2IS55 Software Evolution

Requirements Evolution

Alexander Serebrenik

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Important! Small formula correction

- **A = w1*A1 + w2*A2 + w3*A3**
- The lowest grade is multiplied with 0.1
- The remaining grades are multiplied with 0.2

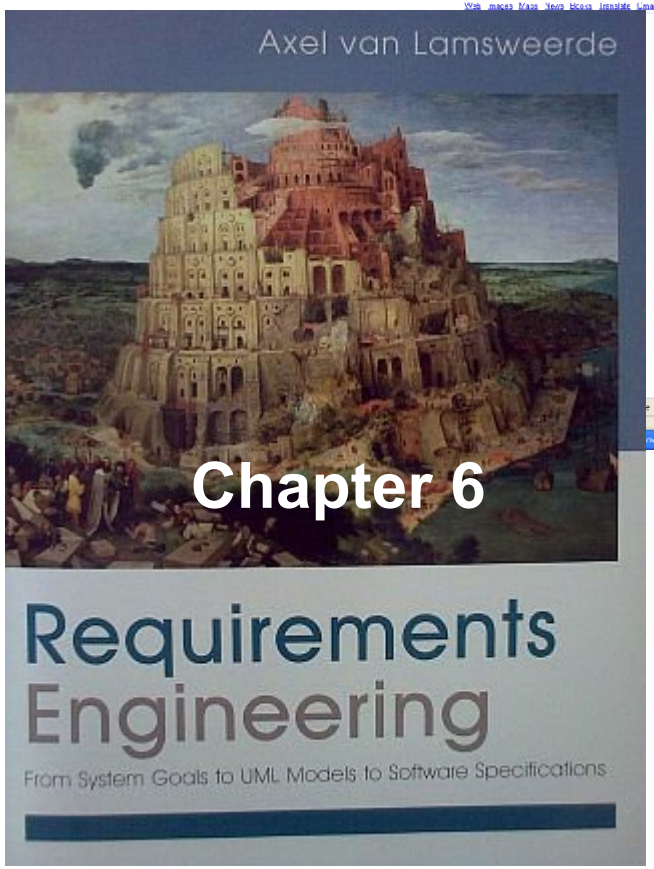
- **A = w1*A1 + w2*A2 + w3*A3**
- The lowest grade is multiplied with ***0.2***
- The remaining grades are multiplied with ***0.4***

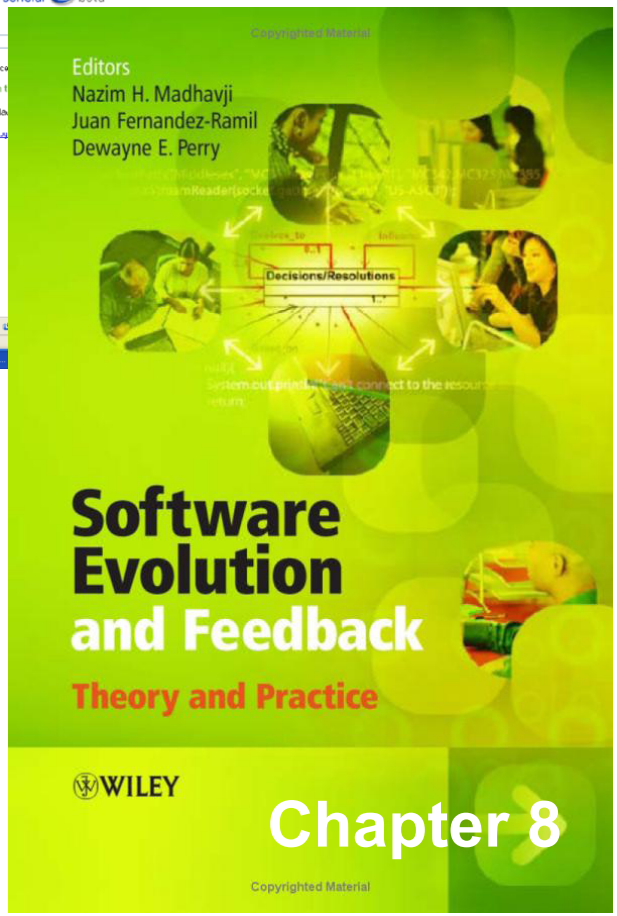
Assignment 1: Requirements Evolution

- VEMUS, Virtual European Music School
  - Long document, you do not need to read it all!
- Goal: analyze the suitability for evolution
- PDF
- How to submit: Peach
  - http://peach.win.tue.nl/
  - Deadline: February 19, 23:59

Sources


Google Scholar - Mozilla Firefox
Google
scholar beta

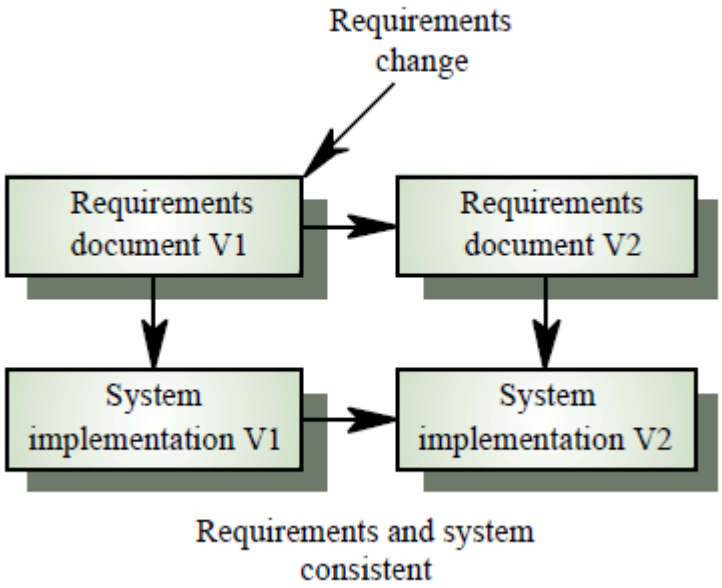

Axel van Lamsweerde
Chapter 6
Requirements Engineering
From System Goals to UML Models to Software Specifications

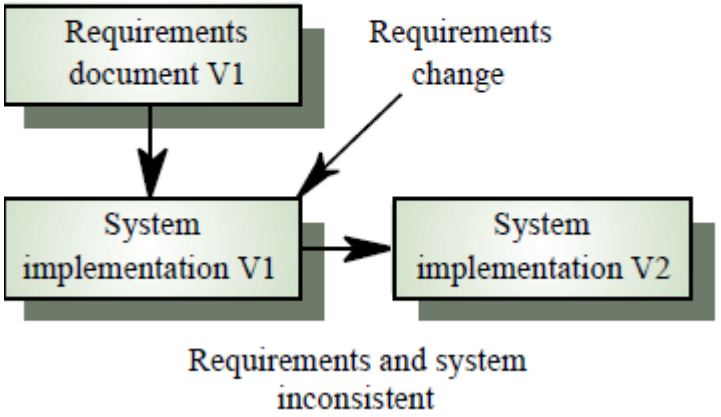

Editors
Nazim H. Madhavji
Juan Fernandez-Ramil
Dewayne E. Perry
Decisions/Resolutions
Software Evolution and Feedback
Theory and Practice
WILEY
Chapter 8

Requirements Evolution

- **Requirements are "just a piece of paper"…**
- **Tend to be forgotten during the evolution**

Sommerville 1996



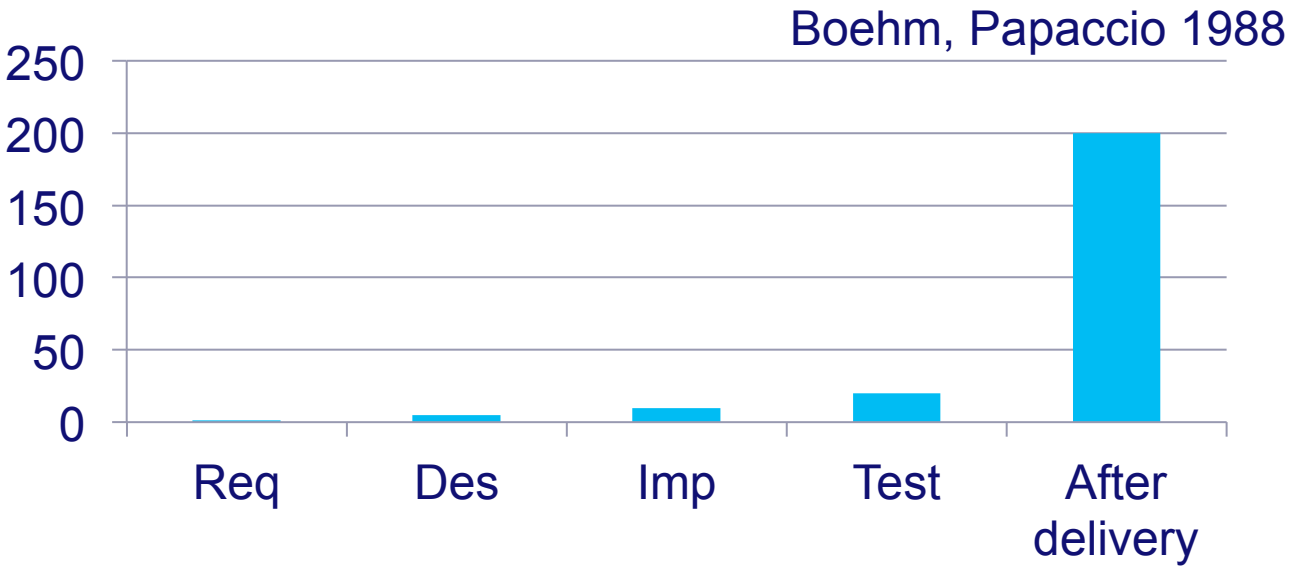


This is how it should be…

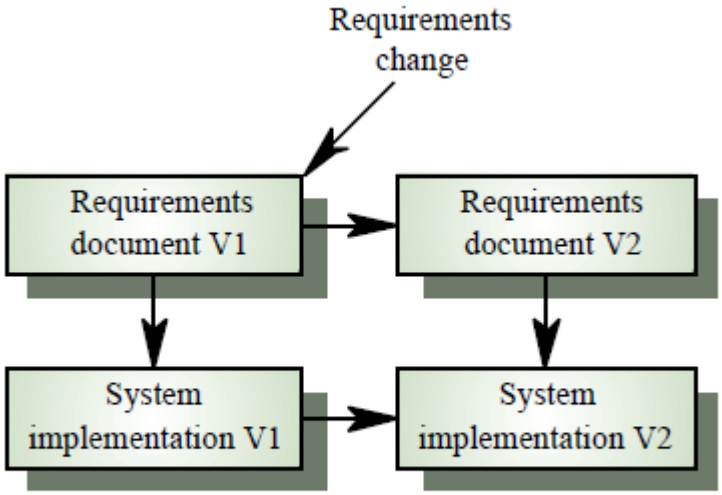
this is how it often is.

Why requirements?

- **Errors in requirements are:**
  - **common:**
    - **25% of all the errors (Jones '91)**
    - **1 error per function point (~ 80 LOC Java; Jones '95)**
  - **expensive**



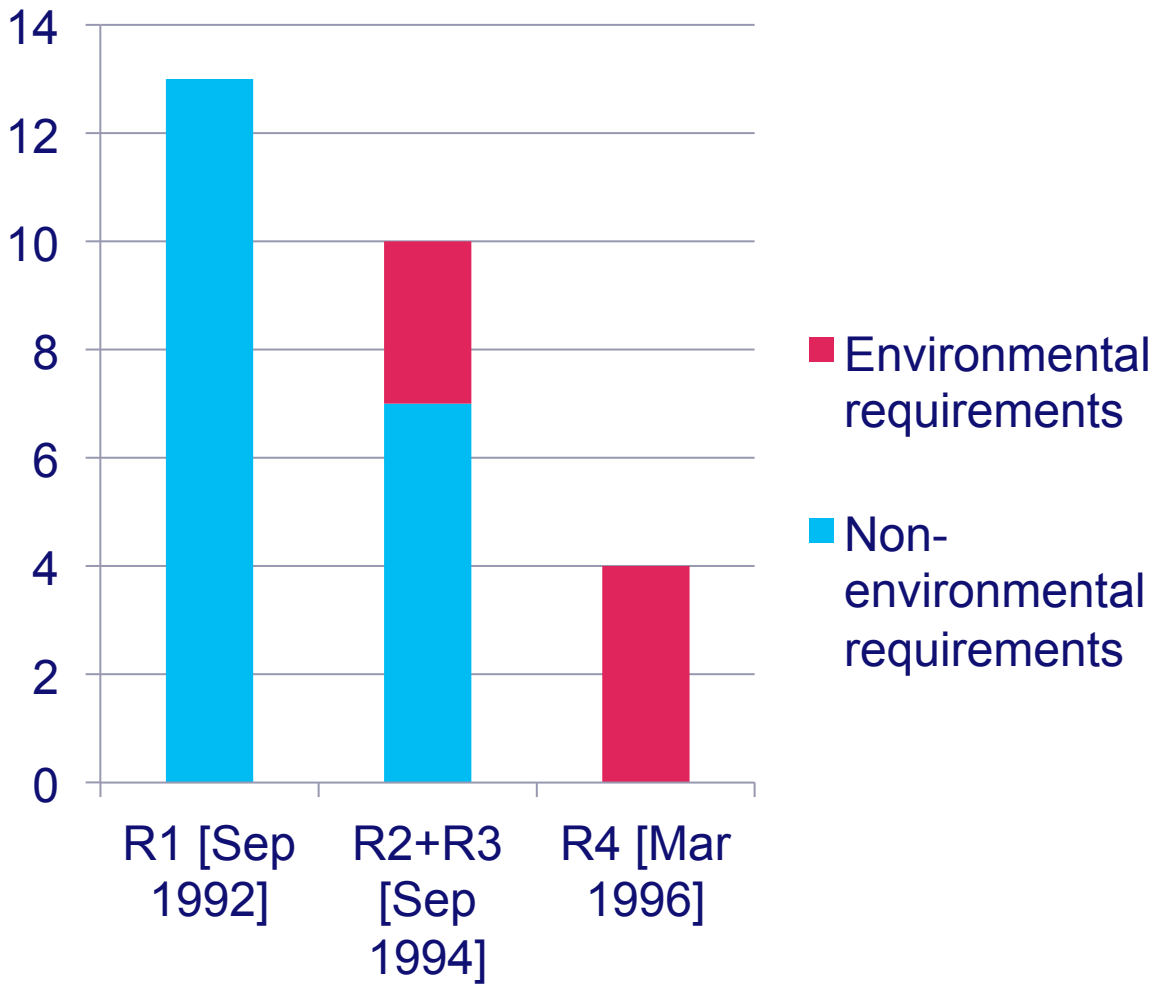
Questions



- **Why** do the requirements evolve?
- How **suited** is a given requirements document for evolution?
- How do the requirements **evolve** and **co-evolve**?

Why do the requirements evolve?

- **Environment changes**
- **[Nanda, Madhavji]**
  - **Congruence Evaluation System**
  - **Research prototype**



Suitability for evolution

- When is evolution difficult?
  - Per requirement:
    - "Bad requirements": vague, subjective, weak, underspecified, overtly complex, unreadable…
    - Volatile requirements
      - Related to dependencies between the requirements
  - Per requirements document:
    - Missing requirements
    - Inconsistent requirements

Bad requirements: Check lists

- **Industrial approach: guidelines, checklists, templates**
- **Manual verification**
  - **“Each requirement should state consequences of losses of availability and breaches of security” (European eGovernment program)**
  - **“Each requirement is measurable” (SMART)**
- **Do you remember what SMART means?**

Bad requirements: Check lists

- **Industrial approach: guidelines, checklists, templates**
- **M**
  -
  -
- **D**

Specific
Measurable
Attainable, Appropriate, Actionable
Realistic
Time-bound, Traceable

Bad requirements: Check lists

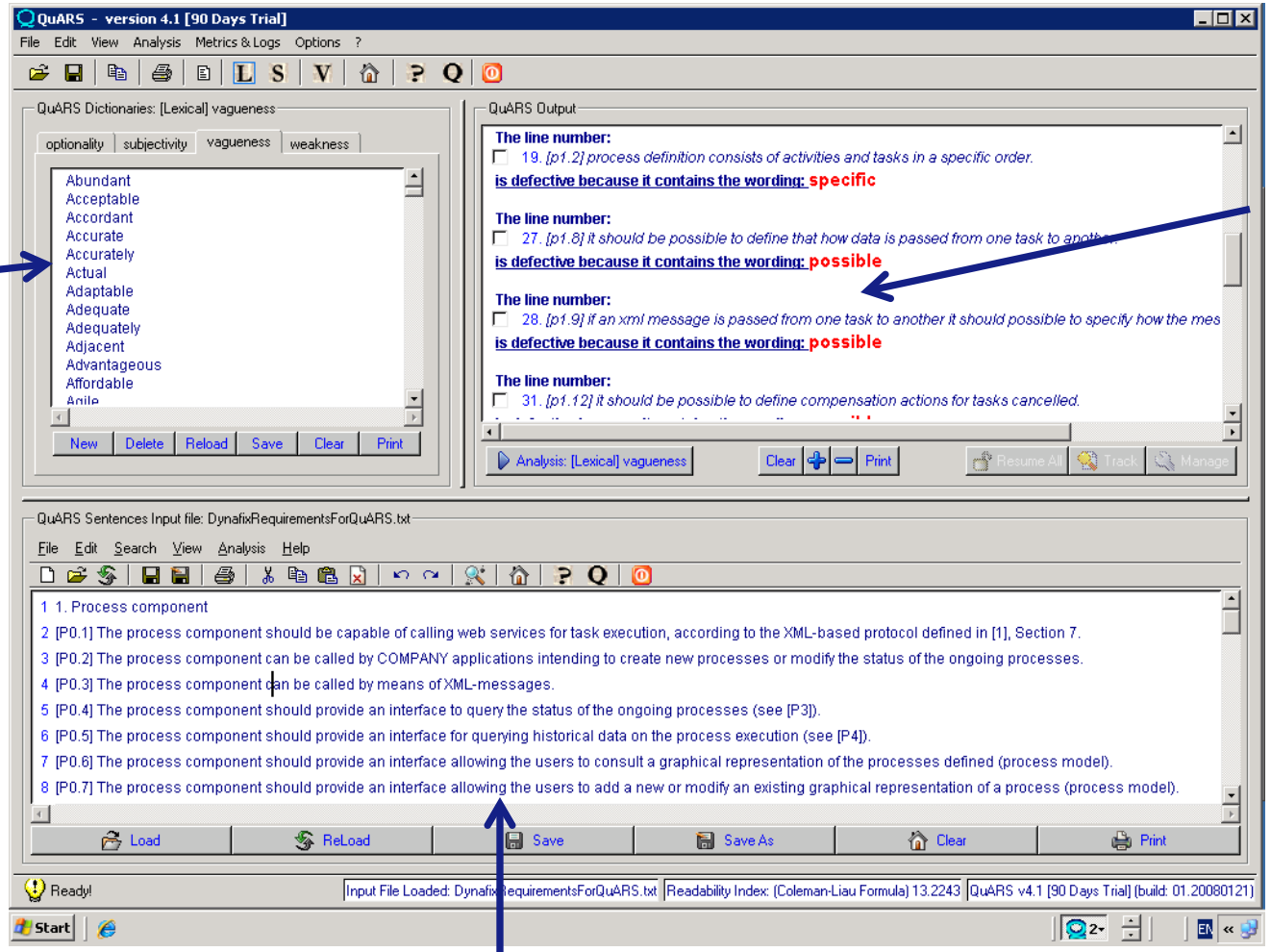
- **Industrial approach: guidelines, checklists, templates**
- **Manual verification**
  - **“Each requirement should state consequences of losses of availability and breaches of security” (European eGovernment program)**
  - **“Each requirement is measurable” (SMART)**
- **Assessment**
  - **is subjective**
  - **requires training and experience**

Bad requirements: More automation?

Problem	Indicators (examples)
Vagueness	clear, significant, useful, adequate, good, bad
Subjectivity	similar, as … as possible, taking … into account
Optionality	possibly, if needed, if appropriate, eventually
Weakness	could, might
Underspecification	(write/read) access, (data/control) flow, “TBD”
Multiplicity	and, or
Implicity	Anaphora (it, these, previous, above)

Fabbrini, Fusani, Gnesi, Lami, 2001.

How to make the requirements better?



Lexical indicators

Defects

Requirements

Readability measurement

- **Requirements from a student project (Horus 2007)**
  - **A. An account is an administrator account, a scientist account or an observer account.**
  - **B. An administrator shall be able to configure whether multiple experiments may be executed simultaneously on a particular satellite.**
  - **C. Experiment data can be retrieved from the system.**

- **Which one is more difficult to read?**
- **Flesch-Kincaid grade level:**
  - **A – 11.2, B – 20.2, C – 8.1**

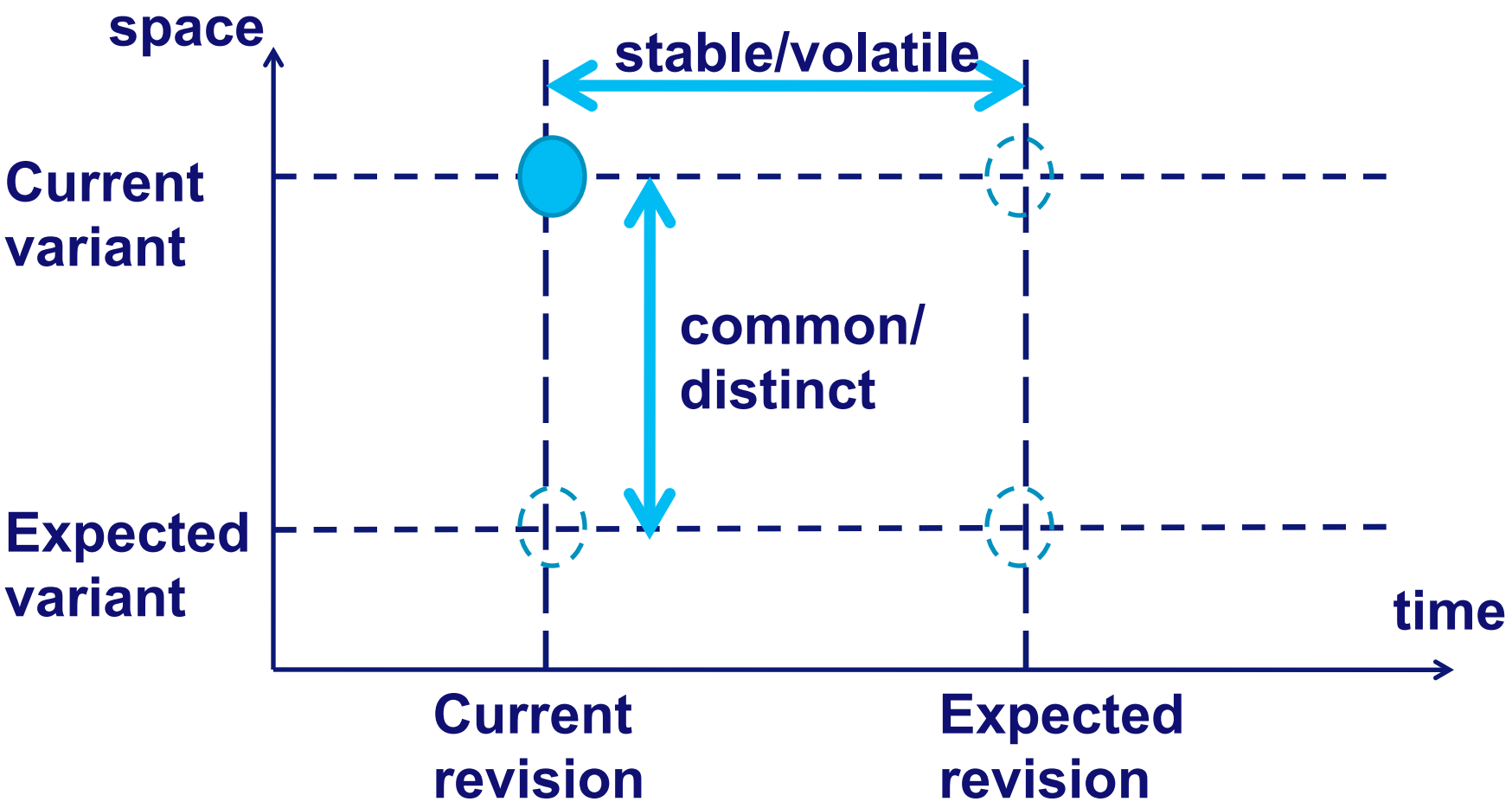
Flesch-Kincaid grade level

- Number of years of (US) education required to understand the text.

$$0.39\left(\frac{\text{total words}}{\text{total sentences}}\right)+11.8\left(\frac{\text{total syllables}}{\text{total words}}\right)-15.59$$

- Might be misleading for smaller texts
  - Use for features instead of individual requirements
  - Feature = group of related requirements
- The NASA study (Wilson,Rosenberg,Hyatt 1997)
  - Mean = 10.76, std dev = 1.59

Second problem: Volatile requirements



- **Variants** – adaptations to different users or environments
- **Revisions** – subsequent versions of one product

How can we identify volatile requirements?

- **Feature = group of related requirements**
  - **Features should separate more and less stable requirements**
  - **Intuition: More stable requirements should not depend on less stable ones**

- **Problems:**
  - **We cannot predict future change**
  - **Analysis of natural language requirements is difficult**

Relative stability

- "Relative stability" – one requirement is more stable (>) than another one.

- Example: meeting schedule system
- Notify the participants vs. Notify the participants by SMS
  - Intention or concept > operation or fact
- Notify the participants vs. Arrange recurrent meetings
  - Core (in any variant/revision) > others
- Notify the participants vs. The system should have an MS Windows "look and feel"
  - Functional > non-functional
- NB: Conflicts and choices among different options are usually quite volatile

Conflicts and choices among different options are usually quite volatile: Example

- Payment Card Industry Security Standards Council
  - Digital Security Standard

- Version 1.2: organizations should change security keys annually (the "best" choice):
  - Too onerous? Not frequent enough?

- Version 2.0: organizations should change keys according to best practices
  - Ambiguity?!

How can we address volatility?

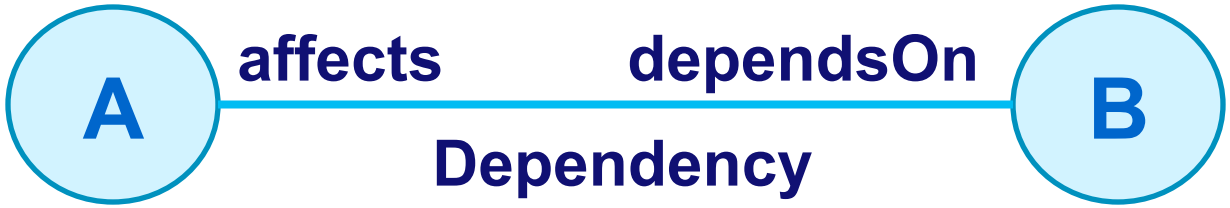
- At requirements engineering time:
  - Try to find a more stable alternative
  - Put special attention to traceability (backwards – rationale, forwards – design, implementation, tests)
  - Anticipate and record responses for future changes

- At design time:
  - Encapsulate volatile requirements in separate modules

Volatility and dependencies

- **“More stable requirements should not depend on less stable ones”**

- **A affects B (B depends on A) if changing A might require changing B.**



Types of dependencies (examples)

- **Use**
  - A explicitly refers to B
  - "IMSETY shall adhere to Table 2.1 for user rights"
- **Generalization/refinement**
  - A is a more general case of B
  - "Observers shall not be authorized to manipulate experiments."

	Users		
Entities	Administrator	Scientist	Observer
Payload	CRUD	R	R
Satellite	CRUD	-	-
User	CRUD	RU	-
Payload command list	CRUD	R	-
Experiment	CRUD	CRUD	R
Observation	RD	R	R
Representation	-	CRD	CRD

Table 2.1: CRUD matrix

Types of dependencies (examples, continued)

- **Temporal**
  - Satisfaction of A should precede/follow satisfaction of B
  - "IMSETY shall require users to be logged in before they can use any of the system's functionality."

- **Satisfiability**
  - Satisfaction of A implies satisfaction of B
  - More general than "generalization/refinement"
  - But also
    - "The system shall interface with an MCS for communication with satellites."
    - "IMSETY shall log all communications with the MCSes."

How can we derive dependency relations?

- Manually
- If requirements are formalized (à la “2IMF30 System validation”) – formal techniques, e.g., using model checking
- Using transitivity

- Keyword-based [Huffman Hayes, Dekhtyar, Osborne 2003]:
  - At any moment, only one scientist is allowed to compose an experiment on a single payload.
  - A scientist shall be able to request the scheduling of the execution of experiments on a predefined moment.

How can we derive dependency relations?

- Manually
- If requirements are formalized (à la “2IMF30 System validation”) – formal techniques, e.g., using model checking
- Using transitivity

2IMW15 Web information retrieval

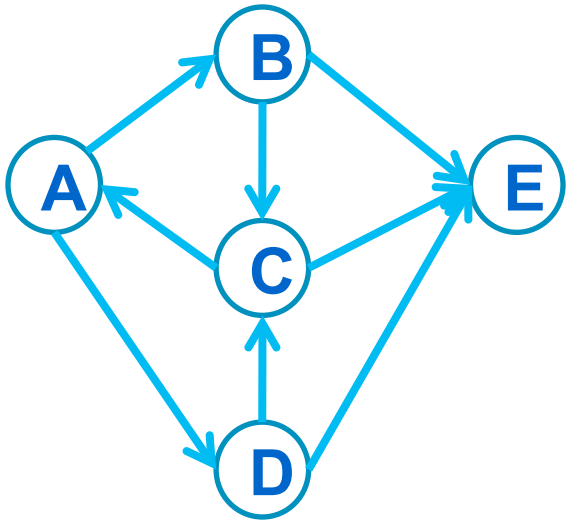
- Keyword-based [Huffma… …e 2003]:
  - At any moment, only one sc… …is allowed to compose an experiment on a single payload.
  - A scientist shall be able to request the scheduling of the execution of experiments on a predefined moment.

Problems (and solutions) with keyword-based

- Terminology vs. regular use
  - scientist, experiment vs. moment
  - solution: domain dictionary
- Synonyms (solution: thesaurus)
- Some requirements are more “essential” for the keyword than others.
- Lexical correlation vs. dependency
  - Which type?

Dependency analysis

- Traceability graph
  - Vertices: requirements
  - Arcs: dependency relations



- What do you think about the requirements document right?
- What does this mean for evolution?
- How would the quality information influence your interpretation?
- What are the limitations of the traceability graph approach?

Inconsistent requirements

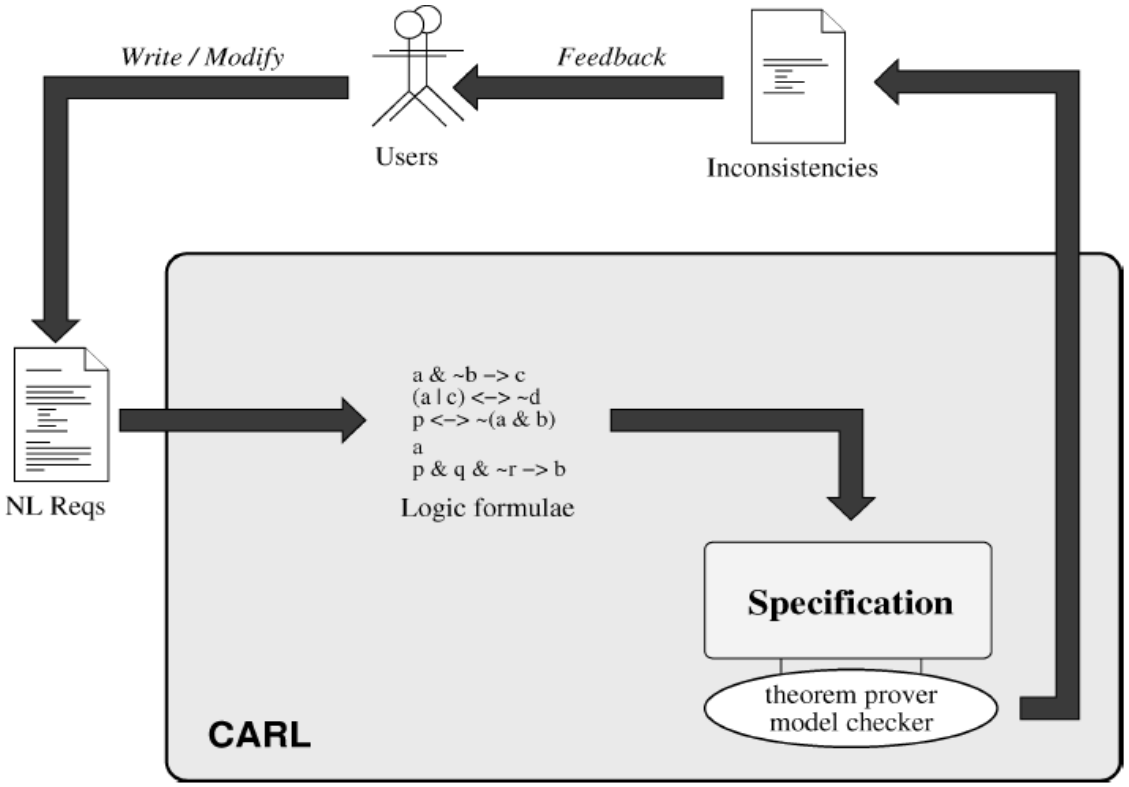
- Different kinds of requirements: use cases, process models, natural language requirements
- Different sources of requirements: multiple documents, multiple stakeholders
- Different types of inconsistencies:
  - Terminological (synonyms, different interpretations of the same term): data dictionary, glossary, ontology
  - Logical (strong) – conjunction of the requirements is *false*
  - Logical (weak) – under some condition conjunction of the requirements is *false*

Logical inconsistencies: Heuristics

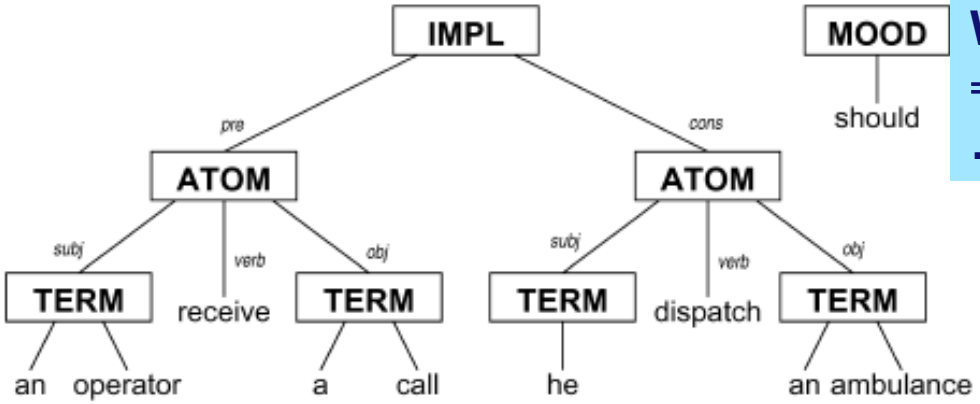
- Logical inconsistencies [van Lamsweerde]
  - Heuristic: scrutinize dependent requirements
  - Heuristic: public availability vs. confidentiality
    - Grades should be publically available
    - Students should not have access to other students' grades
  - Heuristic: increases vs. decreases
    - Increase access to books and journals
    - Reduce operational costs
  - Heuristic: security vs. user-friendliness

Inconsistent requirements: Linguistics+Logics

- **CARL [Gervasi. Zowghi 2005]:**
  - **Translate nat. lang. requirements to logical formulae**
  - **Analyse consistency**



CARL: From text to formula

Original requirement	`When an operator receives a call, he should dispatch an ambulance.`
After morpho-syntactic analysis	`When/CC/WRB an/DT operator/NN receive/VB/Z a/DT call/NN he/PP should/MD dispatch/VB an/DT ambulance/NN`
Parse tree, as produced by the CICO algorithm ($\mathcal{P}$)	
Equivalent logic formula ($\mathcal{T}$)	`receive(operator,call)` → `dispatch(operator,ambulance)` (with priority set to "optional")

WHEN a/SENT b/SENT ⇒ IMP $a $b

…

- **Now CARL can check for (weak) inconsistencies, resolve ambiguities and even "invent" unexpected scenario's.**

CARL: Detect inconsistency

1. **Incident Room Controller**
   a. A medical emergency is either an illness or an accident.
   b. When an operator receives a phone call concerning a medical emergency, (s)he should dispatch a nearby available ambulance.
   c. When an operator receives a phone call concerning a nonmedical emergency, the operator should not dispatch an ambulance, and he should transfer the phone call to another service.
2. **Operations Manager**
   a. When an operator receives a phone call, if an ambulance is not nearby or not available, then the operator should not dispatch that ambulance.
   b. When an operator receives a phone call, (s)he should dispatch a nearby available ambulance.

How can we address inconsistency?

- **Weaken** or **drop** one of the conflicting statements
- **Specialize** the requirement such that the conflict disappears.
  - When an operator receives a phone call *concerning medical emergency*, (s)he should dispatch a nearby available ambulance.
- For weak logical inconsistencies: **avoid** the condition
  - Logical (weak): under some condition conjunction of the requirements is *false*
- **Evaluate** different options and **choose** the "best" one
  - **NB**: Source of volatility

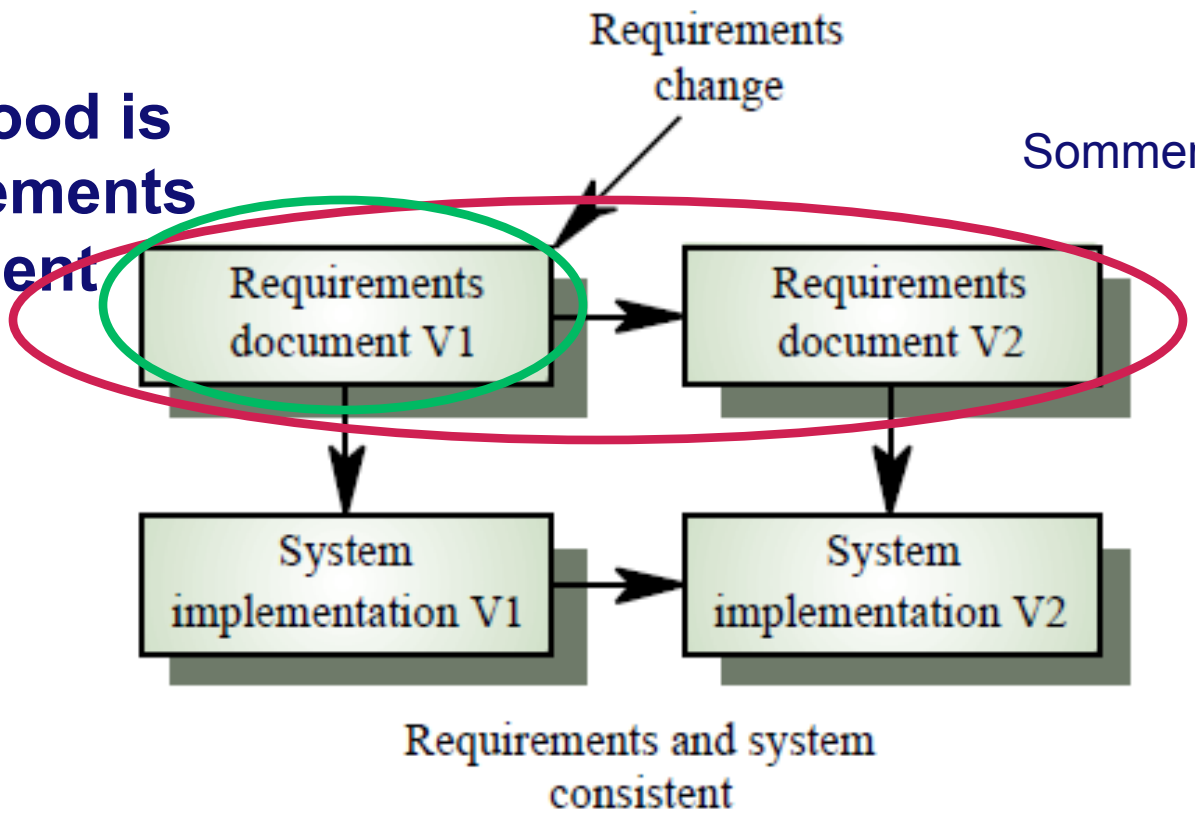
Summary so far…

- Requirements evolution can be hindered by
  - "Bad requirements": vague, subjective, weak, underspecified, overtly complex, unreadable…
  - Volatile requirements
    - Related to dependencies between the requirements
  - Missing requirements
  - Inconsistent requirements

So far…

Next…

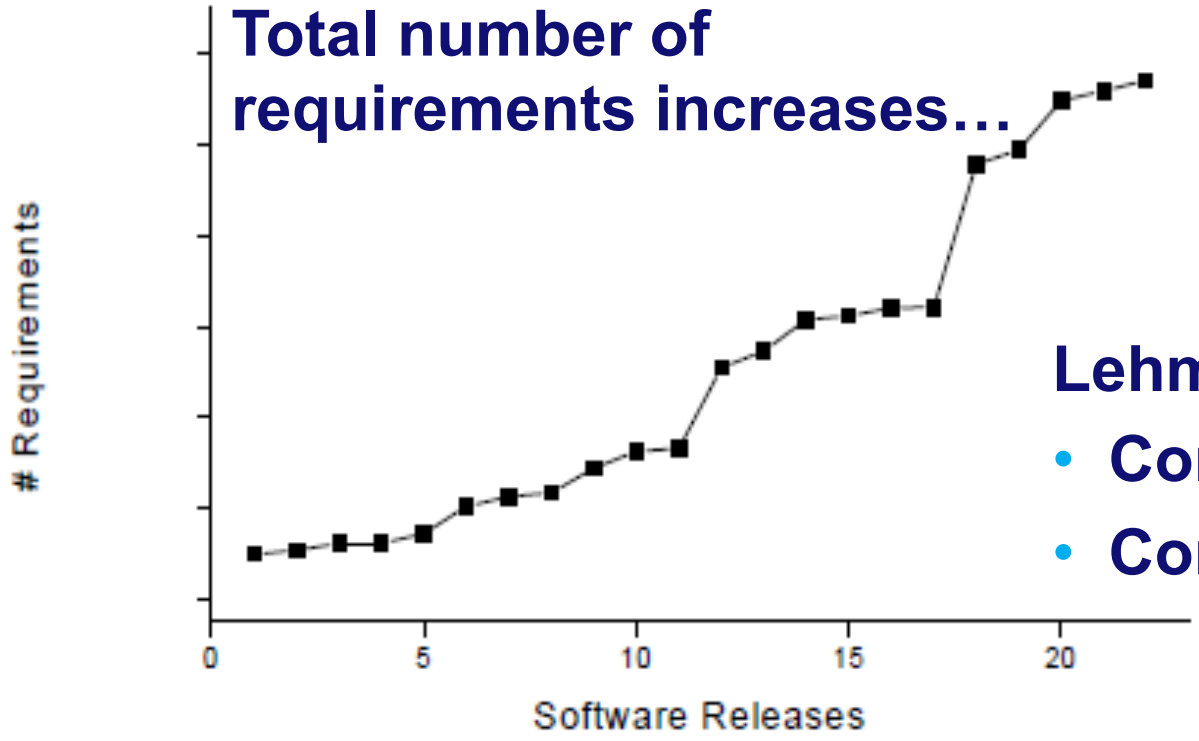
How good is requirements document v1?



Sommerville 1996

How do the requirements evolve?

- Are Lehman's laws applicable?
- [Anderson, Felici 2000]: avionics software

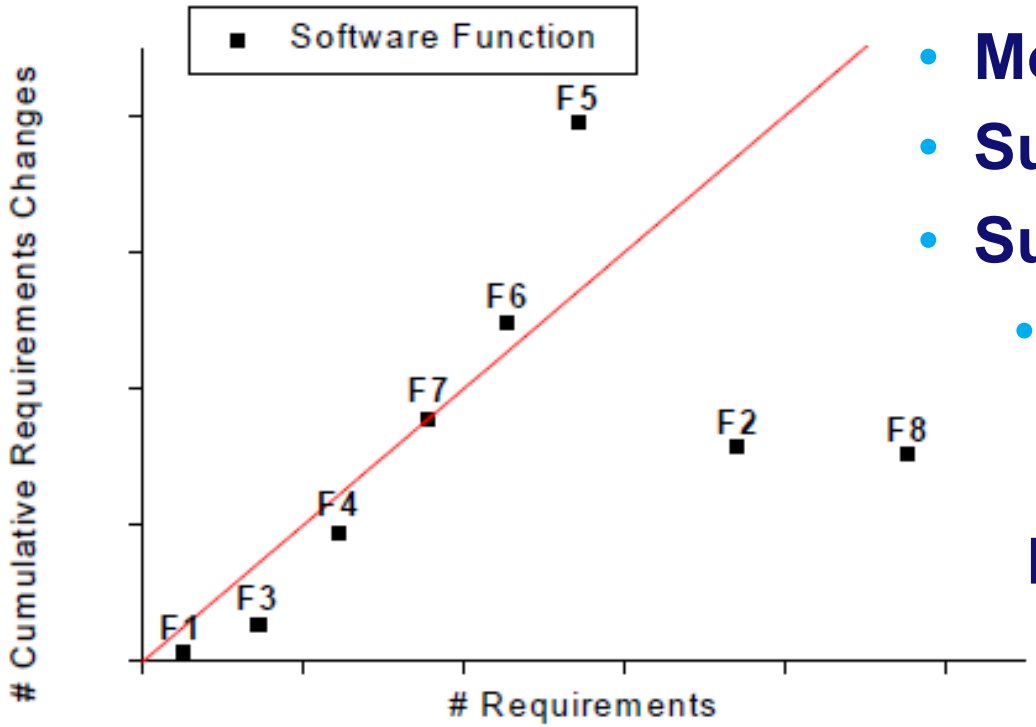


Lehman's laws

- Continuing change
- Continuing growth

Evolution and volatility

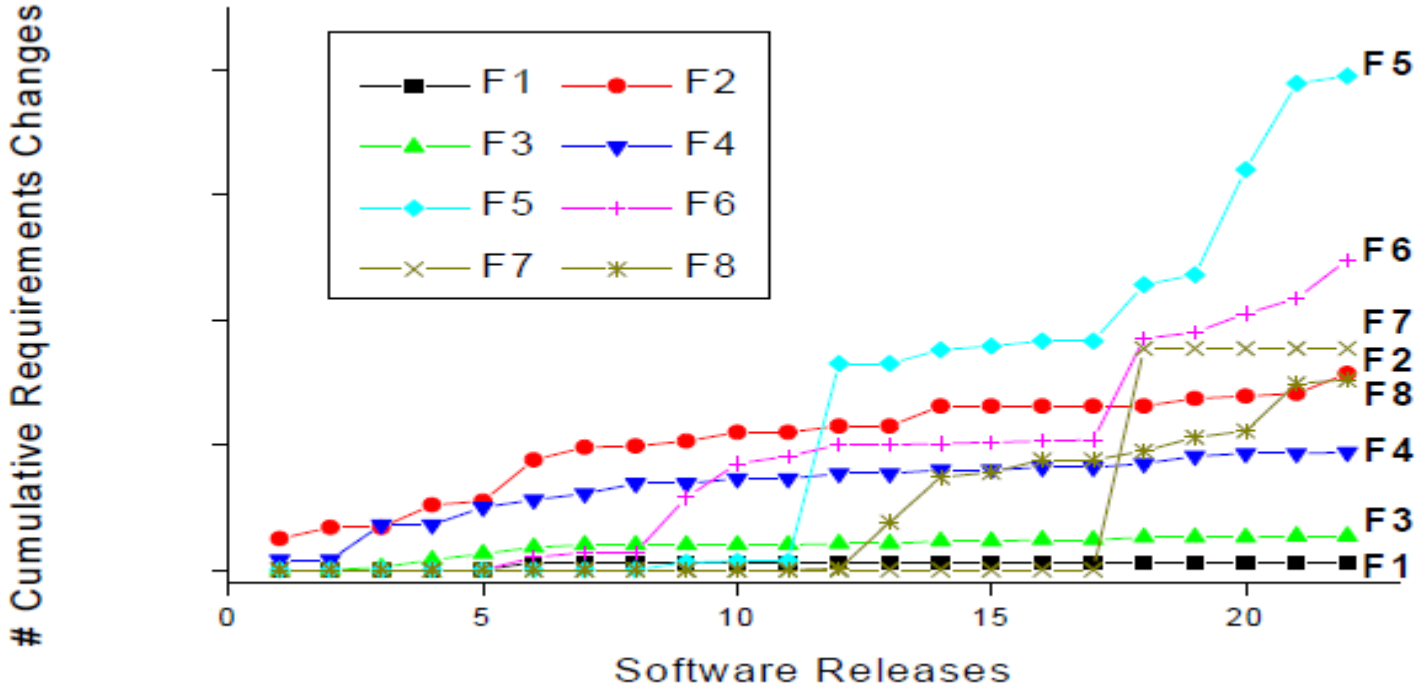
- Volatility = subject to change?
- Eight software functions (=features), different documents



- Mostly linear correlation
- Sublinear: F2, F8
- Superlinear: F5
  - Most likely to change

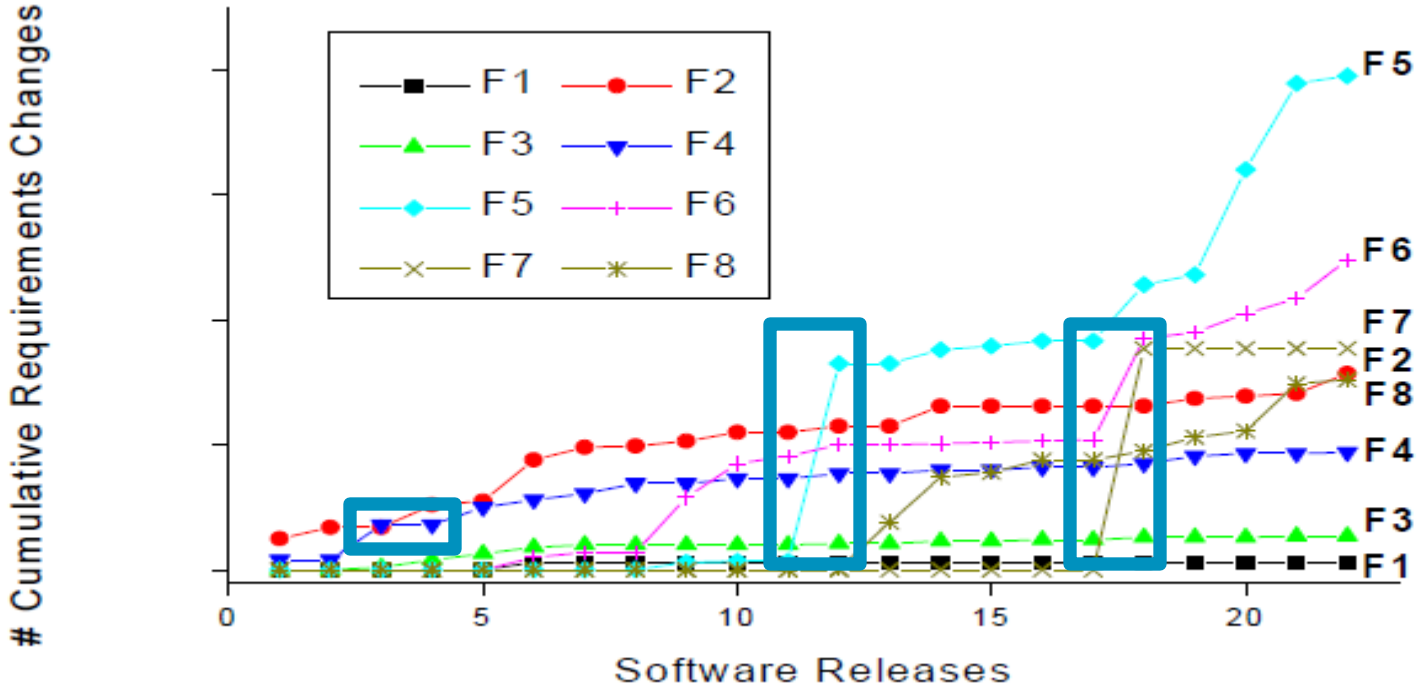
F1 *seems* to be stable

Closer look at the 8 functions (1)



- **F1 is indeed stable.**
- **F1 is about system architecture**
- **Conjecture: system architecture is stable**

Closer look at the 8 functions (2)



- **Different features are likely to change at different times.**

How do the req. evolve – what have we done?

- Calculated
  - the number of requirements
  - the number of changes
  - the cumulative number of changes
- Studied
  - how these values change with time
- Last week something similar for size/complexity/…

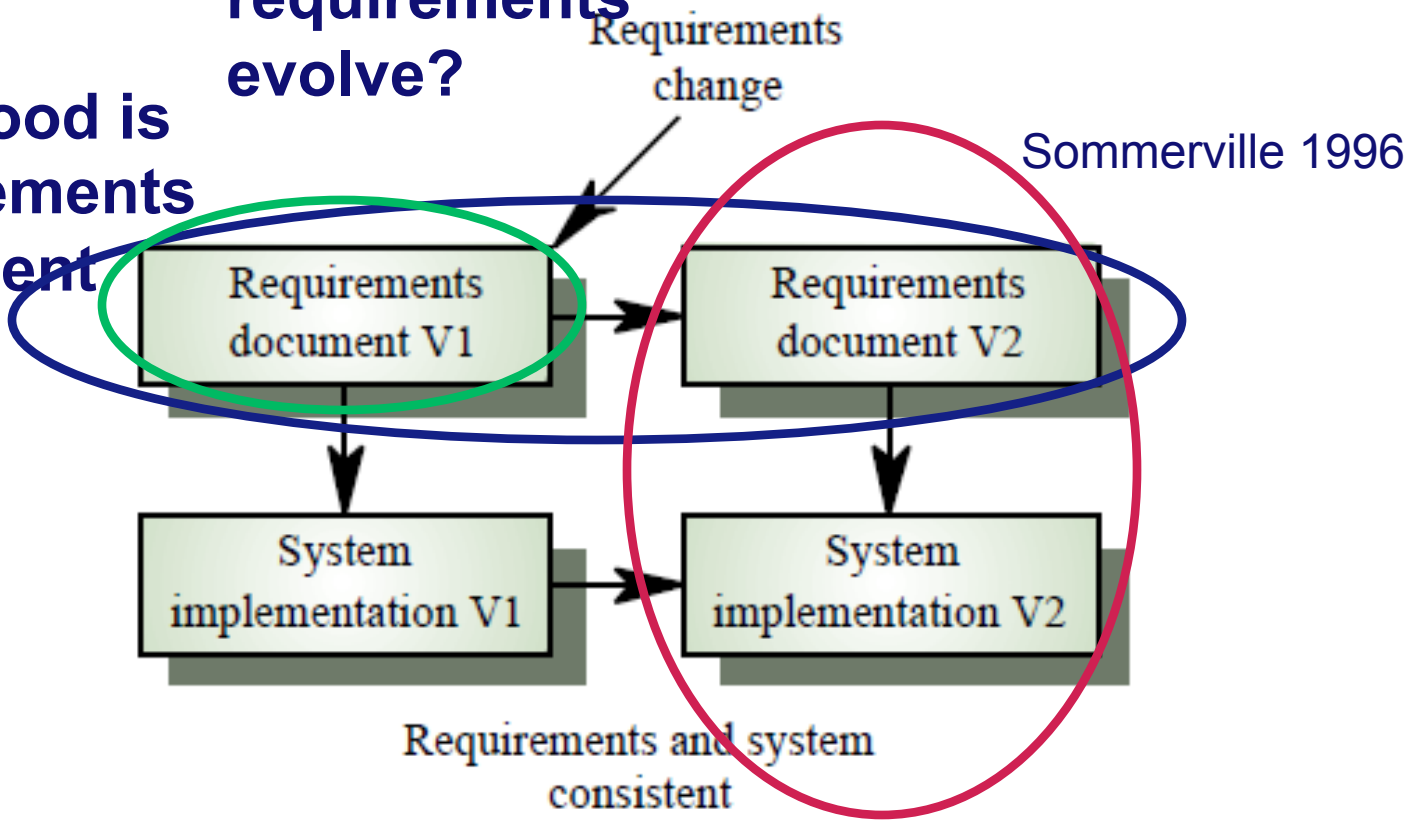
- Generic approach
  - Metrics: function from software artefacts to numbers
  - Time series: sequence of measurements at successive times

So far…

How do the requirements evolve?

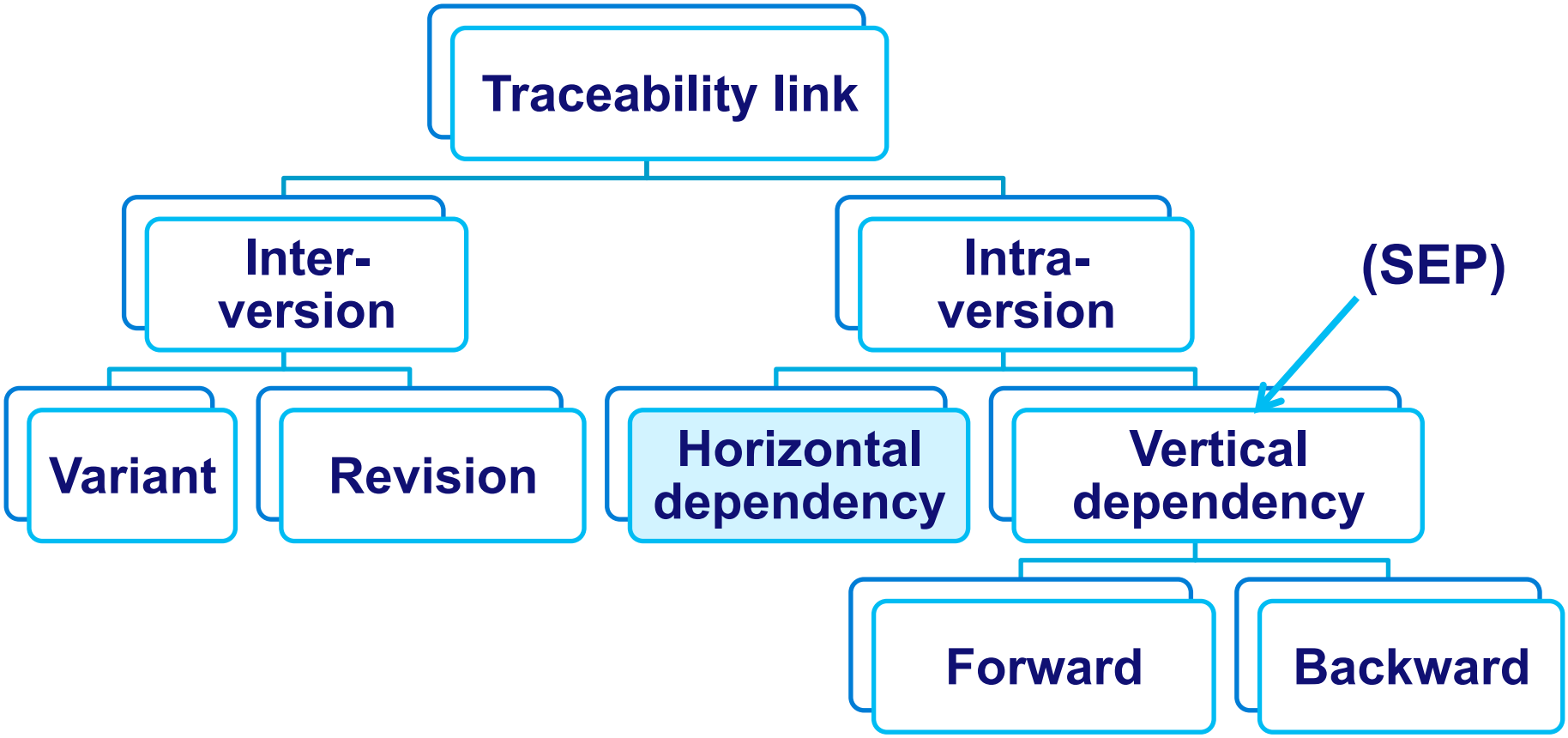
Next…

How good is requirements document v1?



Sommerville 1996

Co-evolution: Traceability links

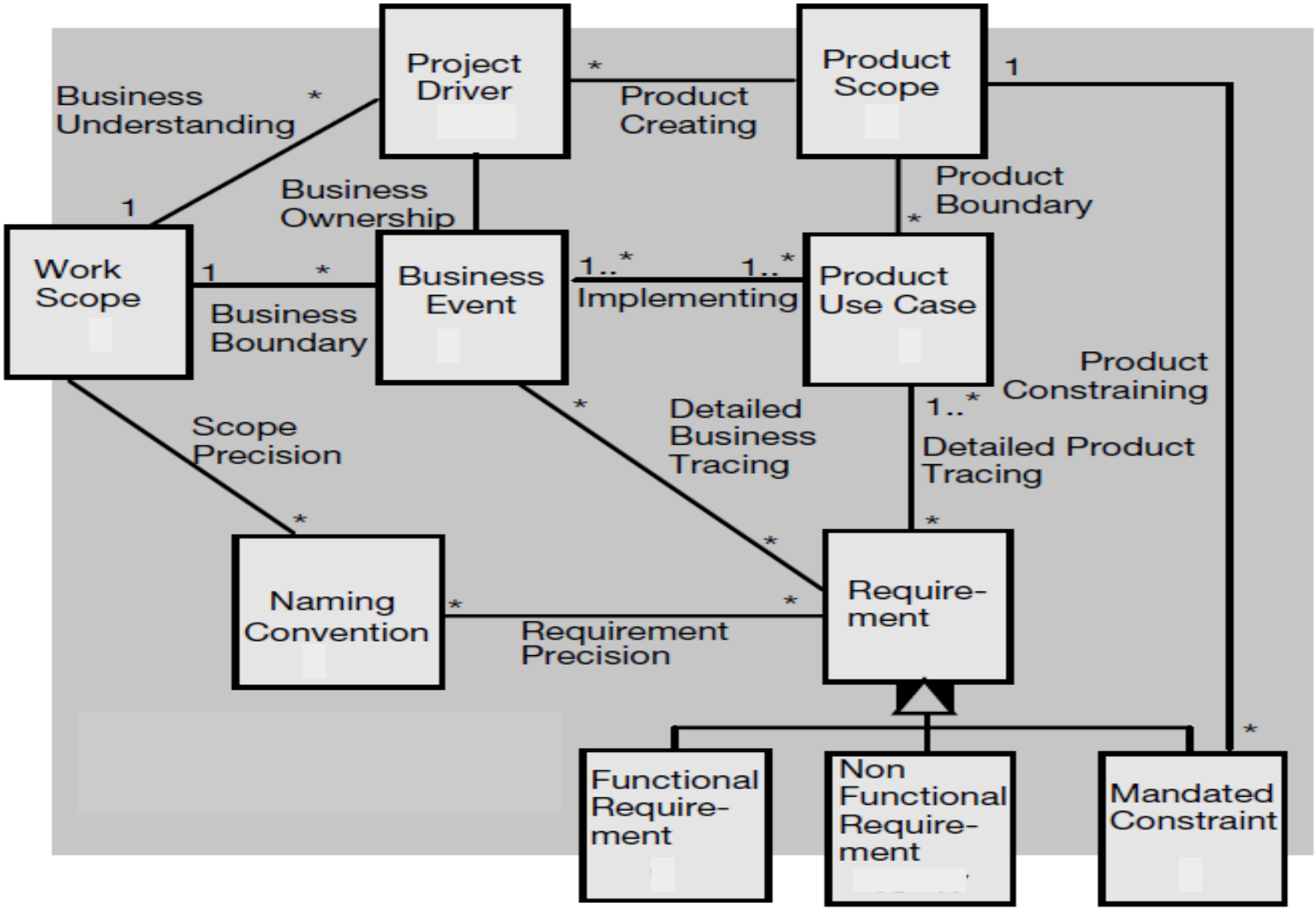


Why do we need traceability?

[Wiegers 2003]

- (Co-)evolution
  - Change impact analysis
  - Maintenance
  - Reengineering
- Certification
- Project tracking
- Reuse
- Risk reduction
- Testing

Backward dependencies [Robertson]



Traceability matrix

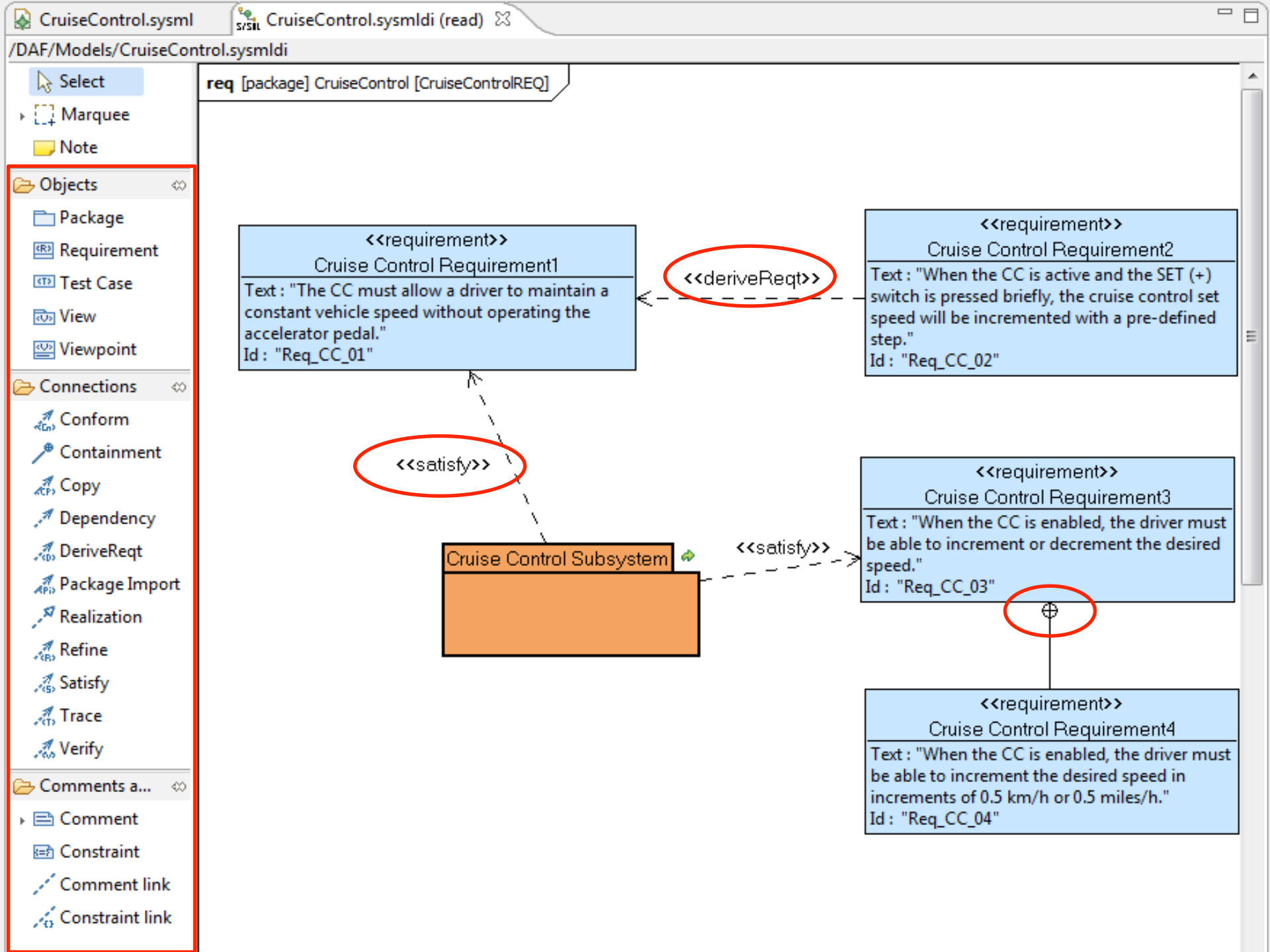
- **Means of expressing traceability information**

User req.	SW req.	Design Elem.	Func	Test Case
UC-28	SR-28, SR-15	Class catalog	sort	7, 8
UC-29	SR-44	Class catalog	import check	12, 13

Two popular techniques

User req.	Use Case		
	UC-1	UC-2	UC-3
UR-1	*		
UR-2	*		
UR-3		*	*
UR-4			*

What are their advantages and disadvantages?


CruiseControl.sysml
CruiseControl.sysmldi (read)
/DAF/Models/CruiseControl.sysmldi
Select
Marquee
Note
Objects
Package
Requirement
Test Case
View
Viewpoint
Connections
Conform
Containment
Copy
Dependency
DeriveReqt
Package Import
Realization
Refine
Satisfy
Trace
Verify
Comments a...
Comment
Constraint
Comment link
Constraint link
req [package] CruiseControl [CruiseControlREQ]
<<requirement>>
Cruise Control Requirement1
Text : "The CC must allow a driver to maintain a constant vehicle speed without operating the accelerator pedal."
Id : "Req_CC_01"
<<deriveReqt>>
<<requirement>>
Cruise Control Requirement2
Text : "When the CC is active and the SET (+) switch is pressed briefly, the cruise control set speed will be incremented with a pre-defined step."
Id : "Req_CC_02"
<<satisfy>>
Cruise Control Subsystem
<<satisfy>>
<<requirement>>
Cruise Control Requirement3
Text : "When the CC is enabled, the driver must be able to increment or decrement the desired speed."
Id : "Req_CC_03"
<<requirement>>
Cruise Control Requirement4
Text : "When the CC is enabled, the driver must be able to increment the desired speed in increments of 0.5 km/h or 0.5 miles/h."
Id : "Req_CC_04"

We need tools!

- **Large amount of information:**
  - **requirements, components, traceability links**
  - **database technology!**

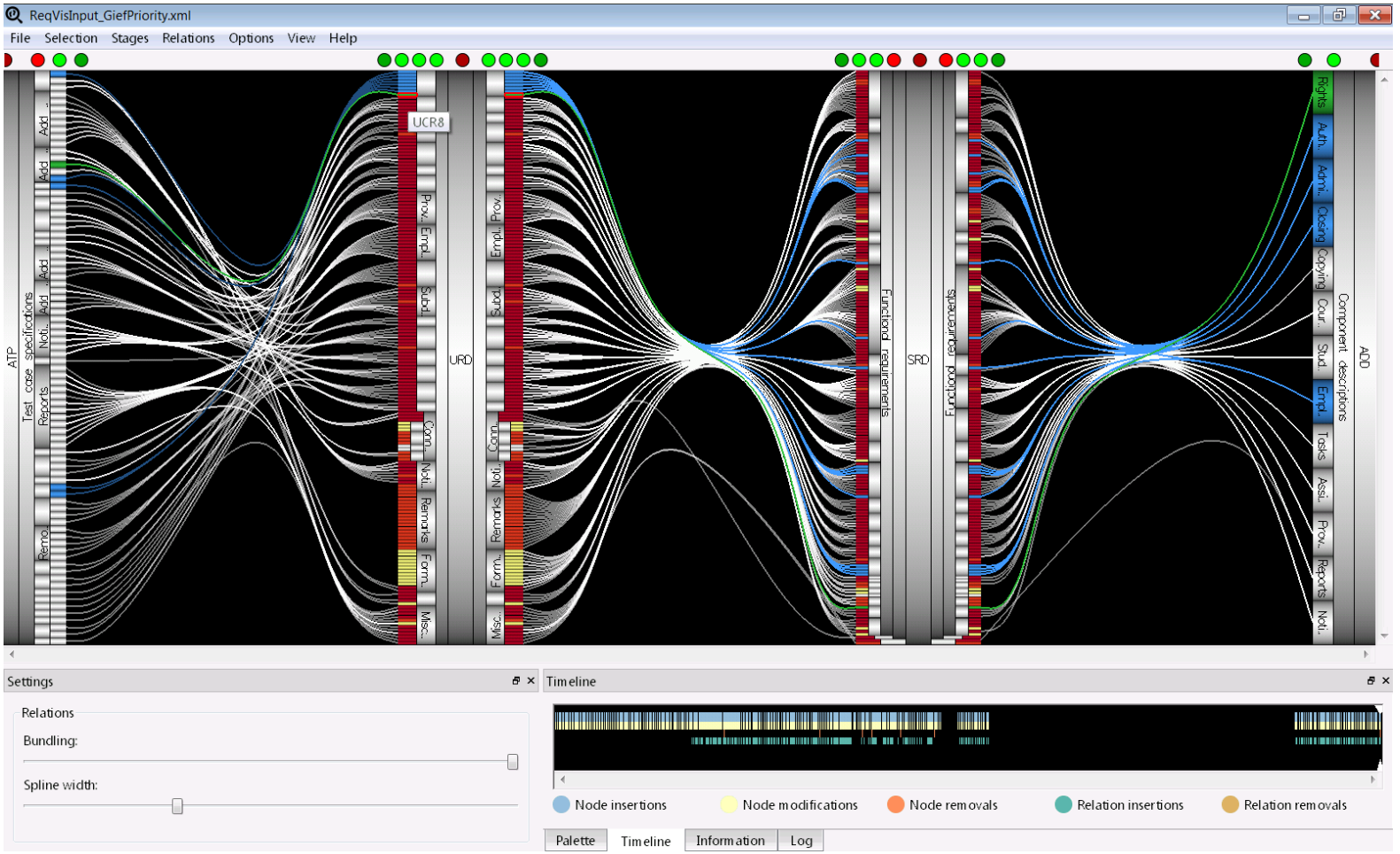
- **Many commercial/OS tools are available**
  - **OS requirements management tool: http://sourceforge.net/projects/osrmt/**
  - **http://requirements.tigris.org/**

- **You might like to try them!**

Still

- **Pro**
  - **Distinguishes different types of dependencies**
- **Contra**
  - **Does not seem to scale up**
  - **What about traceability, e.g., to tests?**
    - Goal: which requirements are covered?
    - Goal: what has to be retested if requirements are changed?

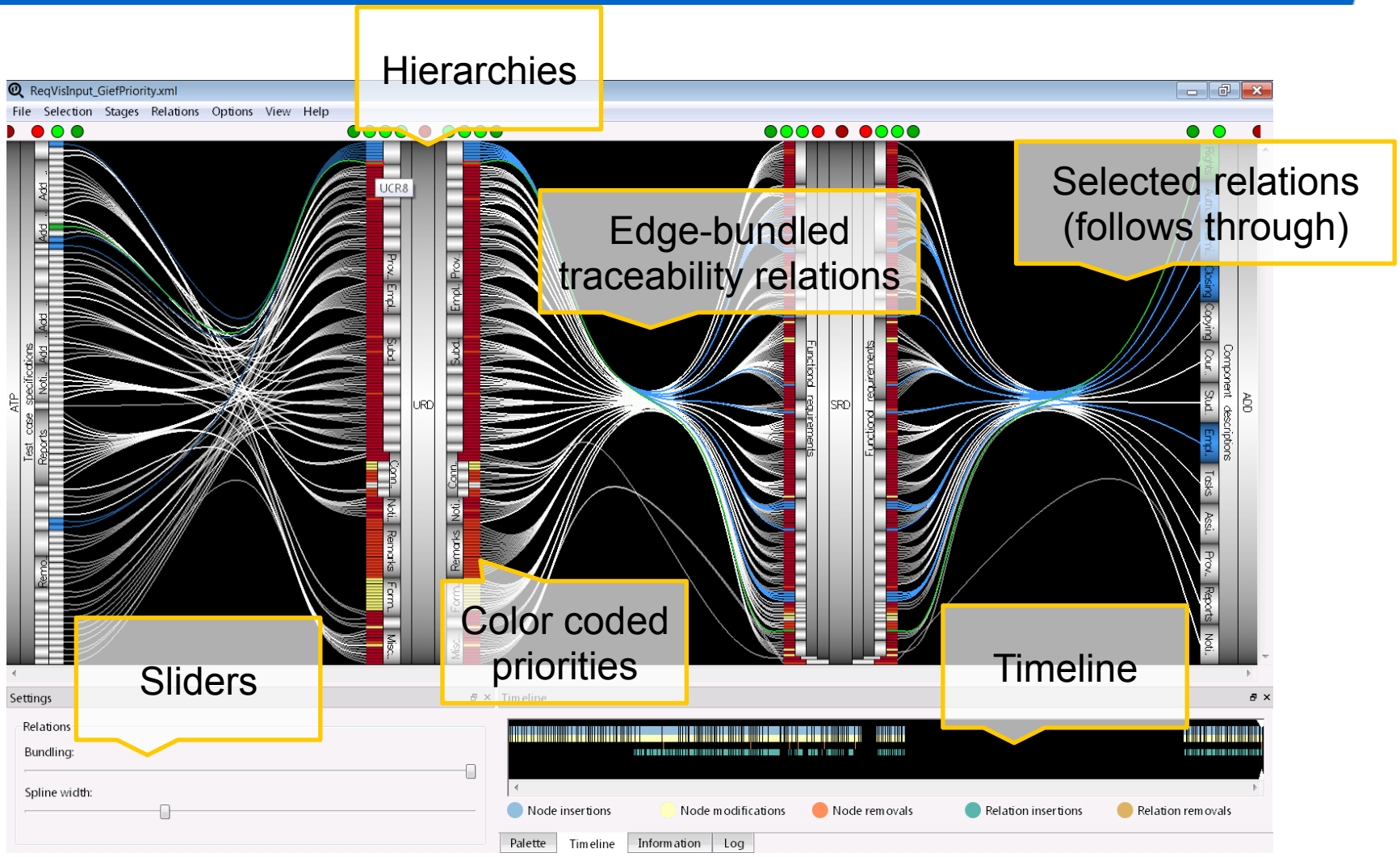
TraceVis: M.Sc. thesis of Wiljan van Ravensteijn 2011



LaQuSo
Laboratory for Quality Software

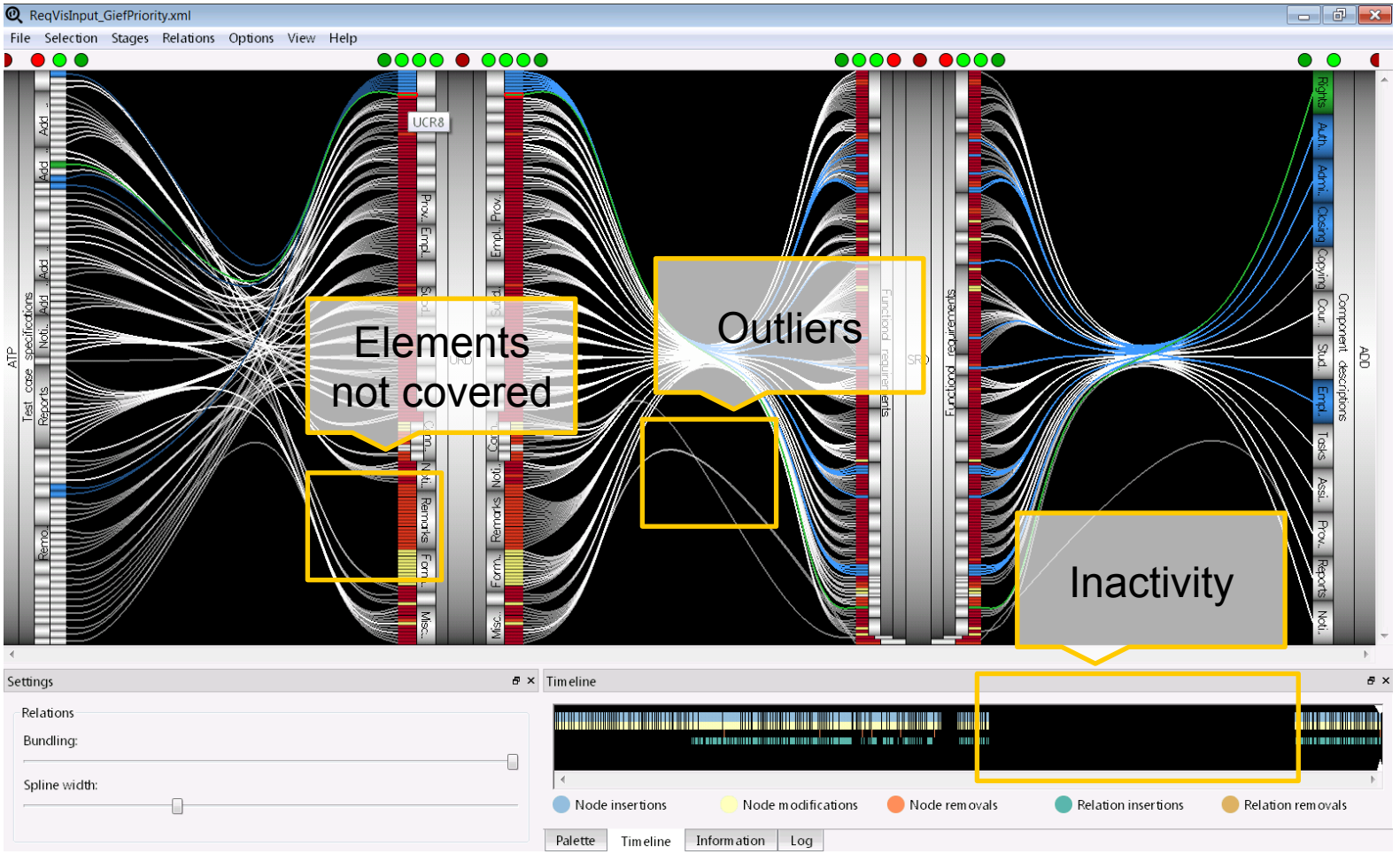
TU/e Technische Universiteit Eindhoven University of Technology

TraceVis - Overview



LaQuSo Laboratory for Quality Software

TraceVis - Patterns



LaQuSo Laboratory for Quality Software

Conclusions

- Requirements often evolve due to environmental changes
- Suitability for evolution: quality, volatility, dependencies, inconsistency
- Evolution:
  - Continuous change and growth
  - System architecture is “almost” stable
- Co-evolution
  - Need for backward and forward traceability

Table 1.3: Research opportunities in requirements evolution (**H** - High, **M** - Moderate, **L** - Low)

Research area (*Evolution and...*)	**Industry adoption**	**Research interest**	**Challenges**
Elicitation	**M** numerous approaches	**L** Most approaches focus on stable systems.	Longitudinal case studies required.
Analysis	**L** conducted by business personnel	**L** most work focuses on adaptation - what to do next.	Understanding system context.
Modeling	**L** models mostly informal	**M** numerous studies of formalization of change, e.g. [277]	Scalability; ease of use by industry.
Management	**M** most tools support some form of impact analysis, but at a simple level.	**H** Numerous frameworks and techniques.	Study mainly on greenfield systems. Little empirical validation.
Traceability	**H** widely seen as important.	**H** see traceability workshops e.g. [693]	Trace recovery; scalability
Empirical research	n/a	**M** - increasing amount of empirical validation in research papers	Reality is messy; Industry reluctance to share data

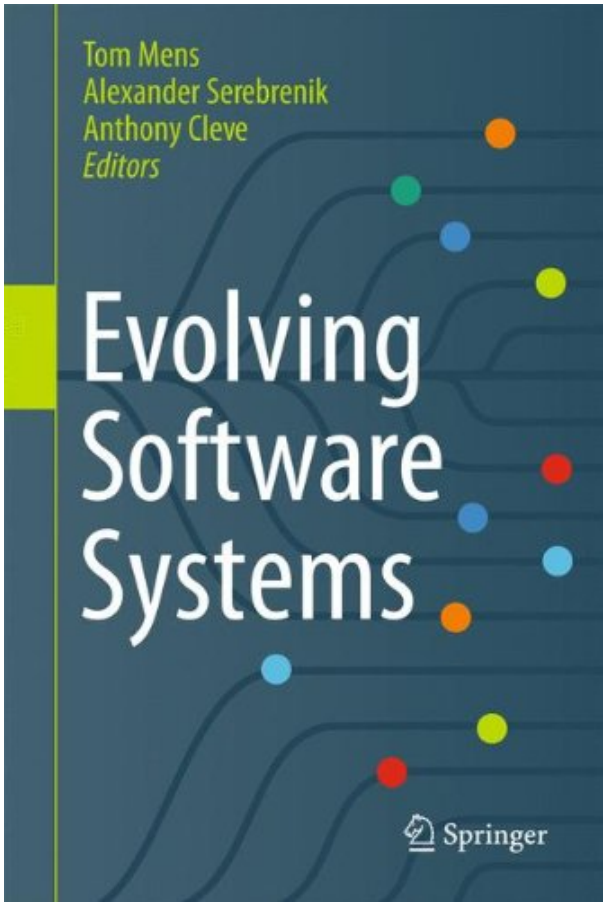
More about requirements evolutions?

- **Chapter 1**

“Requirements Evolution”

Neil Ernst, Alexander Borgida,

Ivan J. Jureta and John Mylopoulos



Assignment 1

- **Organizational:**
  - **Deadline: February 19, 23:59**
  - **PDF**
- **Assess quality of requirements, study dependencies between them and discuss the implications of your findings on the requirements evolution.**
- **Next week:**