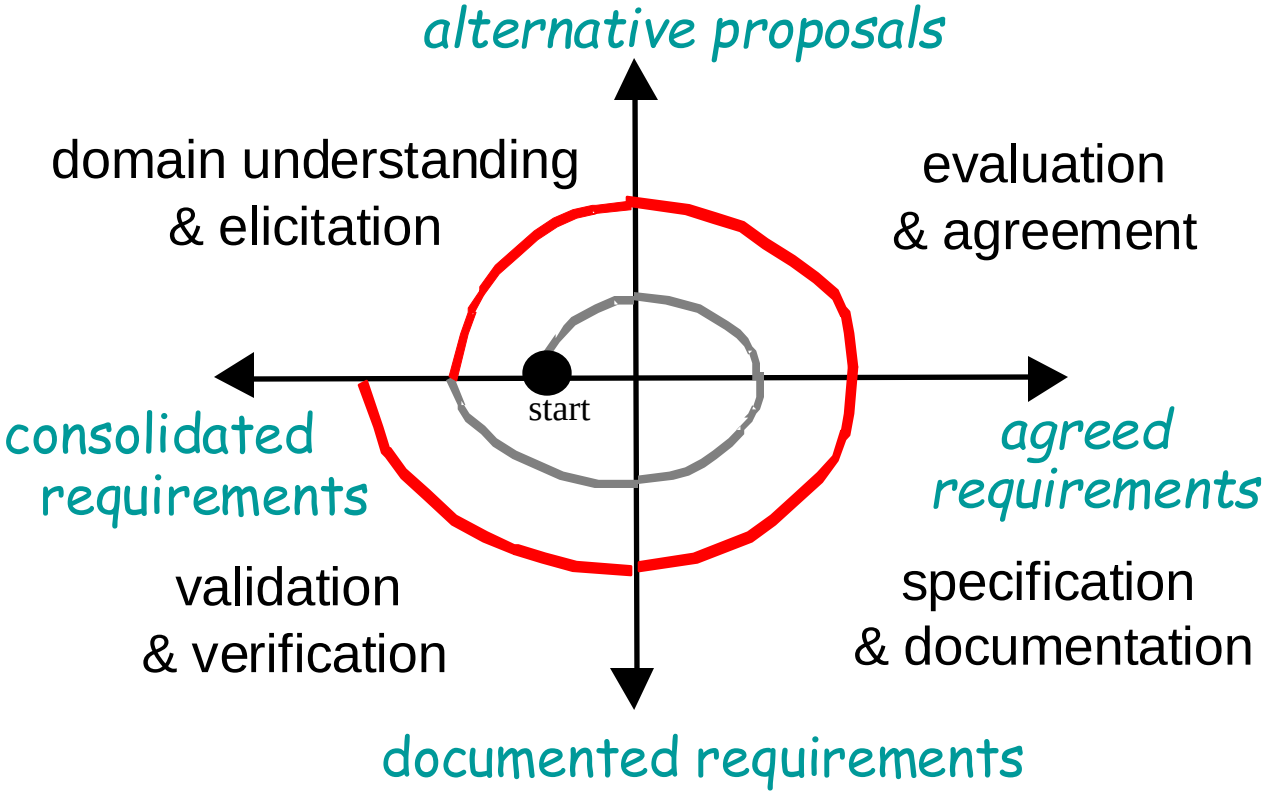
Requirements Evolution

Mariano Ceccato

mariano.ceccato@univr.it

RE products and process



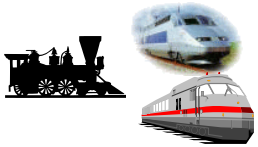
Requirements changes must be managed

- The problem world keeps changing
  - organizational changes, new regulations, new opportunities, alternative technologies, evolving priorities & constraints
  - better understanding of system features, strengths, limitations, defects
- Causes changes in objectives, concepts, reqs, assumptions
  - during RE, during software development, after deployment
- Triggers new RE cycle (elicit, evaluate, document, consolidate)
  - facilitated by change anticipation, traceability management
- Information management problem
  - consistency maintenance, change propagation, versioning
- **Requirements** (change) **management** = process of anticipating, evaluating, agreeing on, propagating changes in RD items

Outline

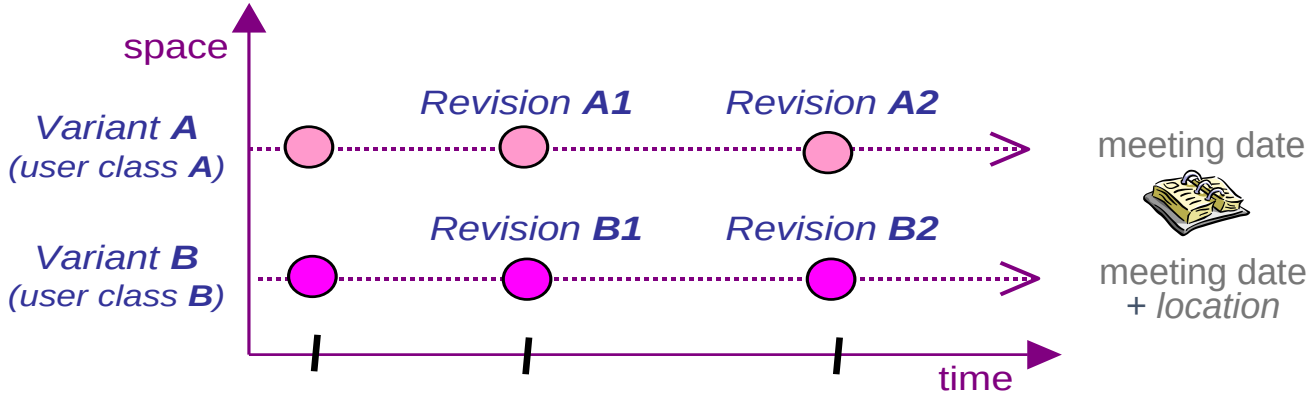
- **The time-space dimension: revisions and variants**
- **Change anticipation**
- Traceability management for evolution support
  - Traceability links
  - The TM process, benefits & cost
  - Traceability management techniques
  - Determining an adequate cost-benefit tradeoff
- Change control
  - Change initiation
  - Change evaluation & prioritization
  - Change consolidation
- Runtime spec monitoring for dynamic change management

Features, revisions, variants



- Feature = change unit
  - **functional/non-functional**: sets of functional/non-functional reqs
  - **environmental**: assumptions, constraints, work procedures, etc
- Feature changes yield new system version
  - **revision**: to correct, improve single-product version
  - **variant**: to adapt, restrict, extend multi-product version

=> commonalities + variations at variation points

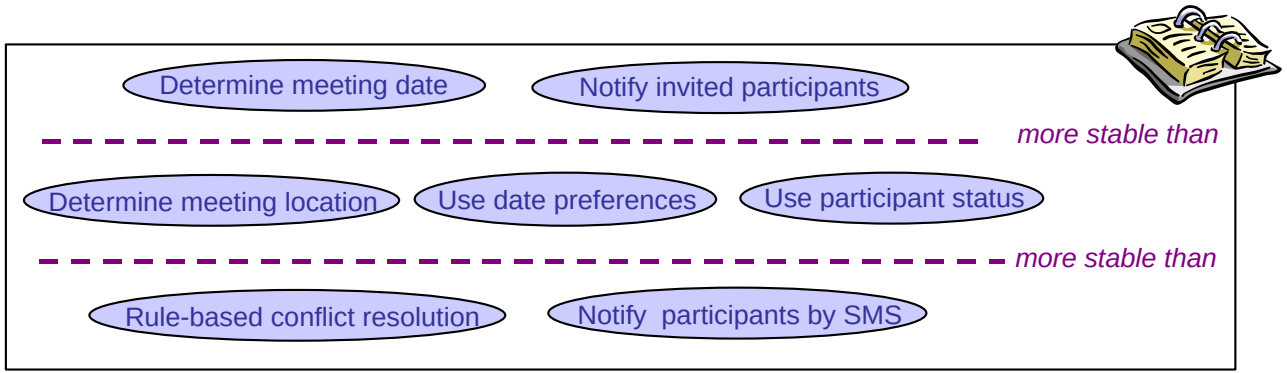


A wide variety of changes

Cause	Change type	Version type	Change time
errors & flaws	corrective	revision	RE, design, implem, post-deployment
better understanding	corrective extension	revision	RE, post-deployment
new functionality	extension	revision, variant	post-deployment
improved feature	ameliorative	revision	post-deployment
new users/usage	adaptative	variant	RE, design, post-deployment
other ways of doing	adaptative	variant	RE, post-deployment
new regulation	adaptative	revision	post-deployment
alternative regulation	adaptative	variant	post-deployment
organizational change	adaptative	revision	post-deployment
new technology	adaptative	revision	post-deployment
new priority/constraint	adaptative	revision	RE, design, implem

We must prepare for change

- Identify likely changes, assess likelihood, document them
  - to consider more stable alternatives at RE time,
    to anticipate adequate response when change will occur
  - to design architecture remaining stable in spite of changes
- By associating levels of stability (or commonality) with groups of statements defining features
  - small number of levels, each containing items of similar stability
  - qualitative & relative (for comparability)



Analyzing likely changes: heuristic rules

- Group within features cohesive sets of statements sharing same stability level and addressing same system objective
- For highest stability level: group features to be found in any contraction, extension, or variant of the system
- Intentional & conceptual aspects are often more stable than operational & factual ones
- Functional aspects meeting key objectives are more stable than non-functional ones
- Choices among alternative options are less stable
  - may be based on incomplete knowledge, volatile assumptions
  - e.g. choice among alternative ...
    - decompositions of same objective into sub-objectives
    - conflict resolutions
    - contermeasures to risks
    - responsibility assignments

Outline

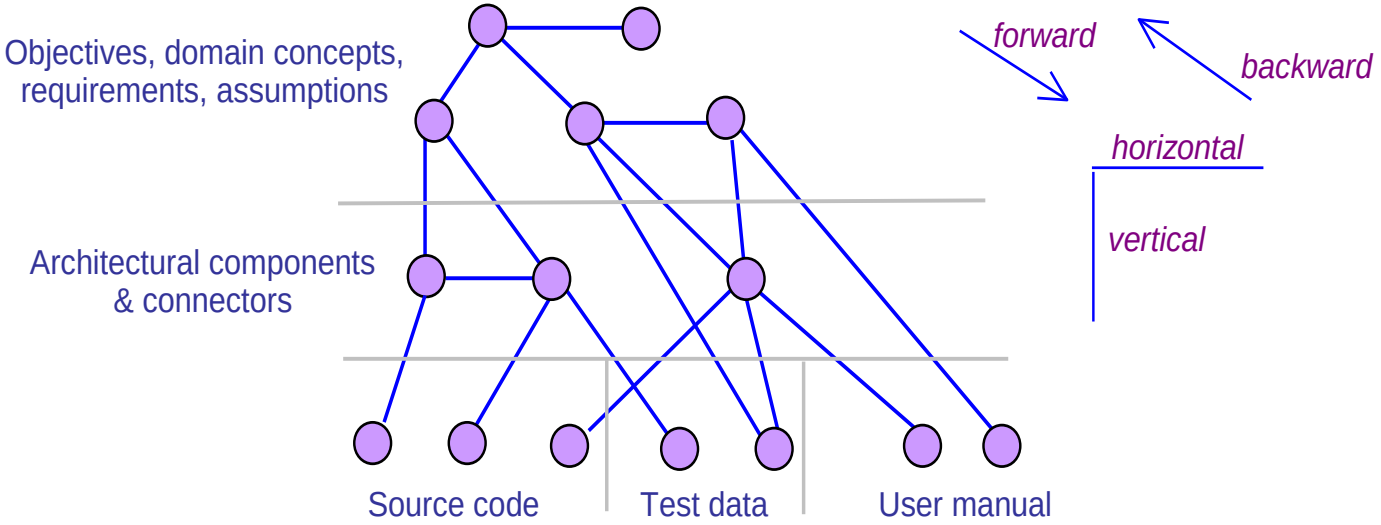
- The time-space dimension: revisions and variants
- Change anticipation
- **Traceability management for evolution support**
  - Traceability links
  - The TM process, benefits & cost
  - Traceability management techniques
  - Determining an adequate cost-benefit tradeoff
- Change control
  - Change initiation
  - Change evaluation & prioritization
  - Change consolidation
- Runtime spec monitoring for dynamic change management

Evolution support requires traceability management

- An item is **traceable** if we can fully figure out ...
  - WHERE it comes from, WHY it is there
  - WHAT it will be used for, HOW it will be used
- Traceability management (TM), roughy ...
  - identify, document, retrieve the rationale & impact of RD items
- Objectives of RE-specific traceability
  - assess impact of proposed changes
  - easily propagate changes to maintain consistency ...
    - among RD items (objectives, functional reqs, NFRs, assumptions, domain properties, concept definitions, etc)
    - between RD items & *downward* software items (design specs, architectural decisions, test data, user manuals, code, etc)

TM relies on traceability links among items

- To be identified, recorded, retrieved
- Bidirectional: for accessibility from ...
  - source to target (**forward** traceability)
  - target to source (**backward** traceability)
- Within same phase (**horizontal**) or among phases (**vertical**)

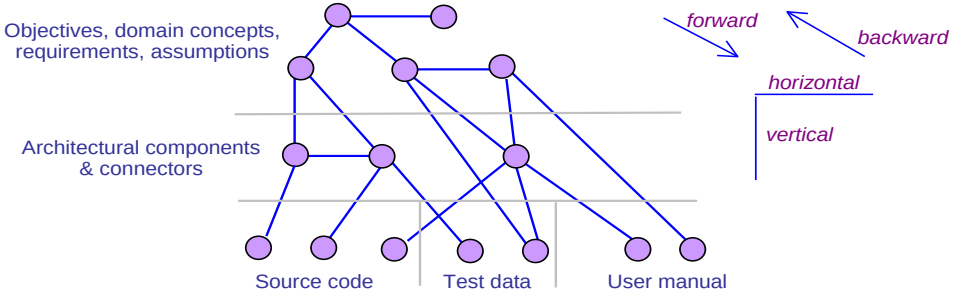


Traceability chains support multiple analyses

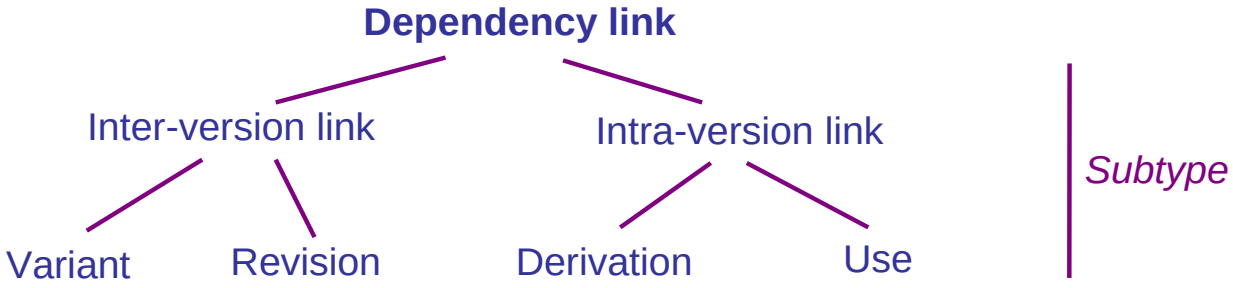
- Backward traceability
  - Why is this here? (and recursively)
  - Where does it come from? (and recursively)
- Forward traceability
  - Where is this taken into account? (and recursively)
  - What are the implications of this? (and recursively)

⇓

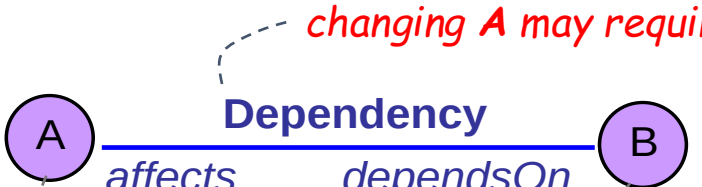
Localize & assess impact of changes along horizontal/vertical links



A taxonomy of traceability link types



More specific link type => more accurate capture & analysis



def of what participant constraints are

reqs for determining most convenient meeting date

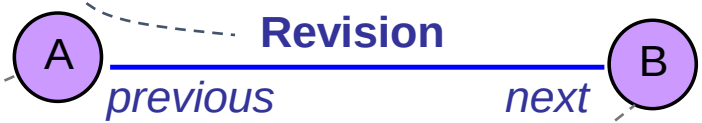
Inter-version traceability: variant, revision links

B has all features of A + specific ones

common reqs for meeting scheduling

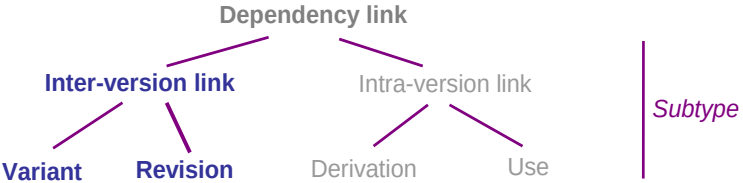
specific reqs for handling important participants

B overrides features of A, adds/removes some, keeps all others



reqs for optimal date to fit exclusion constraints

reqs for optimal date to fit exclusion & preference constraints

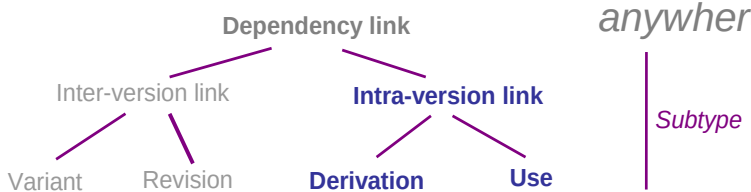


+ link annotation with configuration management info: *date, author, status, rationale*

Intra-version traceability: use, derivation links

changing A makes B incomplete, inconsistent, inadequate or ambiguous

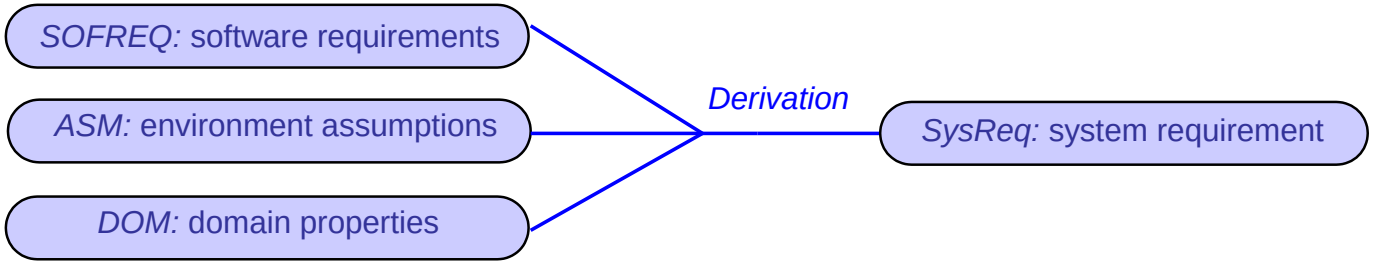
B is built from A under constraint that A must be met



Satisfaction arguments yield derivational traceability links for free

SOFREQ, ASM, DOM |= *SysReq*

"**If** the software requirements in SOFREQ, the assumptions in ASM and the domain properties in DOM are all satisfied and consistent, **then** the system requirement *SysReq* is satisfied"

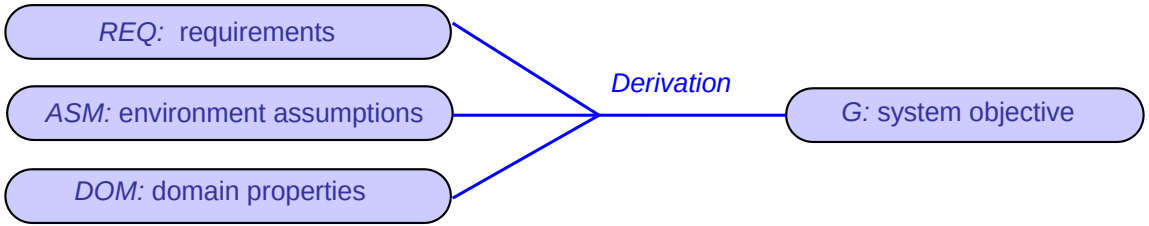


if assumption in ASM is no longer valid, reqs in SOFREQ must be reconsidered to entail satisfaction of SysReq

More accurate argument types yield more accurate derivational traceability link types

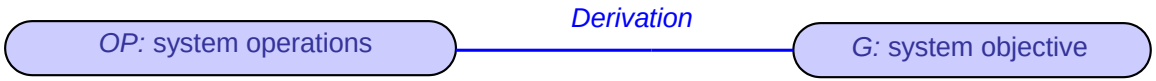
REQ, ASM, DOM |= G

"**If** the reqs in REQ, the assumptions in ASM, the domain props in DOM are all satisfied and consistent, **then** the system goal G is satisfied"

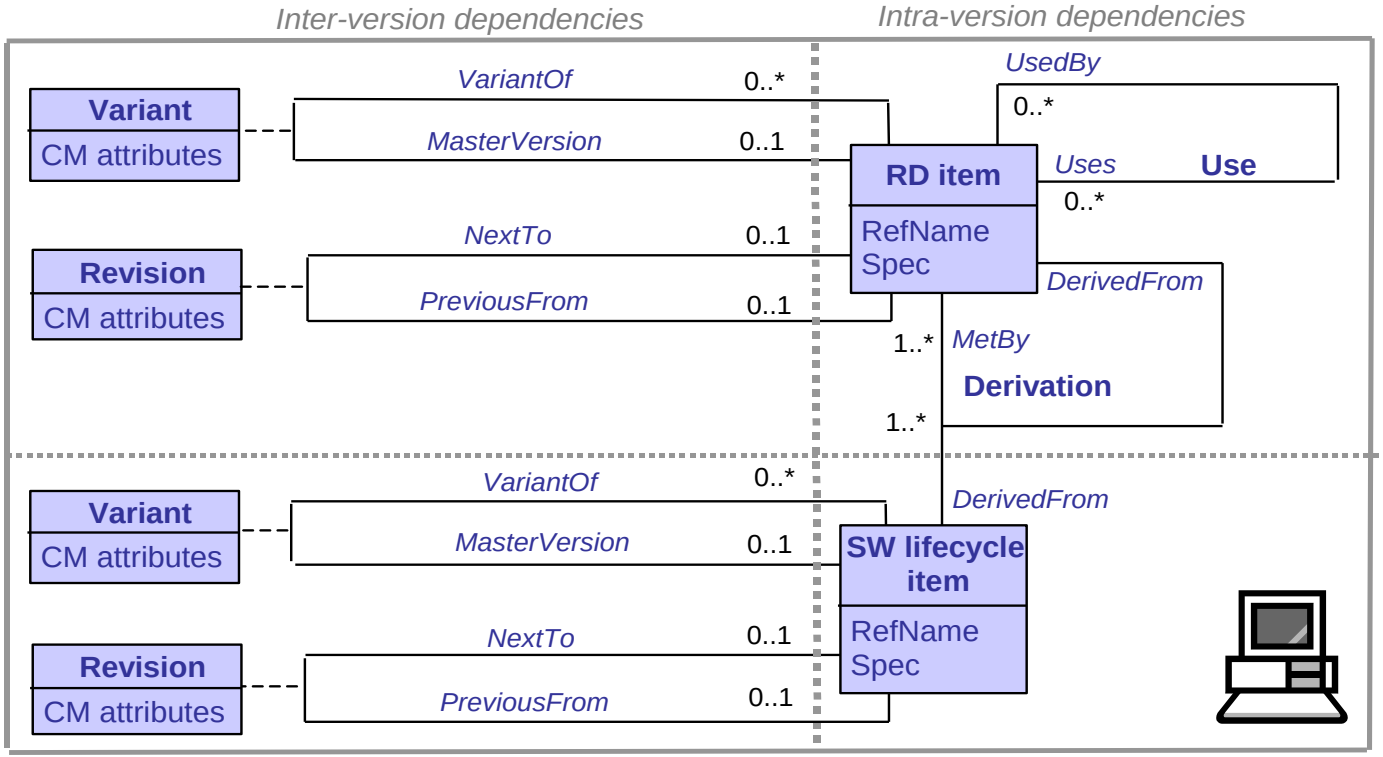


OP |= G

"**If** the operation specs in OP are satisfied and consistent, **then** the system goal G is satisfied"

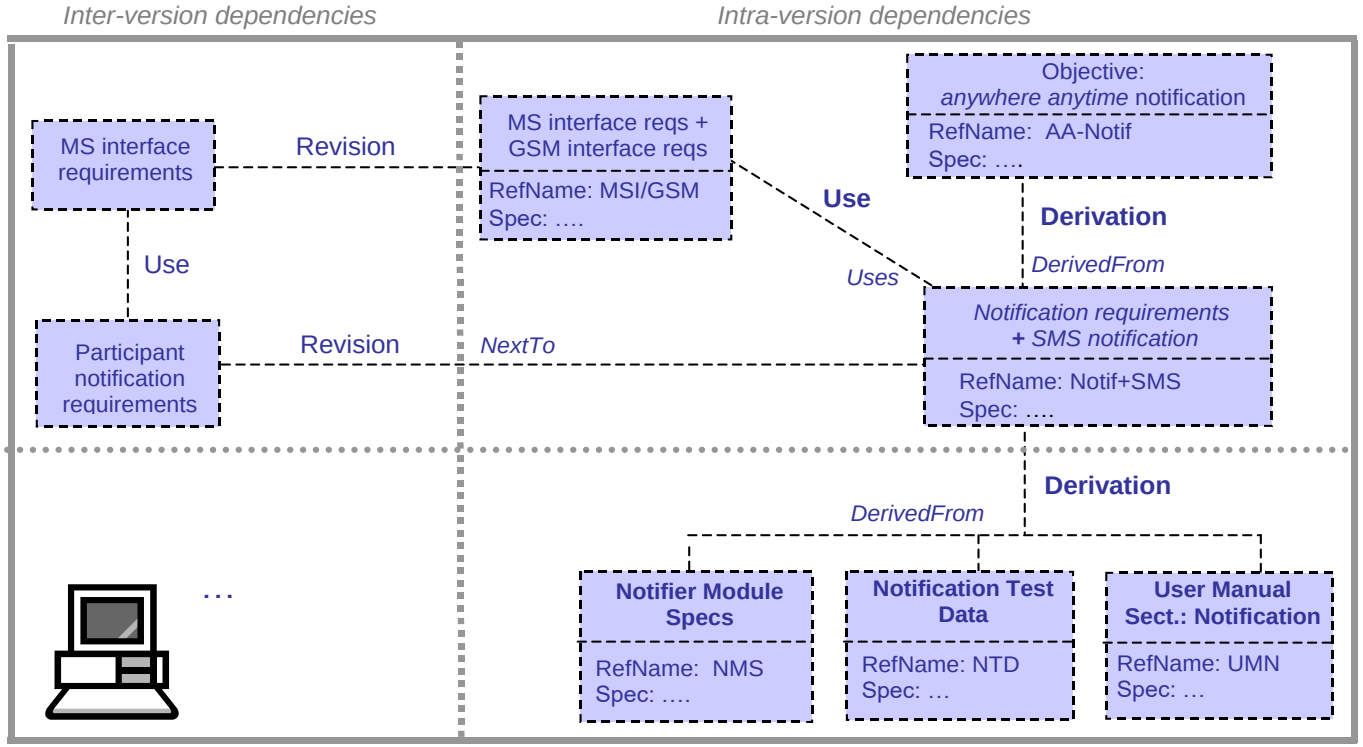


Traceability link types: an entity-relationship meta-model



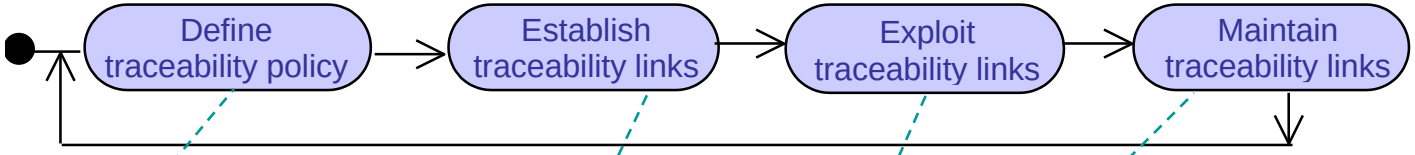
Usable as database schema for traceability database management tool

Item traceability: ER model instantiation



Traceability database analyzable through queries

The traceability management process



Optimal
cost-benefit trade-off
for guiding next steps ?

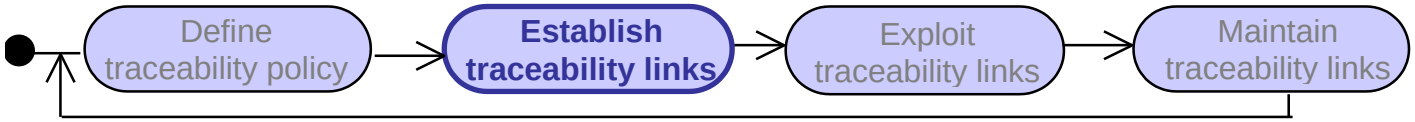
What items should be traced ?
What link types should be used?
=> traceability graph

update traceability graph
after RD changes

retrieve items along paths of traceability graph for
evolution support, rationale analysis, coverage analysis, etc.

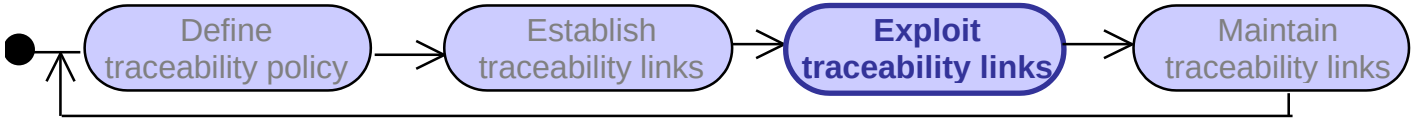
Establishing traceability links

- Four issues to consider ...
  - **granularity** of links & linked items ?
  - link **richness** -- convey semantic (*Use*, *Derivation*) vs. lexical (keywords) ?
  - link **accuracy** -- meets semantics? focussed enough for precise localizaion of dependencies?
  - **overhead** required for link management ?
- These issues interact positively, negatively
  => best compromise needed from cost-benefit assessment
- Finer-grained traceability required for mission critical features, important volatile features
- Outcome: **traceability graph**
  - nodes = traceable items, edges = links labelled by type



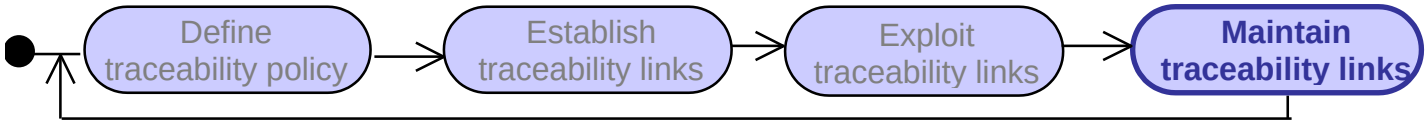
Exploiting traceability links

- For evolution support
  - To get context of requested change:
    follow *dependency* links backwards
  - To assess change impact, to propagate changes:
    follow *dependency* links forwards, horizontally & vertically
- For rationale analysis
  - To find reasons for this RD item: follow *derivation* links upwards
    - ☞ may reveal missing or irrelevant items
- For coverage analysis
  - To assess WHETHER, WHERE, HOW this RD item is used:
    follow *derivation* links downwards
- For other uses: cause-effect tracking of defects, compliance, checking against regulations, project tracking



Maintaining traceability links

- To enable link exploitation, the traceability graph must remain correct & accurate as the RD evolves
- Possible graph updates ...
  - for new item: should it get in ?
  - for modified non-traceable item: should it get in now ?
  - for modified traceable item: should its i/o links be modified ?
  - for deleted traceable item: after change propagation, remove it, remove its i/o links
- May require further graph checking, garbage collection



Outline

- The time-space dimension: revisions and variants
- Change anticipation
- Traceability management for evolution support
  - Traceability links
  - The TM process, benefits & cost
  - **Traceability management techniques**
  - **Determining an adequate cost-benefit tradeoff**
- Change control
  - Change initiation
  - Change evaluation & prioritization
  - Change consolidation
- Runtime spec monitoring for dynamic change management

Traceability management techniques: a wide palette

- Cross referencing
- Traceability matrices
- Feature diagrams (for *Variant* link type)
- Traceability databases
- Traceability model databases
- Specification-based traceability management
- Traceability link generators
- Consistency checkers

Cross referencing

- Select items to be traced, assign unique name
- Define index/tagging scheme for linking these lexically
- Configure standard search/browse engine to this scheme
- Retrieve items by following cross-reference chains

☺ lightweight, readily available

☺ any level of granularity

☹ single, semantics-free link type (lexical reference)

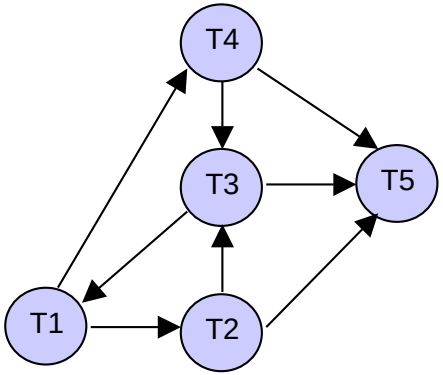
☹ hidden traceability info, cost of maintaining indexing scheme

=> limited control & analysis

Traceability matrices

- Matrix representation of single-relation traceability graph
  - e.g. *Dependency* graph

Traceable item	T1	T2	T3	T4	T5
T1	0	1	0	1	0
T2	0	0	1	0	1
T3	1	0	0	0	1
T4	0	0	1	0	1
T5	0	0	0	0	0



across T_i's row: forward retrieval of elements depending on T_i

down T_i's column: backward retrieval of elements which T_i depends on

☺ forward, backward navigation

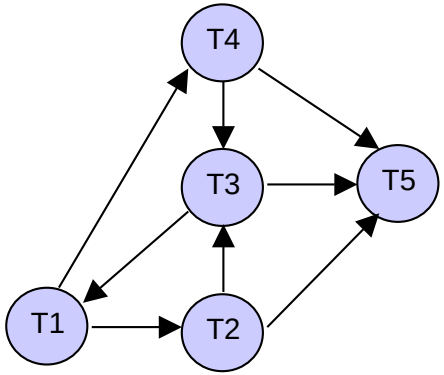
☺ simple forms of analysis e.g. cycle $T1 \rightarrow T4 \rightarrow T3 \rightarrow T1$ can be detected

☹ unmanageable, error-prone for large graphs; single relation only

Traceability lists

- Alternative representation to avoid large sparse matrices

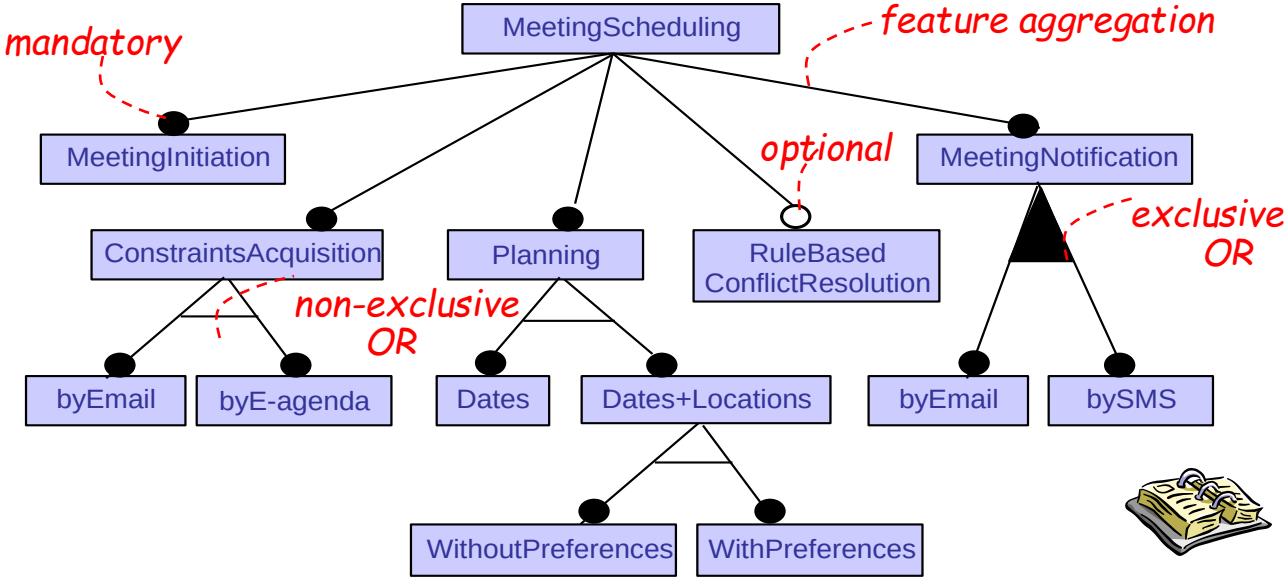
Traceable item	forwardLinkTo
T1	T2, T4
T2	T3, T5
T3	T1, T5
T4	T3, T5
T5	-



☹ Backward navigation is no longer easy

Feature diagrams

- For *Variant* link type: graphical representation of commonalities & variations in system family



☺ Compact representation of large number of variants:

1 (*MeetingInitiation*) × 3 (*ConstraintsAcquisition*) × 3 (*Planning*)
× 2 (*ConflictResolution*) × 2 (*MeetingNotification*) = 36 variants

Traceability databases

- Simple idea:
  - store traceability graph & attached info in database
  - use DBMS facilities: queries, views, versioning, …
- Traceability info is often hierarchically structured
  - project -> document -> data
- User-definable attributes can be attached at any level
  - e.g. date, author, rationale, approval status, dependency, …
- Dedicated facilities for traceability management
  - create, update, delete info units
  - historical tracking of changes
  - baselining of approved versions
  - forward/backward navigation
  - vizualisation of traceability chains, reporting, etc

Traceability databases

☺ Scaleable

☺ Generic, can be instantiated to specific needs

- e.g. multiple link types

☺ Tool support

- e.g. DOORS, RTM, RequisitePro, ...

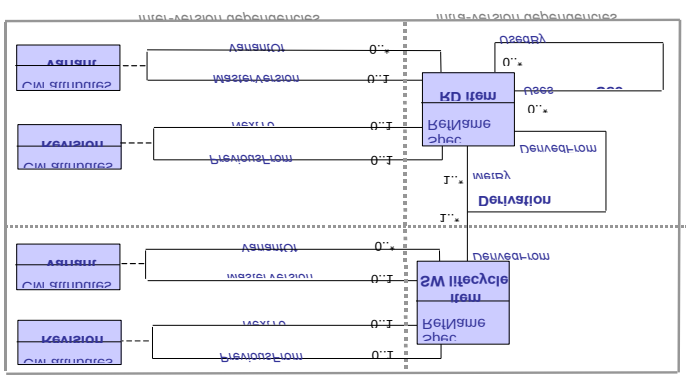
☹ Manual customization to specific needs may be difficult

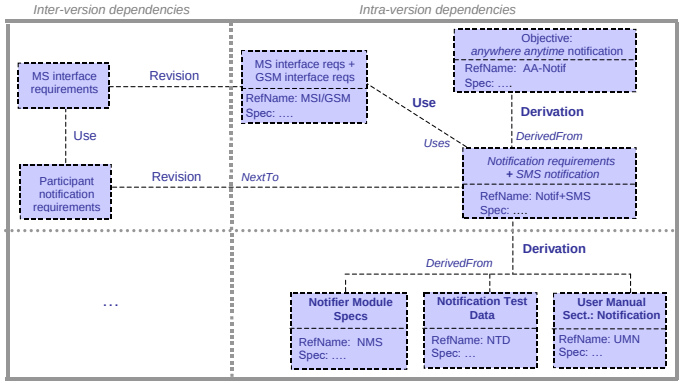
☹ Traceability information lacks structure

- flat user-defined attributes (no model)

Traceability model databases

- Model which type of traceable *item* should be related through which type of *link* (specific to organization or project)
  - entity-relationship model
  - may include process-level info --e.g. contributors, their role
- Generate tailored DBMS from model (DB schema, query system)
- Fill in, retrieve, maintain traceability model instances in DB
  - cf. queries on requirements DB for spec analysis



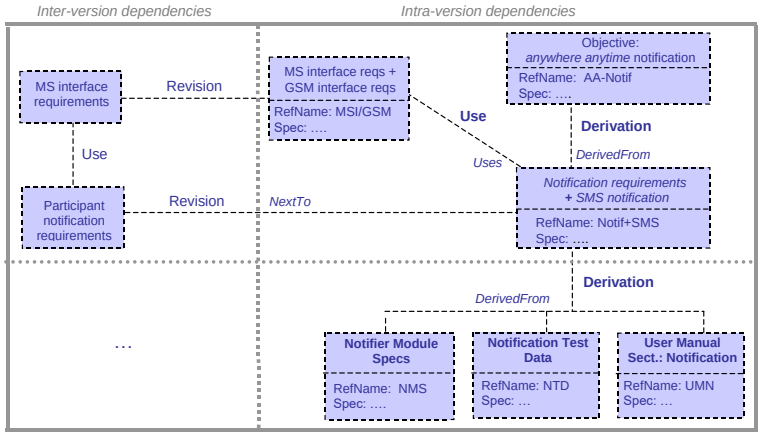


Traceability model databases

- Queries for forward/backward, horizontal/vertical navigation

set ItemsUsingGsmInterfaceReqs = **RD-Section**
which Uses (**RD-Section with RD-Section.RefName** = 'MSI/GSM')

set GsmDependentItems = **ModuleSpecs** ∪ **TestData** ∪ ...∪ **UserManualSection**
which DerivedFrom (**RD-Section** (**with RD-Section.RefName** = 'MSI/GSM'
or in ItemsUsingGsmInterfaceReqs))



- Predefined queries => dependency chains in press-button mode
- Access to executable nodes => change assessment

Traceability management techniques: a wide palette

- Cross referencing
- Traceability matrices
- Feature diagrams (for *Variant* link type)
- Traceability databases
- Traceability model databases
- Specification-based traceability management
- Traceability link generators
- Consistency checkers

Traceability link generators

- Based on information-retrieval techniques for link mining
  - a query lists keywords characterizing *source* RD item
  - *target* items = RD items whose terms match query "closely"
    => candidate source-target links
  - use of similarity measures for target-source matching
    - weights on term importance
    - thesaurus for handling synonyms

☺ Shortcut steps of establishing & maintaining traceability links

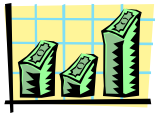
☺ Widely applicable e.g. RDs in natural language

☹ Purely lexical links

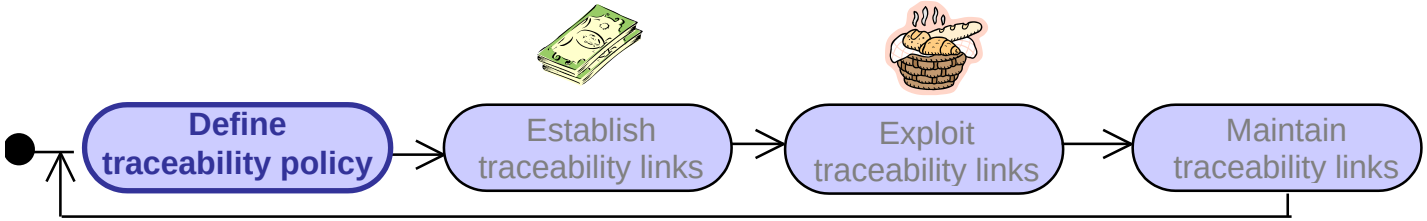
☹ Precision? Recall? => false positives, missed links

=> *Rule-based* link generation as an alternative for more semantics, better precision/recall

Determining cost-benefit trade-offs for traceability management



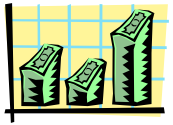
- Goal of traceability policy: achieve optimal balance between...
  - **cost** of establishing & maintaining large traceability graph
  - **benefit** from evolution support, rationale analysis, coverage analysis, compliance analysis, defect tracking, etc.
- Obstacle: delayed gratification
  - you pay now, others will benefit later
- To reach goal in spite of obstacle, a project-specific policy must effectively control *output* parameters from *input* ones



Input parameters for effective traceability policies

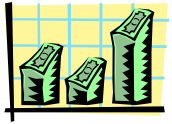
The weight on TM depends on project characteristics...

- size: the larger the project, the higher the weight
- physical distribution, turnover of development team
- timing of development phases: the tighter, the lower the weight
- estimated lifetime of software-to-be
- traceability concerns set explicitly by customers/agencies
- estimated number of requirements/assumptions to be traced
- estimated proportion of mission-critical reqs & assumptions
- estimated proportion of volatile reqs & assumptions



Output parameters for effective traceability policies

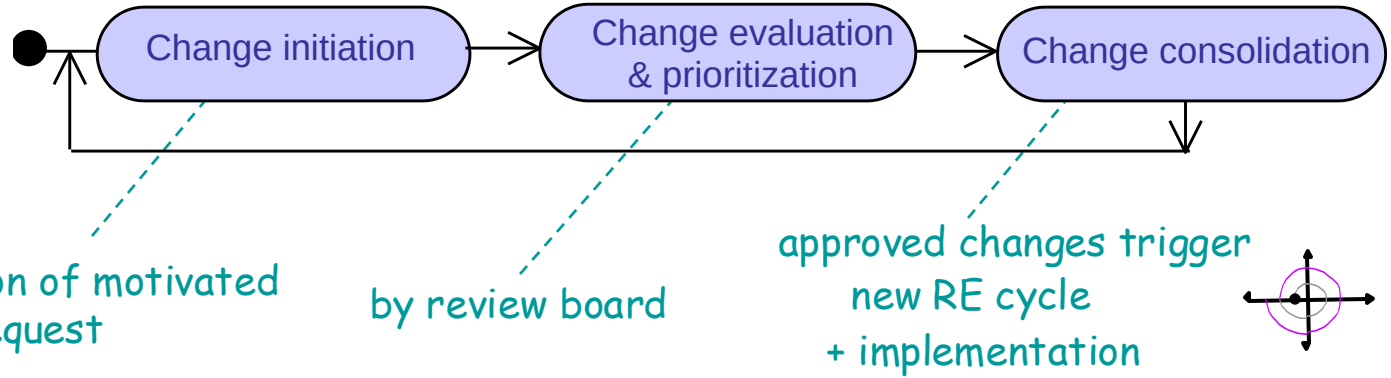
- Based on weight put on TM, a policy must determine...
  - TM scope: what should be traced, what should not?
  - link parameters (as seen before)
    - granularity
    - semantic richness
    - accuracy
    - acceptable overhead
  - TM techniques & tools to be used for decreasing cost
  - points in time where traceability graph will be established, exploited, updated
  - staff in charge of TM
- To be reevaluated throughout project



Requirements evolution: outline

- The time-space dimension: revisions and variants
- Change anticipation
- Traceability management for evolution support
  - Traceability links
  - The TM process, benefits & cost
  - Traceability management techniques
  - Determining an adequate cost-benefit tradeoff
- Change control
  - Change initiation
  - Change evaluation & prioritization
  - Change consolidation
- Runtime monitoring for dynamic change management

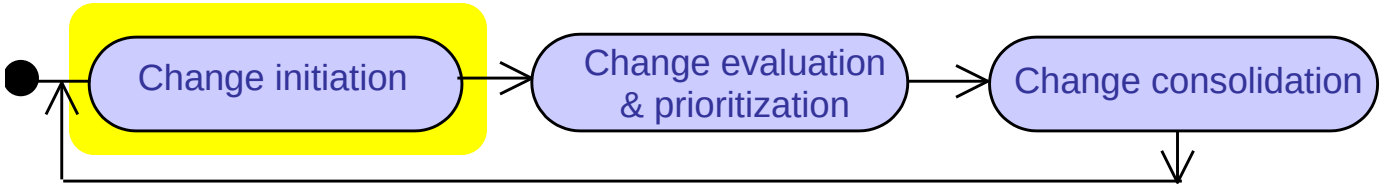
Change control



- Change anticipation & traceability management are preconditions for effective change control
- More or less formal process depending on ...
  - importance of changes
  - type of project, application domain (e.g. mission-critical projects)

Change initiation

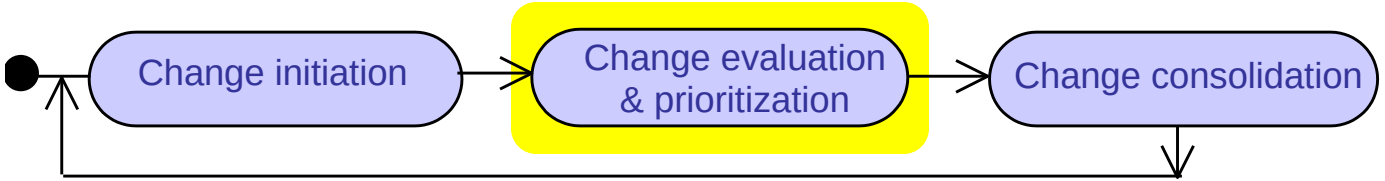
- *Wishlist* accumulates changes asked by insiders or outsiders
  - might be classified as corrective, ameliorative, adaptative
  - with perceived degree of urgency
- Periodically consolidated in motivated *change request*
  - spec of proposed changes, target items involved
  - why these items were introduced, where/how they are used
  - rationale for change
  - stakeholders asking for it
  - level of priority & urgency, why
  - estimated change impact on dependent RD/software items
  - estimated cost of changes, resources required
- On ad-hoc, project-specific form & periodicity



Change evaluation & prioritization

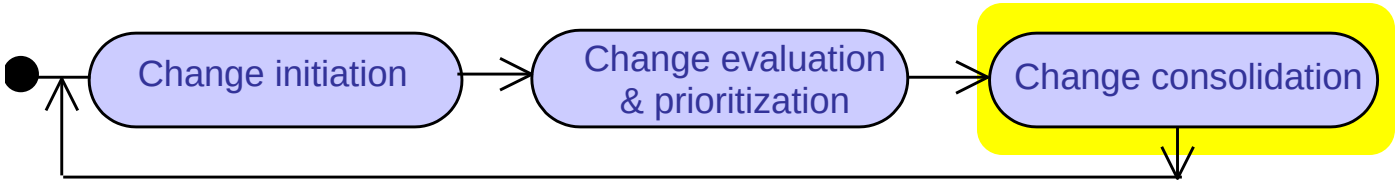
- By independent review board
  - stakeholder representatives: decision makers, domain experts, developers, marketing dept, etc.
- Value/cost assessment of requested changes
  - understand reason, check well-foundedness
  - assess benefits, risks
  - assess impact of change, feasibility, cost

  ☞ Cf. requirements evaluation techniques
- Outcome: agreed status for each requested change
  - approve, ignore, amend, defer
  - to be documented for next board meetings



Change consolidation

- Perform all approved changes => new system version
  - forward propagation of changes through RD along horizontal links of traceability graph (new RE cycle in spiral process)
  - baselining of new RD version
  - forward propagation through software items along vertical links of traceability graph
  - update traceability graph
- Tool support for change control
  - collaborative tools, database tools for managing change requests
  - traceability management tools
  - version control tools

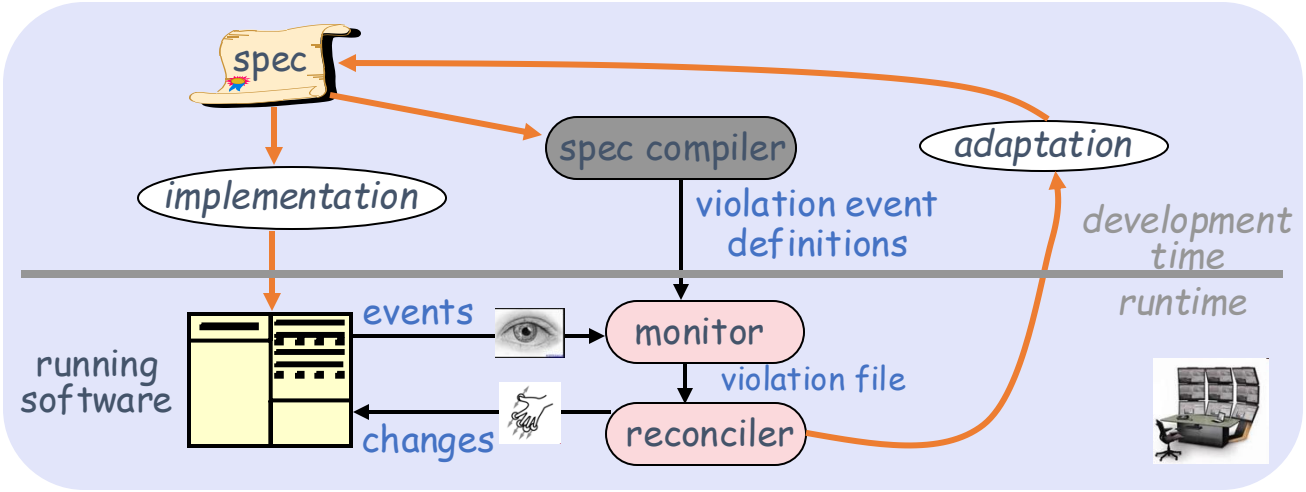


Outline

- The time-space dimension: revisions and variants
- Change anticipation
- Traceability management for evolution support
  - Traceability links
  - The TM process, benefits & cost
  - Traceability management techniques
  - Determining an adequate cost-benefit tradeoff
- Change control
  - Change initiation
  - Change evaluation & prioritization
  - Change consolidation
- Runtime spec monitoring for dynamic change management

Runtime spec monitoring for dyn. adaptation

- Motivation
  - assumptions underlying reqs might be no longer valid at runtime
  - system might be running under new/other conditions
  - variations anticipated at RE time might be too costly to handle
- General architecture for self-adapting system:



Runtime spec monitoring for dyn. adaptation

- At RE time ...
  - identify specs to be monitored
    - requirements, assumptions
    - volatile, unrealistic in some cases, mission-critical, ...
  - for each such spec: derive dedicated spec monitor
    - can be automatically generated if formal spec
- At development time ...
  - design, implement software according to specs
  - design, implement alternative routes for excessive spec violation
- At system runtime ...
  - **monitor** tracks system events, generates warnings if spec violation, and triggers system reconfiguration if too frequent violations
  - **reconciler** forces shifting to alternative route when triggered
    - rule-based system reconfiguration

Dynamic evolution: example

1. Identify unstable or sometimes unrealistic assumption

   Participant receiving email request for constraints will provide them within X days

2. Derive monitor detecting no-reply events within deadline

3. Design, implement alternative route in case of spec violation

   Send reminder to participant ? Send request to secretary ?

   Issue warning to initiator ? etc

development time

runtime

4. Monitor violation events during system execution and trigger reconciler according to reconfiguration rules

   If # times participant did not return constraints within *X* days, last *Y* months, **> *Z***

   then send reminder to alternative contact person *P*

   If # times participant did not return constraints within *X* days, last *Y* months, **> *Z***

   then get participant's constraints from e-agenda

5. Shift to alternative route for *this* participant

Requirements evolution, summary: prepare for change

- Requirements consistency, completeness, adequacy must be preserved through inevitably evolving problem world
  - wide variety of corrective, ameliorative, adaptative changes
- We must anticipate changes, specify levels of stability
- Traceability management is another precondition for evolution
  - Variety of traceability link types specializing dependencies
    - multi-directional: backward, forward, horizontal, vertical
  - Issues: link granularity, semantic richness, accuracy, overhead
  - Variety of supporting techniques
  - Cost-benefit tradeoff to be determined
- Change control iterates on initiation, evaluation-prioritization, consolidation cycles
  - more or less formal process depending on project type
- Specs can be monitored for on-the-fly change management in self-adapting system