%
Other	13.9%

[ATTRECTO team: The truth about the accuracy of software estimations, 2018]

Safety Requirements

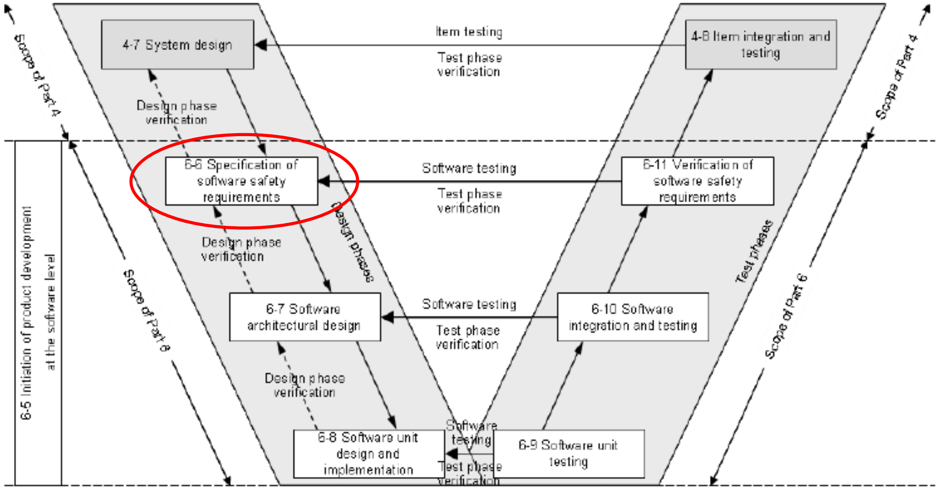
Safety Requirements (1/2)

- **Safety requirements** are requirements which ensure an adequate safety of a system
- For critical systems usually a separate **safety requirements document** sets out the safety requirements of the system.
- Analogous to requirements in general, **safety requirements can be grouped as**:
  - functional safety requirements
  - non-functional safety requirements
  - context of operation

Safety Requirements (2/2)

- **Functional safety requirements**: services, to be carried out by the system in order to guarantee safety.
  - Example: if the heat in the reactor of a nuclear power plant exceeds a certain temperature, an automatic shut down mechanism has to be activated.
- **Non-functional safety requirements**: other requirements (e.g., reliability, availability).
- **Context of operation**: describe the context, under which a system is supposed to operate (e.g., environment, administrative procedures).
  - Example: requirement that the outside temperature should not exceed 50 °C. $\rightarrow$ Operating the unit in a desert is not appropriate.

Software Safety Requirements [ISO26262]: Development Process Perspective



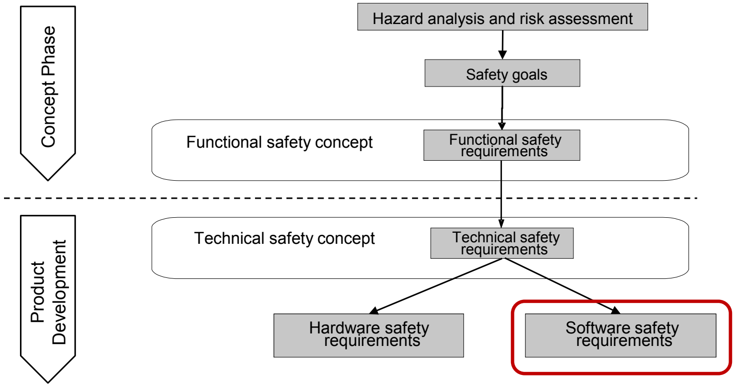
4-7 System design
Item testing
Test phase verification
4-8 Item integration and testing
Scope of Part 4
Design phase verification
6-6 Specification of software safety requirements
Software testing
Test phase verification
6-11 Verification of software safety requirements
Design phases
Test phases
Design phase verification
6-7 Software architectural design
Software testing
Test phase verification
6-10 Software integration and testing
Design phase verification
6-8 Software unit design and implementation
Software testing
Test phase verification
6-9 Software unit testing
6-5 Initiation of product development at the software level
Scope of Part 6
Scope of Part 4
Scope of Part 6

[ISO26262]

Software Safety Requirements [ISO26262]: Objectives

- Derive **software safety requirements specification** from technical safety concept and system design
- Address each software-based function whose failure could lead to a violation of a technical safety requirement allocated to software
- Comply with requirements for ’Specification and management of safety requirements’ [ISO 26262-8:2011, Clause 6]
- Verify that software safety requirements and hardware–software interface requirements are consistent with technical safety concept and system design specification

Software Safety Requirements [ISO26262]: Safety Analysis Perspective


Concept Phase
Hazard analysis and risk assessment
Safety goals
Functional safety concept
Functional safety requirements
Product Development
Technical safety concept
Technical safety requirements
Hardware safety requirements
Software safety requirements

- Technical safety requirements have been allocated to hardware and software
- Software safety requirements are further detailed considering hardware constraints hardware and their impact on software

Software Safety Requirements [ISO26262]: Coverage

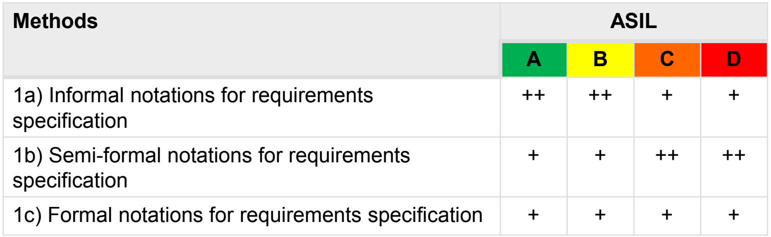
- Specified system and hardware configurations
- Hardware-software interface (HIS) specification
- Relevant requirements of hardware design specification
- Timing constraints
- External interfaces
- Operating modes of vehicle, system, or hardware having an impact on software
- ASIL decomposition can be applied [ISO 26262-9:2011, Clause 5]
- Refine HSI specification sufficiently

Software Safety Requirements [ISO26262]: Procedure

- Specify or reference other functions without safety requirements that are carried out by the embedded software as well
- Plan verification of software requirements specification and refined HSI
- Verify software requirements specification and refined HSI

Software Safety Requirements [ISO26262]: Specification

- Use appropriate combination of ...
  - natural language
  - specification notations listed below



Methods	ASIL			
	A	B	C	D
1a) Informal notations for requirements specification	++	++	+	+
1b) Semi-formal notations for requirements specification	+	+	++	++
1c) Formal notations for requirements specification	+	+	+	+

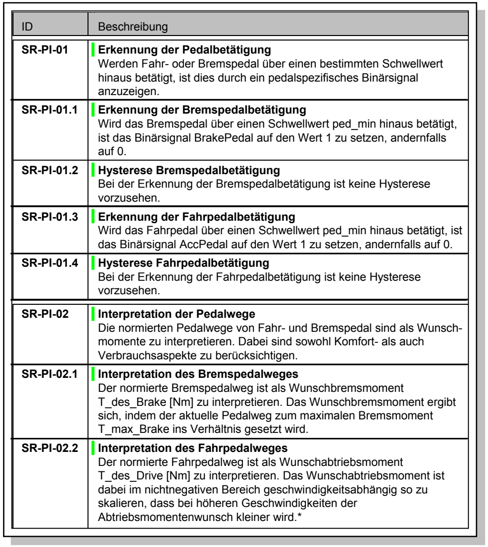
For lower level safety requirements (e.g. software and hardware safety requirements) notations listed in the table are more appropriate.

Source(s): ISO 26262-8:2011

Specification Example: Adaptive Cruise Control (ACC)

❑ **ACC - Pedal Interpretation**

Das Subsystem „PedalInterpretation“ interpretiert die aktuellen, normierten Pedalwege von Fahr- und Bremspedal unter Verwendung von Kennlinien und der aktuellen Geschwindigkeit als Wunschmomente.
Darüber hinaus werden zwei Flags generiert, die anzeigen, ob die Pedale als gedrückt angesehen werden oder nicht.



ID	Beschreibung
SR-PI-01	**Erkennung der Pedalbetätigung** Werden Fahr- oder Bremspedal über einen bestimmten Schwellwert hinaus betätigt, ist dies durch ein pedalspezifisches Binärsignal anzuzeigen.
SR-PI-01.1	**Erkennung der Bremspedalbetätigung** Wird das Bremspedal über einen Schwellwert ped_min hinaus betätigt, ist das Binärsignal BrakePedal auf den Wert 1 zu setzen, andernfalls auf 0.
SR-PI-01.2	**Hysterese Bremspedalbetätigung** Bei der Erkennung der Bremspedalbetätigung ist keine Hysterese vorzusehen.
SR-PI-01.3	**Erkennung der Fahrpedalbetätigung** Wird das Fahrpedal über einen Schwellwert ped_min hinaus betätigt, ist das Binärsignal AccPedal auf den Wert 1 zu setzen, andernfalls auf 0.
SR-PI-01.4	**Hysterese Fahrpedalbetätigung** Bei der Erkennung der Fahrpedalbetätigung ist keine Hysterese vorzusehen.
SR-PI-02	**Interpretation der Pedalwege** Die normierten Pedalwege von Fahr- und Bremspedal sind als Wunschmomente zu interpretieren. Dabei sind sowohl Komfort- als auch Verbrauchsaspekte zu berücksichtigen.
SR-PI-02.1	**Interpretation des Bremspedalweges** Der normierte Bremspedalweg ist als Wunschbremsmoment T_des_Brake [Nm] zu interpretieren. Das Wunschbremsmoment ergibt sich, indem der aktuelle Pedalweg zum maximalen Bremsmoment T_max_Brake ins Verhältnis gesetzt wird.
SR-PI-02.2	**Interpretation des Fahrpedalweges** Der normierte Fahrpedalweg ist als Wunschabtriebsmoment T_des_Drive [Nm] zu interpretieren. Das Wunschabtriebsmoment ist dabei im nichtnegativen Bereich geschwindigkeitsabhängig so zu skalieren, dass bei höheren Geschwindigkeiten der Abtriebsmomentenwunsch kleiner wird.*

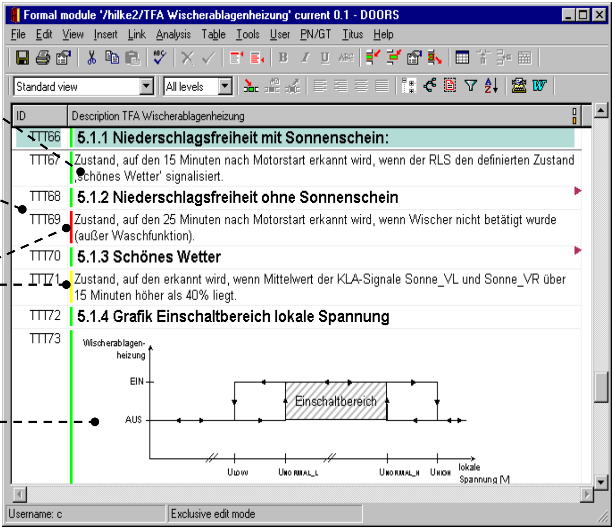
Tool Example: IBM Rational DOORS

Requirement

Unique Identification

Change Report Compared to the Last Freeze

Integration of Graphics (OLE-Objects etc.)



Formal module '/hilke2/TFA Wischerablagenheizung' current 0.1 - DOORS

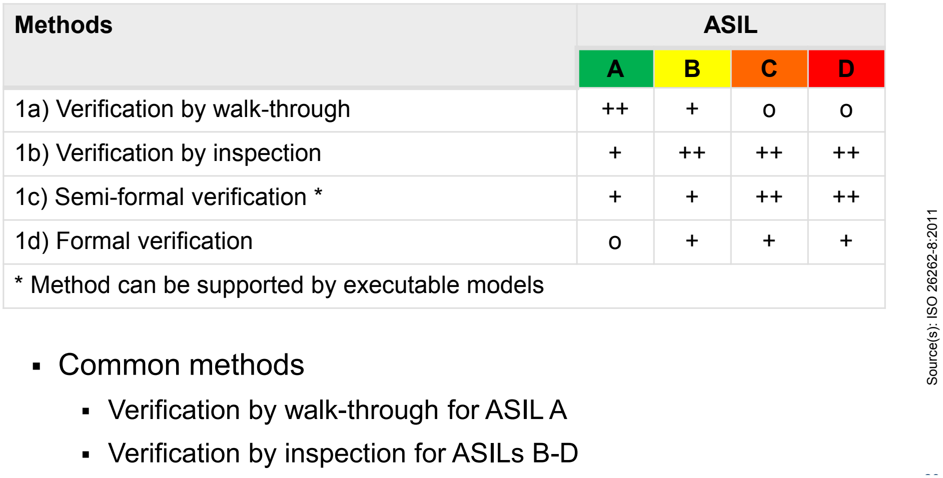
File Edit View Insert Link Analysis Table Tools User PN/GT Titus Help

Standard view | All levels

ID	Description TFA Wischerablagenheizung
TTT66	**5.1.1 Niederschlagsfreiheit mit Sonnenschein:**
TTT67	Zustand, auf den 15 Minuten nach Motorstart erkannt wird, wenn der RLS den definierten Zustand ‚schönes Wetter' signalisiert.
TTT68	**5.1.2 Niederschlagsfreiheit ohne Sonnenschein**
TTT69	Zustand, auf den 25 Minuten nach Motorstart erkannt wird, wenn Wischer nicht betätigt wurde (außer Waschfunktion).
TTT70	**5.1.3 Schönes Wetter**
TTT71	Zustand, auf den erkannt wird, wenn Mittelwert der KLA-Signale Sonne_VL und Sonne_VR über 15 Minuten höher als 40% liegt.
TTT72	**5.1.4 Grafik Einschaltbereich lokale Spannung**
TTT73	

Username: c | Exclusive edit mode

Verification of software safety requirements



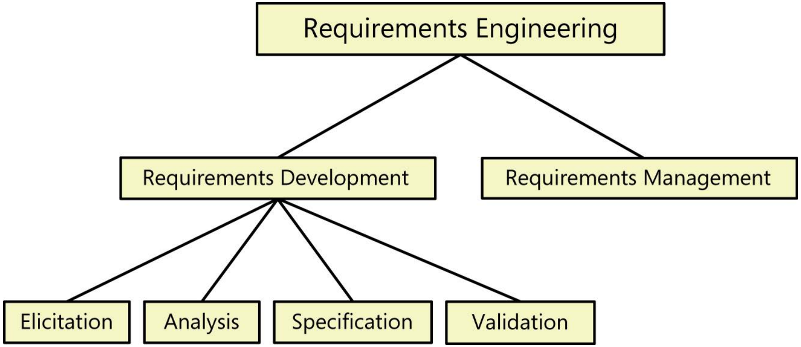
Methods	ASIL			
	A	B	C	D
1a) Verification by walk-through	++	+	o	o
1b) Verification by inspection	+	++	++	++
1c) Semi-formal verification *	+	+	++	++
1d) Formal verification	o	+	+	+
* Method can be supported by executable models				

- Common methods
  - Verification by walk-through for ASIL A
  - Verification by inspection for ASILs B-D

Source(s): ISO 26262-8:2011

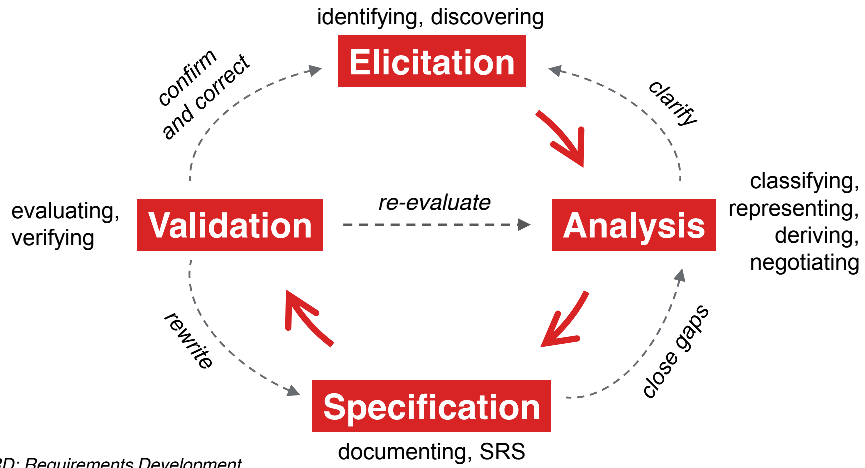
Requirements Engineering

Requirements Development and Management



[Wiegers and Beatty 2013]

Requirements Development Process



RD: Requirements Development
SRS: Software Requirements Specification

Requirements Elicitation

Step 1: Elicitation

Elicitation – all activities for discovering requirements, e.g., interviews, workshops, document analysis, prototyping.

Key actions:

- Identifying the product’s expected user classes and other stakeholders.
- Understanding user tasks and goals and the business objectives with which those tasks align.
- Learning about the environment in which the new product will be used.
- Working with individuals who represent each user class to understand their functionality needs and their quality expectations.

Stakeholders

Persons or organizations who...

- have a valid interest in the system
- are affected by the system

For example ...

- anyone who operates the system (normal and maintenance operators)
- anyone who benefits from the system
  (functional, political, financial and social beneficiaries)
- anyone involved in purchasing or procuring the system
- organizations which regulate aspects of the system
  (financial, safety, and other regulators)
- organizations responsible for systems interfacing with the system under design
- people or organizations opposed to the system (negative stakeholders)

Elicit Requirements

- **Interviews** are the best way to elicit requirements
- Explore requirements **systematically**
- Sounds simple – but is the hardest part!

Elicitation is Often Challenging

- **Problems of scope**

  What is the **boundary** of the system? • What **details** are actually required?

- **Problems of understanding** Users do not know what they want • don't know what is needed • have a poor understanding of their computing environment • don't have a full understanding of their domain • omit "obvious" stuff • are ambiguous

- **Problems of volatility**

  Requirements change over time

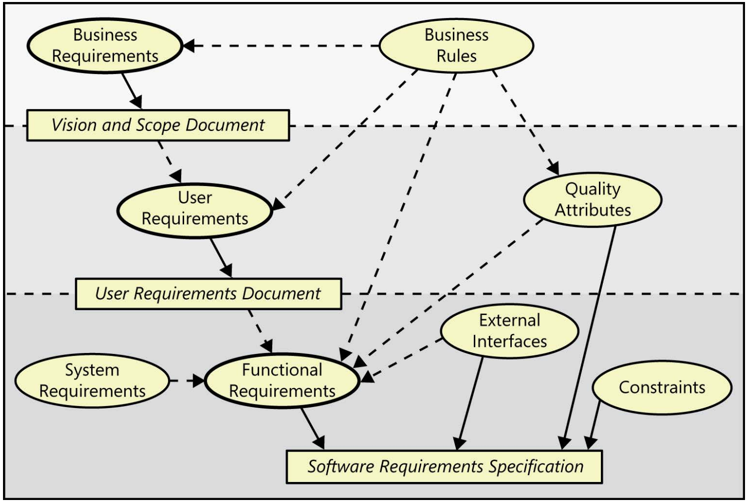
I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?
E-mail: SCOTTADAMS@AOL.COM
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
www.dilbert.com

Fundamental Requirements Types

- **Functional requirements** – an action the product must take to be useful
  - Example: *The product shall allow to track individual payments of coffee servings.*
- **Non-functional requirements (NFR)** – a property or quality the product must have
  - Example: *The product shall be accessible in multiple languages (such as German and English)*
- **Constraints** – global requirements on the project or the product
  - Example: *The product shall be available before March 1st.*

[Franke Flair]

Differentiated View on Requirements Types (1/2)



[Wiegers and Beatty 2013]

Differentiated View on Requirements Types (2/2)

Term	Definition
Business requirement	A high-level business objective of the organization that builds a product or of a customer who procures it.
Business rule	A policy, guideline, standard, or regulation that defines or constrains some aspect of the business. Not a software requirement in itself, but the origin of several types of software requirements.
Constraint	A restriction that is imposed on the choices available to the developer for the design and construction of a product.
External interface requirement	A description of a connection between a software system and a user, another software system, or a hardware device.
Feature	One or more logically related system capabilities that provide value to a user and are described by a set of functional requirements.
Functional requirement	A description of a behavior that a system will exhibit under specific conditions.
Non-functional requirement	A description of a property or characteristic that a system must exhibit or a constraint that it must respect.
Quality attribute	A kind of nonfunctional requirement that describes a service or performance characteristic of a product.
System requirement	A top-level requirement for a product that contains multiple subsystems, which could be all software or software and hardware.
User requirement	A goal or task that specific classes of users must be able to perform with a system, or a desired product attribute.

Requirements Analysis

Step 2: Analysis

Requirements Analysis – gaining richer and more precise understanding of each requirement and representing sets of requirements in multiple ways.

Key actions

- analyzing users' information to distinguish task goals from functional RQs, quality expectations, business rules, suggested solutions, and other information
- decomposing high-level RQs into an appropriate level of detail
- deriving functional RQs from other requirements information
- understanding the relative importance of quality attributes
- allocating RQs to software components defined in the system architecture
- negotiating implementation priorities
- identifying gaps in RQs or unnecessary RQs as they relate to the defined scope

Requirements Specification

Step 3: Specification (i.e. Specifying Requirements)

Requirements specification – involves representing and storing the collected requirements knowledge in a persistent and well-organized fashion.

Key action

- translating the collected user needs into written requirements and diagrams suitable for comprehension, review, and use by their intended audiences

Requirements Styles: Contract Style

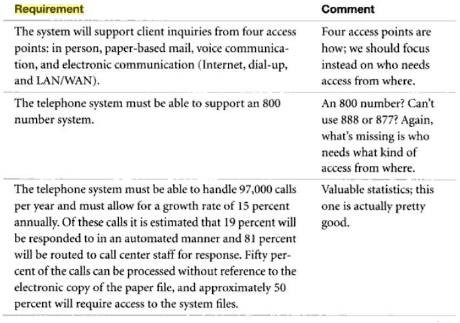
- abstract all requirements with little context
- provide a contract between sponsors and developers
- can run to hundreds of pages

- Classify product features as
  - Must-have features

    "The product must conform to accessibility guidelines"
  - May-have features

    "The product may eventually be voice-controlled"
  - Must-not-have features

    "The product supports only one language"
  - be explicit about must-not-have features



Requirement	Comment
The system will support client inquiries from four access points: in person, paper-based mail, voice communication, and electronic communication (Internet, dial-up, and LAN/WAN).	Four access points are how; we should focus instead on who needs access from where.
The telephone system must be able to support an 800 number system.	An 800 number? Can't use 888 or 877? Again, what's missing is who needs what kind of access from where.
The telephone system must be able to handle 97,000 calls per year and must allow for a growth rate of 15 percent annually. Of these calls it is estimated that 19 percent will be responded to in an automated manner and 81 percent will be routed to call center staff for response. Fifty percent of the calls can be processed without reference to the electronic copy of the paper file, and approximately 50 percent will require access to the system files.	Valuable statistics; this one is actually pretty good.

[Daryl Kulak, Eamonn Guiney: Use cases: requirements in context]

Requirements Styles: Use Case Style (1/2)

- An **actor** is something that can act – a person, a system, or an organization
- A **scenario** is a specific sequence of actions and interactions between actors (where at least one actor is a system)
- A **use case** is a collection of related scenarios – successful and failing ones
- Shall be useful for clients as well as for developers

Requirements Styles: Use Case Style (2/2)

Actors and Goals

- What are the **boundaries** of the system? Is it the software, hardware and software, also the user, or a whole organization?
- Who are the primary actors – i.e., the **stakeholders**?
- What are the **goals** of these actors?
- Describe how the system fulfills these goals (including all exceptions)

Use Case Example: Safe Home (1/3)

Initial Scenario

Use case: display camera views

Actor: homeowner

If I'm at a remote location, I can use any PC with appropriate browser software to log on to the SafeHome Web site. I enter my user ID and two levels of passwords and, once I'm validated, I have access to all the functionality. To access a specific camera view, I select "surveillance" and then "select a camera". Alternatively, I can look at thumbnail snapshots from all cameras by selecting "all cameras". Once I choose a camera, I select "view" ...

[...]

Use Case Example: Safe Home (2/3)

Refined Scenario

Use case: display camera views

Actor: homeowner

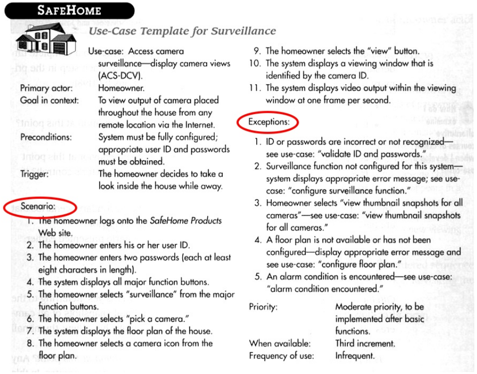
1. The homeowner logs on to the Web Site
2. The homeowner enters his/her user ID
3. The homeowner enters two passwords
4. The system displays all major function buttons
5. The homeowner selects “surveillance” button
6. The homeowner selects “Pick a camera” ...

[...]

Use Case Example: Safe Home (3/3)

Alternative Interactions

- Can the actor take some other action at this point?
- Is it possible that the actor encounters some error condition? If so, which one?
- Is it possible that some other behavior is encountered? If so, which one?



SAFEHOME

Use-Case Template for Surveillance

Use-case:	Access camera surveillance—display camera views (ACS-DCV).
Primary actor:	Homeowner.
Goal in context:	To view output of camera placed throughout the house from any remote location via the Internet.
Preconditions:	System must be fully configured; appropriate user ID and passwords must be obtained.
Trigger:	The homeowner decides to take a look inside the house while away.

Scenario:

1. The homeowner logs onto the *SafeHome Products* Web site.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects "surveillance" from the major function buttons.
6. The homeowner selects "pick a camera."
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the "view" button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

Exceptions:

1. ID or passwords are incorrect or not recognized—see use-case: "validate ID and passwords."
2. Surveillance function not configured for this system—system displays appropriate error message; see use-case: "configure surveillance function."
3. Homeowner selects "view thumbnail snapshots for all cameras"—see use-case: "view thumbnail snapshots for all cameras."
4. A floor plan is not available or has not been configured—display appropriate error message and see use-case: "configure floor plan."
5. An alarm condition is encountered—see use-case: "alarm condition encountered."

Priority:	Moderate priority, to be implemented after basic functions.
When available:	Third increment.
Frequency of use:	Infrequent.

[...]

→ Exploring alternatives the key to successful requirements analysis

Requirements Styles: Goal-Based Notation (1/2)

- **Goal structuring notation (GSN)** as a prominent semi-formal notation (cp. [1])
- graphical notation used to "document explicitly the elements and structure of an argument and the argument's relationship to evidence."
- ISO 26262-10, paragraph 5.3.1 explicitly mentions the GSN as a suitable notation for a safety case argument

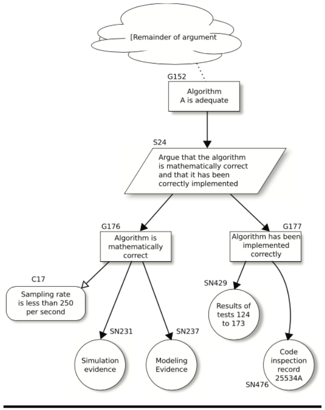
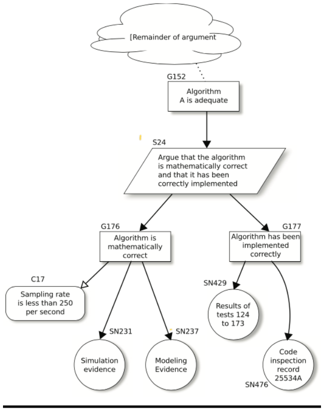

[Remainder of argument
G152
Algorithm A is adequate
S24
Argue that the algorithm is mathematically correct and that it has been correctly implemented
G176
Algorithm is mathematically correct
G177
Algorithm has been implemented correctly
C17
Sampling rate is less than 250 per second
SN429
Results of tests 124 to 173
SN231
SN237
Simulation evidence
Modeling Evidence
Code inspection record 25534A
SN476

Figure A.1 Sample of a GSN argument.

[Chris Hobbs: Embedded Software Development for Safety-Critical Systems]

Requirements Styles: Goal-Based Notation (2/2)

- **GSN symbols**:
  - **rectangle** contains a claim (known as a "goal" in GSN-speak). This will be the statement whose truth we want to demonstrate. For example: "The German word for Friday is Freitag."
  - **parallelogram** contains a strategy: What strategy is used to justify the claim? For example: "The strategy to determine the German word for Friday will be to consult a reputable dictionary."
  - **rounded rectangle** contains a context: The claim is made within a particular context. For example: "We only claim that German word for Friday is Freitag in modern German – we make no claim about older forms of the language."
  - **oval** contains an assumption: The strategy is acceptable if it is assumed that … For example: "We assume that there is a single German word for Friday (unlike, for example, Saturday)."
  - **circle** represents a solution: effectively a description of the evidence that will be presented to justify the claim. For example: "The evidence to support the claim that the German word for Friday is Freitag is the entry on page 234, second column, of Duden's official EU wordlist (ISBN 12-120192-199281)."
  - **diamond** represents incomplete work; there should be none of these in the final case.



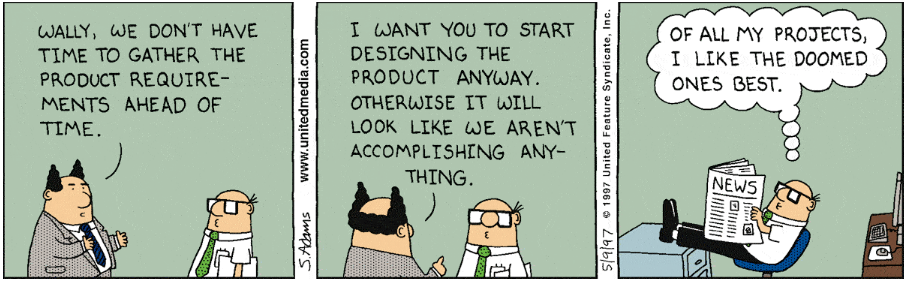
[Remainder of argument
G152
Algorithm A is adequate
S24
Argue that the algorithm is mathematically correct and that it has been correctly implemented
G176
Algorithm is mathematically correct
G177
Algorithm has been implemented correctly
C17
Sampling rate is less than 250 per second
SN429
Results of tests 124 to 173
SN231
Simulation evidence
SN237
Modeling Evidence
Code inspection record 25534A
SN476

Figure A.1 Sample of a GSN argument.

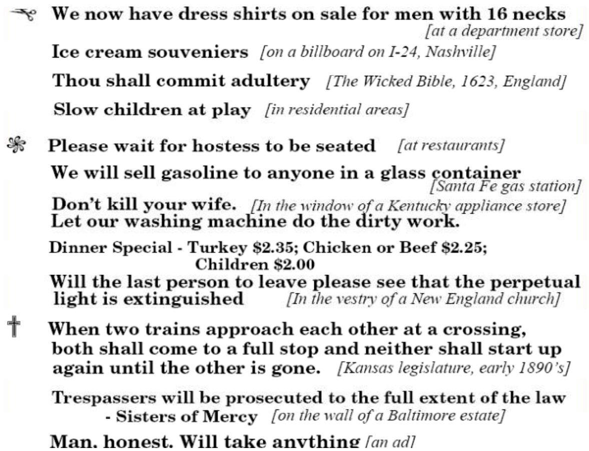
[Chris Hobbs: **Embedded Software Development for Safety-Critical Systems**]

The Software Requirements Specification (SRS)

Knowing Your Requirements Is Important, But Also Their Quality


WALLY, WE DON'T HAVE TIME TO GATHER THE PRODUCT REQUIRE- MENTS AHEAD OF TIME.
I WANT YOU TO START DESIGNING THE PRODUCT ANYWAY. OTHERWISE IT WILL LOOK LIKE WE AREN'T ACCOMPLISHING ANY- THING.
OF ALL MY PROJECTS, I LIKE THE DOOMED ONES BEST.
NEWS
www.unitedmedia.com
S. Adams
5/9/97 © 1997 United Feature Syndicate, Inc.

Many Requirements Problems Arise From Inaccurate Natural Language



We now have dress shirts on sale for men with 16 necks *[at a department store]*

Ice cream souveniers *[on a billboard on I-24, Nashville]*

Thou shall commit adultery *[The Wicked Bible, 1623, England]*

Slow children at play *[in residential areas]*

Please wait for hostess to be seated *[at restaurants]*

We will sell gasoline to anyone in a glass container *[Santa Fe gas station]*

Don't kill your wife. *[In the window of a Kentucky appliance store]* **Let our washing machine do the dirty work.**

Dinner Special - Turkey $2.35; Chicken or Beef $2.25; Children $2.00

Will the last person to leave please see that the perpetual light is extinguished *[In the vestry of a New England church]*

When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other is gone. *[Kansas legislature, early 1890's]*

Trespassers will be prosecuted to the full extent of the law - Sisters of Mercy *[on the wall of a Baltimore estate]*

Man. honest. Will take anything *[an ad]*

[Lawrence Chung]

Typical Requirements Problems

- Insufficient user involvement
- Inaccurate planning
- Creeping user requirements
- Ambiguous requirements
- Gold plating
- Overlooked stakeholders

Qualities of a Requirement: Valid (or “Correct”)

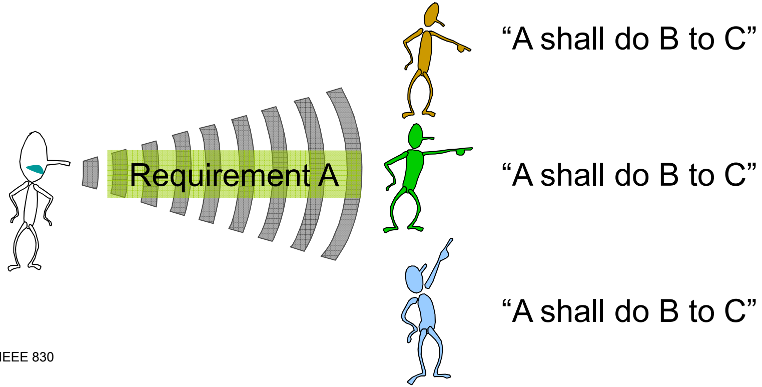
A requirement is valid if:

- Expresses the real needs of the stakeholders (customers, users, ...)
- Does not contain anything that is not “required”
- Example: *“When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other is gone.”* [Kansas legislature, early 1890’s]
  - How would you detect these potential problems?
  - How should you deal with “feature interaction problems”?
  - $\rightarrow$ scenario analysis and/or formalism

Qualities of a Requirement: Unambiguous (1/3)

A requirement is unambiguous:

- Every statement can be read in exactly one way (has only one interpretation)



ref - IEEE 830

Qualities of a Requirement: Unambiguous (2/3)

Mary had a little lamb.

had -	Past of have
have -	1a: To hold in possession as property
	4a: To acquire or get possession of: OBTAIN (best to be had)
	4c: ACCEPT; to have in marriage
	5a: To be marked or characterized by (have red hair)
	10a: To hold in a position of disadvantage or certain defeat
	10b: TRICK, FOOL (been had by a partner)
	12: BEGET, BEAR (have a baby)
	13: To partake of (have dinner)
	14: BRIBE, SUBORN (can be had for a price)
lamb -	1a: A young sheep esp. less than one year old or without permanent teeth
	1b: The young of various other animals (as smaller antelopes)
	2a: A person as gentle or weak as a lamb
	2b: DEAR, PET
	2c: A person easily cheated or deceived especially in trading securities
	3a: The flesh of lamb used as food

[Lawrence Chung]

Qualities of a Requirement: Unambiguous (3/3)

Have	Lamb	Interpretation
1a	1a	Mary owned a little sheep under one year of age or without permanent teeth.
4a	1a	Mary acquired a little sheep under one year of age or without permanent teeth.
5a	1a	Mary is the person who owned a little sheep under one year of age or without permanent teeth.
10a	1a	Mary held a little sheep under one year of age or without permanent teeth in a position of disadvantage.
10b	1a	Mary tricked a little sheep under one year of age or without permanent teeth.
12	1b	Mary gave birth to a young antelope.
12	2a	Mary is (or was) the mother of a particular small, gentle person.
13	3a	Mary ate a little of the flesh of lamb.
14	2c	Mary bribed a small person trading in securities who was easily cheated.

[Lawrence Chung]

Requirements Specification Completeness

A requirement specification is complete if:

- it contains all the things the system **must do** ... and all the things it **must not do**
- **conceptual completeness is reached**
  - e.g.: responses to all classes of input
- **structural completeness is reached**
  - e.g.: no TBDs left (to be done)

- How would you find missing requirements?
  - → elicitation techniques
  - → use of "ontological" primitives (goals, agents, decisions, rationale, entities, activities, constraints)
  - → use of organizational primitives (classes, associations/aggregations, generalizations, views)

Qualities of a Requirement: Verifiable

A requirement is verifiable if:

- There exists some finite, cost-effective process with which a person or machine can check that the product meets the requirement. (it exists a process to test satisfaction of each requirement)

Are these requirements verifiable? If not, what is a better way to state them?

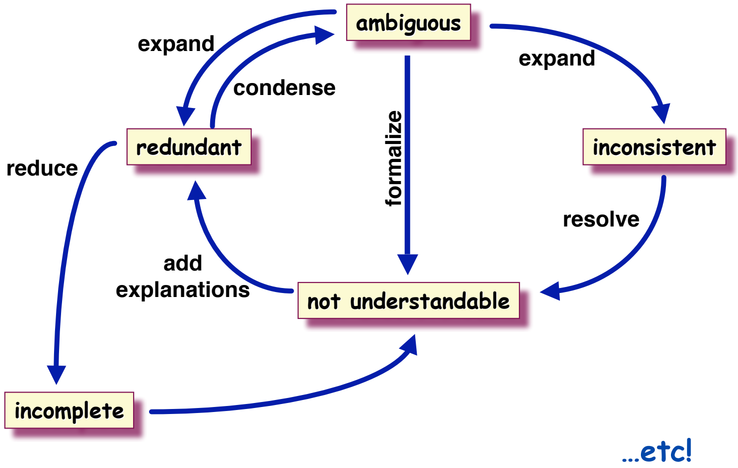
- The system supports up to 1,000 simultaneous users.
- The system shall respond to an arbitrary query in 500 msec.
- The color shall be a pleasing shade of green.
- The system shall be available 24 x 7.
- The system shall export view data in comma-separated format, according to the IEEE specification.

Desiderata for Requirements Specifications

- **Valid (or "correct")**
  - Expresses the real needs of the stakeholders (customers, users,…)
  - Does not contain anything that is *not* "required"
- **Unambiguous**
  - Every statement can be read in exactly one way
- **Complete**
  - All the things the system must do…
  - …and all the things it must not do!
  - Conceptual Completeness
    - E.g. responses to all classes of input
  - Structural Completeness
    - E.g. no TBDs!!!
- **Understandable (Clear)**
  - E.g. by non-computer specialists
- **Consistent**
  - Doesn't contradict itself
  - Uses all terms consistently
- **Ranked**
  - Indicates relative importance / stability of each requirement
- **Verifiable**
  - A process exists to test satisfaction of each requirement
- **Modifiable**
  - Can be changed without difficulty
    - Good structure and cross-referencing
- **Traceable**
  - Origin of each requirement is clear
  - Labels each requirement for future referencing

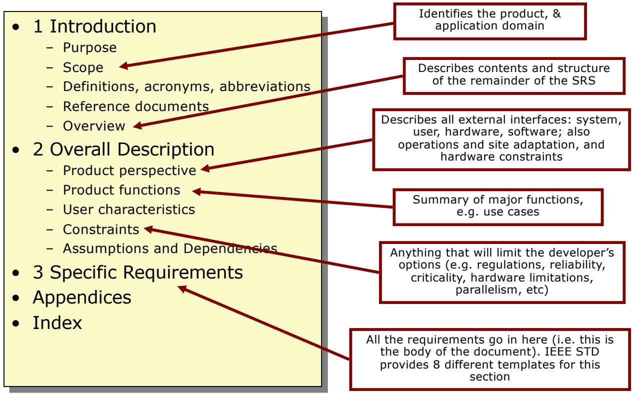
[adapted from IEEE-STD-830-1998] by Steve Easterbrook

There Is No Perfect SRS: Constant Evolution



[Steve Easterbrook, CSC340F]

A Standard for Software Requirements Specifications [IEEE 830:1998]



- 1 Introduction
  - Purpose
  - Scope
  - Definitions, acronyms, abbreviations
  - Reference documents
  - Overview
- 2 Overall Description
  - Product perspective
  - Product functions
  - User characteristics
  - Constraints
  - Assumptions and Dependencies
- 3 Specific Requirements
- Appendices
- Index

Identifies the product, & application domain

Describes contents and structure of the remainder of the SRS

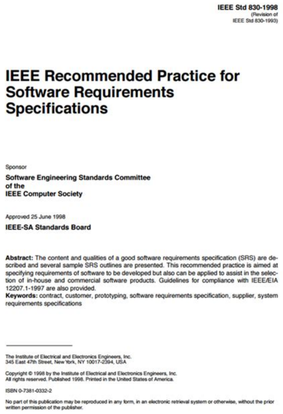
Describes all external interfaces: system, user, hardware, software; also operations and site adaptation, and hardware constraints

Summary of major functions, e.g. use cases

Anything that will limit the developer's options (e.g. regulations, reliability, criticality, hardware limitations, parallelism, etc)

All the requirements go in here (i.e. this is the body of the document). IEEE STD provides 8 different templates for this section

[IEEE-STD-830-1998], [Blum 1992, p160]



IEEE Std 830-1998
(Revision of
IEEE Std 830-1993)

IEEE Recommended Practice for Software Requirements Specifications

Sponsor

Software Engineering Standards Committee of the IEEE Computer Society

Approved 25 June 1998

IEEE-SA Standards Board

Abstract: The content and qualities of a good software requirements specification (SRS) are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. Guidelines for compliance with IEEE/EIA 12207.1-1997 are also provided.

Keywords: contract, customer, prototyping, software requirements specification, supplier, system requirements specifications

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1998 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1998. Printed in the United States of America.

ISBN 0-7381-0332-2

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

[IEEE-STD-830-1998]

[IEEE 830:1998] Section 3: Requirements

- 3.1 External Interface Requirements
  - 3.1.1 User Interfaces
  - 3.1.2 Hardware Interfaces
  - 3.1.3 Software Interfaces
  - 3.1.4 Communication Interfaces
- 3.2 Functional Requirements
  - *this section organized by mode, user class, feature, etc. For example:*
  - 3.2.1 Mode 1
    - *3.2.1.1 Functional Requirement 1.1*
    - ...
  - 3.2.2 Mode 2
    - *3.2.1.1 Functional Requirement 1.1*
    - ...
  - ...
  - 3.2.2 Mode n
    - ...

3.3 Performance Requirements

Remember to state this in measurable terms!

3.4 Design Constraints

3.4.1 Standards compliance

3.4.2 Hardware limitations

etc.

3.5 Software System Attributes

3.5.1 Reliability

3.5.2 Availability

3.5.3 Security

3.5.4 Maintainability

3.5.5 Portability

3.6 Other Requirements

[IEEE-STD-830-1998], [Blum 1992, p160]

Questions?