The Triptych Process Model [1]
Process Assessment and Improvement

Dines Bjørner

Computer Science and Engineering	Graduate School of Information Science
Informatics and Mathematical Modelling	Japan Adv. Inst. of Science & Technology
Technical University of Denmark	1-1, Asahidai, Tatsunokuchi
DK-28000 Kgs.Lyngby	Nomi, Ishikawa 923-1292
Denmark	Japan

bjorner@gmail.com

June 14, 2006. Compiled October 2, 2006

[1] Invited keynote talk for the JASPIC (Japan Software Process Improvement Consortium) Conference, 12–13 October 2006, Tsukuba, Japan

- The triptych approach to software engineering proceeds on the basis of carefully monitored and controlled possibly iterated progression through
  - ⋆ domain engineering and
  - ⋆ requirements engineering to
  - ⋆ software design.
- In this talk I will outline
  - ⋆ these three phases,
  - ⋆ show the many stages of development within each and also
  - ⋆ indicate the many steps within each stage.

The Triptych Dogma
Background

- ⋆ In the past, as exemplified in major software engineering textbooks,
  - ◇ software engineering focused on requirements engineering and software design.
- ⋆ The triptych dogma
  - ◇ extends the two (requirements engineering and software design)
  - ◇ into three (domain engineering plus the two phases already mentioned).

The Dogma

- ⋆ Justifying requirements prescriptions:
  - ◇ Before software can be designed
  - ◇ we must understand the requirements.
- ⋆ Justifying domain descriptions.
  - ◇ Before requirements can be prescribed
  - ◇ we must understand the domain.
- ⋆ Justifying the triptych:
  - ◇ First analysing and describing the (application) domain,
  - ◇ then analysing and prescribing the requirements, and
  - ◇ finally analysing and specifying the software design and code.

New Aspects

⋆ The relatively new aspect of software development is here 'domain engineering'.

⋆ This new aspect "translates" into a number of new methodological aspects of domain and requirements engineering.

⋆ The next, the major section will survey these aspects.

⋆ All of this is covered extensively in volume 3 of the three volume book Software Engineering, published early this year by Springer

⋆ Go buy it! Now!.

The Triptych Process Models and Documents
Common Aspects
Process Models

⋆ The triptych process model is the composition of three process models: one each for

◇ domain engineering,

◇ requirements engineering and

◇ software design.

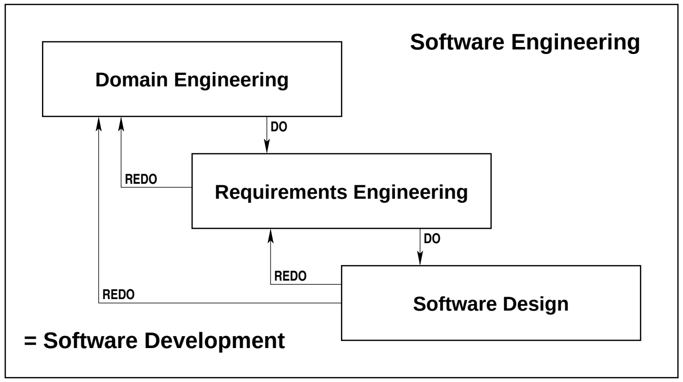

Figure 1: A simplified view of the triptych process model

Documents

⋆ Common to all three phases of software development are that they primarily manifest themselves in documents.

⋆ Figures to be commented later, illustrate the breadth, depth and quite substantial number of such resulting documents.

⋆ And common to each set of such documents is the more-or-less administrative “working out” of *information document*:

1. Information
   (a) Name, Place and Date
   (b) Partners
   (c) Current Situation
   (d) Needs and Ideas
   (e) Concepts and Facilities
   (f) Scope and Span
   (g) Assumptions and Dependencies
   (h) Implicit/Derivative Goals
   (i) Synopsis
   (j) Standards Compliance
   (k) Contracts
   (l) The Teams
      i. Management
      ii. Developers
      iii. Client Staff
      iv. Consultants
   (m) Plans
      i. Project Graph
      ii. Budget
      iii. Funding
      iv. Accounts
   (n) Management
      i. Assessement
      ii. Improvement
         A. Plans
         B. Actions

Figure 2: Informative documents

The Domain Engineering Process Model
Domain Models

- ⋆ A main result of domain engineering development, as applied to some specific application domain, is a domain model.
- ⋆ Domain models are in the form of descriptions.
- ⋆ Domain descriptions describe what there is, and as it is.
- ⋆ There is no presumption of requirements implied by these descriptions.
- ⋆ They are not requirements prescriptions.
- ⋆ By analogy,
  - ⋄ physicists [domain engineers] are describing mother nature [application domains]
  - ⋄ and engineers [requirements engineers and software designers] are prescribing and implementing requirements.

Domain Engineering, A Narrative

⋆ The domain engineering triptych dogma, advocates

⋆ (item 2.) the following stages of description development (after work on information documents [items 1.a–l] have been duly completed):

- ◇ (2.a) identification of as wide a spectrum of domain stakeholders,
- ◇ (2.b) acquisition of domain understanding,
- ◇ (2.c) establishment (and subsequent, throughout all stages, use and maintenance) of a domain terminology (ontological terms),
- ◇ (2.d) understanding and rough-sketching all relevant business processes,
- ◇ (2.e) domain modelling (all domain facets), and
- ◇ (2.f) the domain model completion (including consolidation).

⋆ Intertwined with the domain description parts (item 2., subitems (a–f)) are the analysis parts with

◇ (3.a) analysis aiming at identifying inconsistencies, conflicts and incompletenesses,

◇ (3.b) domain validation,

◇ (3.c) domain verification,and

◇ (3.d) possible work on establishing a domain theory.

Domain Engineering Documents

2. Descriptions
   (a) Stakeholders
   (b) The Acquisition Process
      i. Studies
      ii. Interviews
      iii. Questionnaires
      iv. Indexed Description Units
   (c) Terminology
   (d) Business Processes
   (e) Facets:
      i. Intrinsics
      ii. Support Technologies
      iii. Management and Organisation
      iv. Rules and Regulations
      v. Scripts
      vi. Human Behaviour
   (f) Consolidated Description
3. Analyses
   (a) Domain Analysis and Concept Formation
      i. Inconsistencies
      ii. Conflicts
      iii. Incompletenesses
      iv. Resolutions
   (b) Domain Validation
      i. Stakeholder Walkthroughs
      ii. Resolutions
   (c) Domain Verification
      i. Model Checkings
      ii. Theorems and Proofs
      iii. Test Cases and Tests
   (d) (Towards a) Domain Theory

Figure 3: Domain engineering document table-of-contents

Domain Engineering Stages and Steps

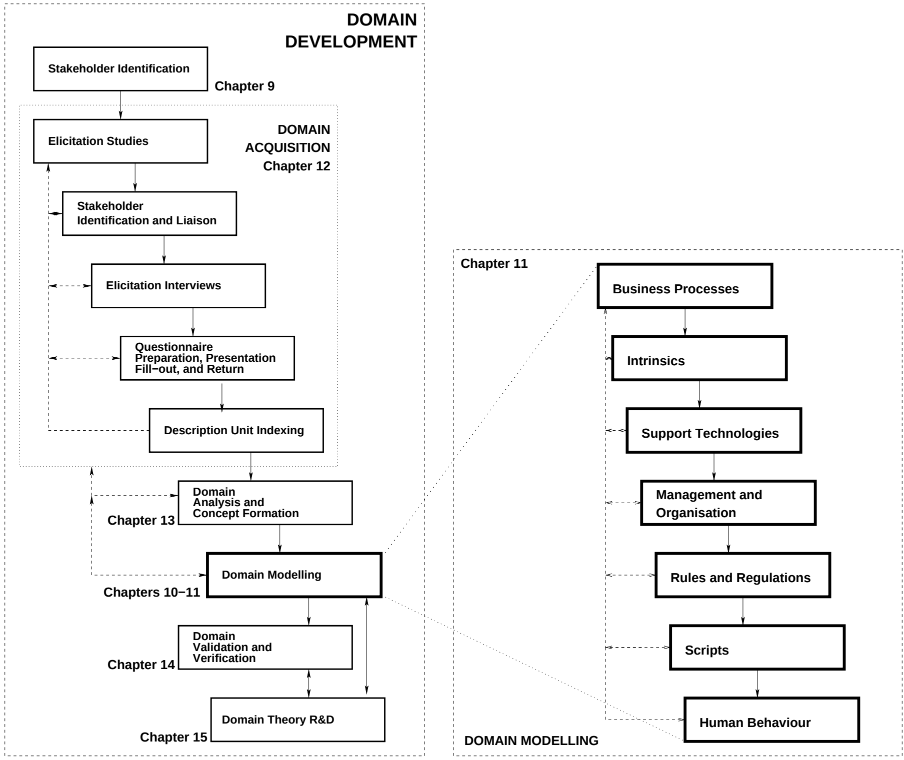

Figure 4: The domain engineering process model diagram

Figure 4 diagrams, in box-and-edge form, the stages and steps of domain engineering development and their interrelations.

The Requirements Engineering Process Model
The Machine

- ⋆ Requirements is about prescribing the machine:
  - ◇ the hardware and
  - ◇ the software

  which shall implement the requirements.
- ⋆ The machine resides in the domain.
- ⋆ Once developed we shall
  - ◇ sometimes refer to that domain as the environment of the machine —
  - ◇ with the machine + that environment becoming a new domain.

Requirements Models

- ⋆ A main result of requirements engineering development,
  - ◇ as applied to some specific application domain[2],
  - ◇ is a requirements model.
- ⋆ Domain models are in the form of descriptions.
- ⋆ Requirements prescriptions prescribe what there should be.

[2] Examples of domains are: (1) the financial service industry as a whole, (1.1) a bank, (1.1.1) a bank's mortgage lending business; (2) the transportation industry as a whole, (2.1) a railway system, (2.1.1) an interlocking system; etcetera.

Requirements Engineering, A Narrative

The requirements engineering triptych dogma advocates

- ⋆ (item 2.) the following stages of prescription development (after work on information documents [items 1.a–l] have been duly completed):
  - ◇ (2.a) identification of as wide a spectrum of requirements stakeholders,
  - ◇ (2.b) acquisition of requirements statements,
  - ◇ (2.c) rough-sketching first ideas of a requirements model in order to ("eureka") discover un-formulated requirements,
  - ◇ (2.d) establishment (and subsequent, throughout all stages, use and maintenance) of a requirements terminology (ontological terms),

and

- ◇ (2.e) requirements modelling of all requirements facets:
  - ∘ (2.e.i) business process reengineering (BPR),
  - ∘ (2.e.ii) domain requirements,
  - ∘ (2.e.iii) interface requirements,
  - ∘ (2.e.iv) machine requirements, and
  - ∘ (2.e.iv) completion of a full requirements prescription.

⋆ Intertwined with the requirements prescription parts (item 2., subitems (a–e)) are the analysis parts with

- ◇ (3.a) analysis aiming at identifying inconsistencies, conflicts and incompletenesses,
- ◇ (3.b) requirements validation,
- ◇ (3.c) requirements verification, and
- ◇ (3.d) possible work on establishing a requirements theory.

Requirements Engineering Documents

2. Prescriptions
   (a) Stakeholders
   (b) The Acquisition Process
      i. Studies
      ii. Interviews
      iii. Questionnaires
      iv. Indexed Description Units
   (c) Rough Sketches (Eurekas, IV)
   (d) Terminology
   (e) Facets:
      i. Business Process Re-engineering
         - Sanctity of the Intrinsics
         - Support Technology
         - Management and Organisation
         - Rules and Regulation
         - Human Behaviour
         - Scripting
      ii. Domain Requirements
         - Projection
         - Determination
         - Instantiation
         - Extension
         - Fitting
      iii. Interface Requirements
         - Shared Phenomena and Concept Identification
         - Shared Data Initialisation
         - Shared Data Refreshment
         - Man-Machine Dialogue
         - Physiological Interface
         - Machine-Machine Dialogue

2. (Prescriptions, continued)
   (e) (Facets, continued)
      iv. Machine Requirements
         - Performance
            ⋆ Storage
            ⋆ Time
            ⋆ Software Size
         - Dependability
            ⋆ Accessability
            ⋆ Availability
            ⋆ Reliability
            ⋆ Robustness
            ⋆ Safety
            ⋆ Security
         - Maintenance
            ⋆ Adaptive
            ⋆ Corrective
            ⋆ Perfective
            ⋆ Preventive
         - Platform
            ⋆ Development Platform
            ⋆ Demonstration Platform
            ⋆ Execution Platform
            ⋆ Maintenance Platform
         - Documentation Requirements
         - Other Requirements
      v. Full Reqs. Facets Doc.

Figure 5: Requirements engineering document table-of-contents: prescription documents

3. Analyses
   (a) Requirements Analysis and Concept Formation
      i. Inconsistencies
      ii. Conflicts
      iii. Incompletenesses
      iv. Resolutions
   (b) Requirements Validation
      i. Stakeholder Walk-through and Reports
      ii. Resolutions
   (c) Requirements Verification
      i. Model Checkings
      ii. Theorem Proofs
      iii. Test Cases and Tests
   (d) Requirements Theory
   (e) Satisfaction and Feasibility Studies
      i. Satisfaction: Correctness, unambiguity, completeness, consistency, stability, verifiability, modifiability, traceability
      ii. Feasibility: Technical, economic, BPR

Figure 6: Requirements engineering document table-of-contents: analytic documents

Requirements Engineering Stages and Steps

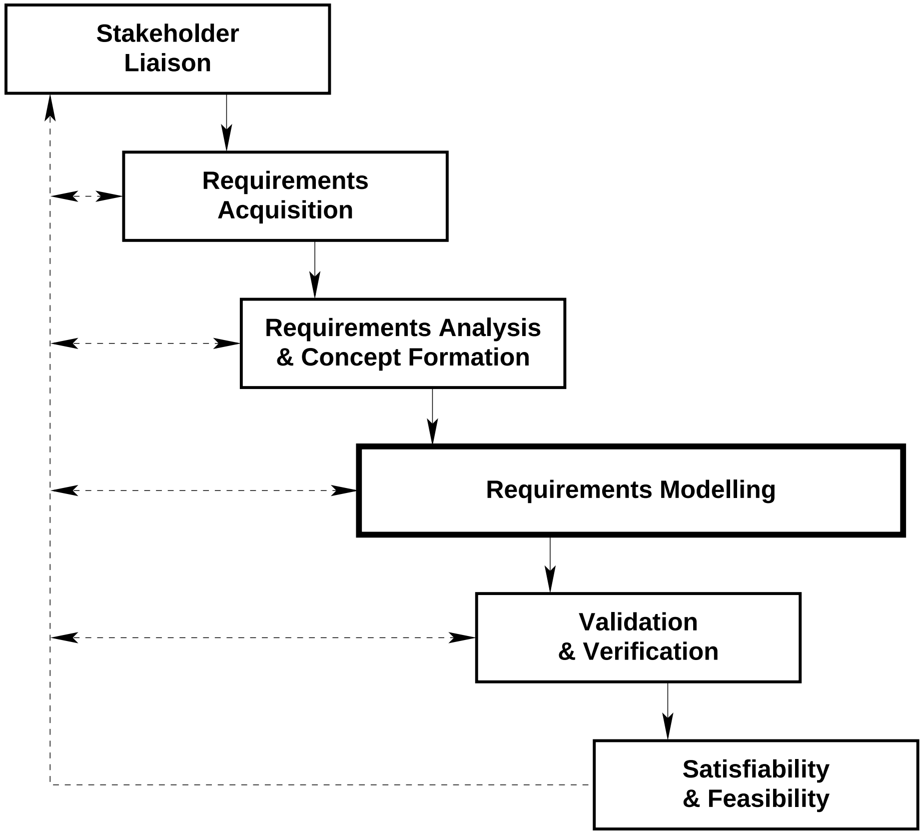

Figure 7: Diagramming a requirements process model

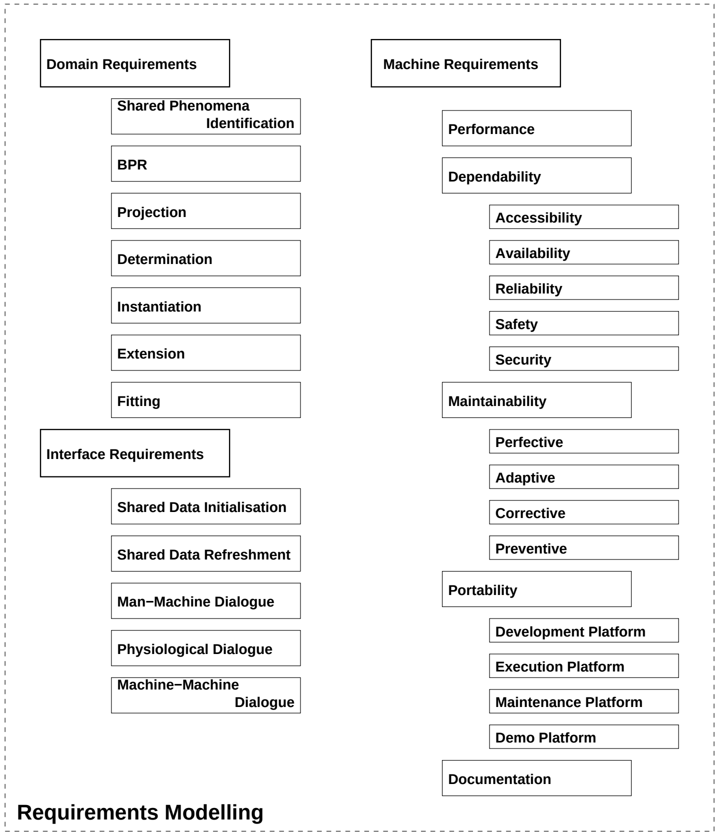

Figure 8: The requirements modelling stage

The Software Design Process Model
Software Design, A Narrative

- The software design process is here simplified into four stages (Fig. 9 items 2.a–d):
  - ⋆ software architecture design,
  - ⋆ component design,
  - ⋆ module design, and (module)
  - ⋆ program coding.
- Each of these may consist of two or more steps of development (cf. Fig. 10).
- Between adjacent steps there is a correctness obligation (V:MC:T, verification, model checking and testing).
  - ⋆ Verification proofs usually are of the kind: $\mathcal{D}, \mathcal{S} \models \mathcal{R}$
  - ⋆ which means that the proof that the $\mathcal{S}$oftware implements the $\mathcal{R}$equirements entails reference to the $\mathcal{D}$.

Software Design Documents

2. Software Specifications
   (a) Architecture Design ($S_{a_1} \ldots S_{a_n}$)
   (b) Component Design ($S_{c_{1_i}} \ldots S_{c_{n_j}}$)
   (c) Module Design ($S_{m_1} \ldots S_{m_m}$)
   (d) Program Coding ($S_{k_1}, \ldots, S_{k_n}$)
3. Analyses
   (a) Analysis Objectives and Strategies
   (b) Verification (S_{i_p}, $S_i \sqsupseteq_{L_i} S_{i+1}$)
      i. Theorems and Lemmas L_i
      ii. Proof Scripts $\wp_i$
      iii. Proofs Π_i
   (c) Model Checking ($S_i \sqsupseteq P_{i-1}$)
      i. Model Checkers
      ii. Propositions P_i
      iii. Model Checks $\mathcal{M}_i$
   (d) Testing ($S_i \sqsupseteq T_i$)
      i. Manual Testing
         - Manual Tests $M_{S_1} \ldots M_{S_\mu}$
      ii. Computerised Testing
         A. Unit (or Module) Tests C_u
         B. Component Tests C_c
         C. Integration Tests C_i
         D. System Tests $C_s \ldots C_{s_{i_{t_s}}}$
   (e) Evaluation of Adequacy of Analysis

Legend:

S Specification

L Theorem or Lemma

$\wp_i$ Proof Scripts

Π_i Proof Listings

P Proposition

$\mathcal{M}$ Model Check (run, report)

T Test Formulation

M Manual Check Report

C Computerised Check (run, report)

$\sqsupseteq$ "is correct with respect to (wrt.)"

$\sqsupseteq_\ell$ "is correct, modulo ℓ, wrt."

Figure 9: Software design document table-of-contents

Software Design Stages and Steps

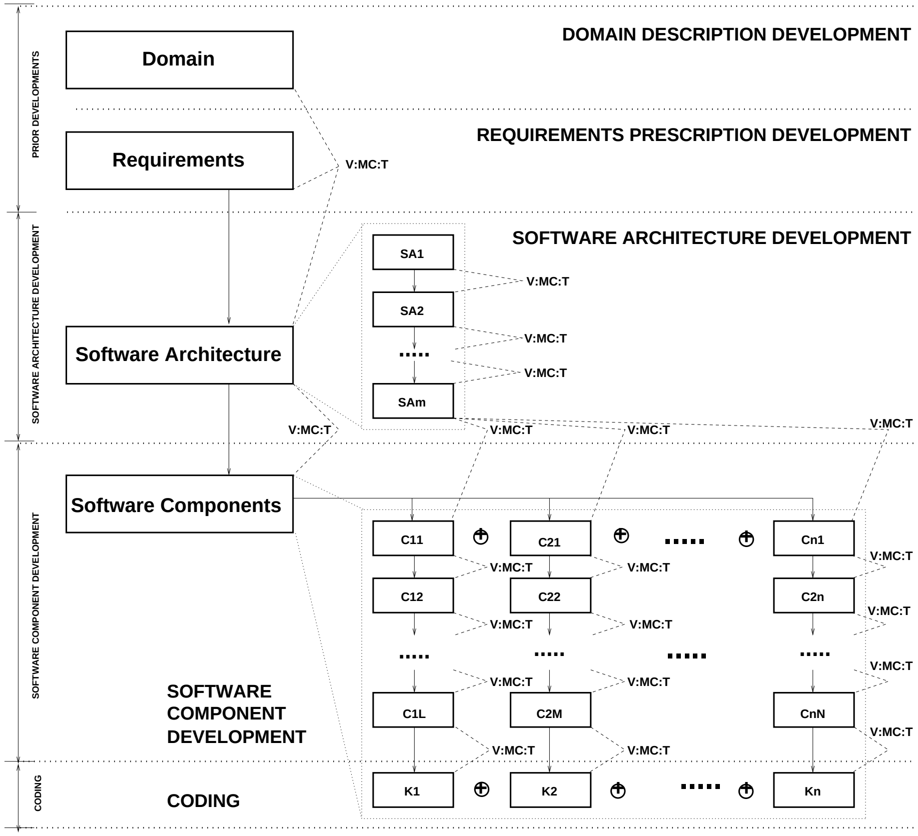

Figure 10: The software design development processes

Review of the Triptych Process

The Process Model: Diagrams and Tables-of-content

- We have surveyed the (mainly) software development processes according to the triptych dogma.
- We have seen that these processes can be diagrammed and also be "mapped" onto tables-of-content of the documents resulting from respective phases.

Process Model Semantics

- Diagrams,
- such as those of Figs. 1, 4, 7–8 and 10,
  - ⋆ reflect some pragmatics,
  - ⋆ has some syntax and embodies, hopefully
  - ⋆ some semantics.

- We wish, here, to emphasise the semantics:

  - *What is important to mention here, justifying this separate section,*
  - *is that each of the boxes of the description, prescription and software design parts of*
  - *Figs. 4, 7, 8 and 10*
  - *and each of their interconnecting edges*
  - *embody a clear set of method principles, techniques and tools*
  - *with many of these techniques also being pursuable formally and supported, or supportable, by theory-based tools.*

In the following we shall assume that the above ***bullets*** on the semantics of the process model diagrams is taken for granted.

Informal versus Formal Development

- The term 'development' covers any combination of the three phases:
  - ⋆ domain, requirements or software design only;
  - ⋆ domain+requirements or requirements+software design,
  - ⋆ or all three phases "more-or-less" consecutively.
- Development can, as shown in my book, all three volumes, be pursued
  - ⋆ **informally** or
  - ⋆ **formally**,
- and therefore in any "graded scale" combination of these.

- **0. Informal development** means: no formalisation of
  - ⋆ domain descriptions,
  - ⋆ requirements prescriptions or
  - ⋆ software design specifications

  are attempted.
- Thus verification cannot be done using formal proofs or model checking.
- Only testing.

There are, roughly speaking three "points" on the semi-formal to formal scale of development.

- **1. Systematic development** formalises domain descriptions, requirements prescriptions and software design specifications. But that is just about as much formalisation that is attempted.
- **2. Rigorous development** extends systematic development by stating all "crucial"[3] properties and maybe even sketch or carry through the proof or model checking of properties.
- **3. Formal development** requires that all necessary (including correctness) properties are formally expressed and theorem proved or model checked.

The triptych paradigm allows for any of these latter three (1.–2.–3.) forms of development.

[3] We do not here further characterise what we mean by 'crucial'.

Adherence to Phases, Stages and Steps

It is important to stress the following assumption:

- *Adhering to the triptych paradigm, to us,*
  - ⋆ *means that all phases, stages and steps as outlined above are followed.*
  - ⋆ *This means that documents are produced as per the tables-of-contents shown earlier.*

Our treatment, next, of process assessment and improvement, is based on, i.e., starts with the above assumption.

Process Assessment and Improvement Management

Notions of ‘Process Assessment’ and ‘Improvement’

- In order to speak of ‘assessment’ and ‘improvement’ we must identify that which is being assessed and improved:
  - ⋆ the results of following one set of method principles, techniques, tools and their management, over following another such set.
- Process assessment is now about judging adherence of a given process to its process model, pragmatically, semantically and syntactically (pss, usually in reverse order):
  - ⋆ to which (pss) degrees does the process fulfill what is “laid down” in the process model.
- Process improvement is then about changing the assessed development processes such that the results of using the changed processes are assessed to have been improved.
- By “assessment” and “improvement” we first of all mean “assessing and improving documents”.
- The documents are those emanating from activities denoted by nodes and edges of the process model.

- Each such box and each such edge may have many documents “attached” to it, and each such document has its syntax, semantics and pragmatics.
- The syntax and semantics can usually be given very precise definitions.
- Hence we can, in a sense, objectively “measure” (assess) whether a document “lives up” to that syntax and that semantics! For pragmatics the “measure” is more subjective.
- To be able to “measure” process improvement one must therefore attach to each planned document for each box and each edge a “measure” of compliance.
- Is a document in 100% compliance with those syntactic, semantics and pragmatic measures or is it not?
- Or more precisely: where on a scale from 0 to 1 lies the quality of a document wrt. an “ideal”.

Software Process Assessment 1

Process Model Syntax and Semantics: *In order to handle process improvement (à la CMM, from a lower to a higher level) — using the triptych approach — managers (as well as, of course, developers), must be* **intimately familiar with the syntax and semantics of the documents** *produced and expected to be produced by process model node and edge activities. This is a strong requirement and can not be expected by just any software development organisation. And there are really no shortcuts.*[4] *Process improvement — wrt. the precision of monitoring resource usage — is predicated on this assumption: that management is strongly based on professional awareness of triptych principles, techniques and tools. The "degree"*[5] *to which a development document adheres to the syntax and semantics of the relevant box or edge thus provides an assessment.*

[4] In other branches of engineering project managers (i.e., project leaders) and developers, the "engineers at floor level" basically all have the same, normalising education. Hence they are intimately familiar with the syntax and semantics of their tasks. The problem is in software engineering.

[5] This "degree" notion is not defined here

Several groups, worldwide, the most well known is perhaps Praxis High Integrity Systems, http://www.praxis-his.com, practices this on a daily basis. So do many members of ForTIA: The Formal Techniques Industrial Association, www.fortia.org.

SOFTWARE PROCESS IMPROVEMENT 1
Process Model Syntax and Semantics: *To improve this general aspect of the possible processes that developers and managers might be able to pursue under the banner of the Triptych Process Model one simply has to resort to* **education and training** . *There is no substitute.*

- We choose here to **also** "anchor" our discourse of 'process improvement' by referring to the *Capability Maturity Model* (CMM) of Watts S. Humphrey (WSH).
- CMM postulates five levels of maturity of development groups.
  - ⋆ Level 1 being a lowest, in a sense "least desirable",
  - ⋆ and level 5 being the highest, "most desirable" level of professionalism that WSH finds useful to define.
- Process improvement, by a development group,
  - ⋆ is now the improvement of the development processes
  - ⋆ such that the group (i.e., the software house)
  - ⋆ advances from level i to level $i + j$
  - ⋆ where i, j are positive numbers and $i + j$ is less than 6.
- So let us first review WSH's notion of CMM.

The CMM: Capability Maturity Model

1. **Level 1, Initial**
2. **Level 2, Repeatable**
3. **Level 3, Defined**
4. **Level 4, Managed**
5. **Level 5, Optimizing**

- A critical distinction between maturity level 4 and maturity level 5 is the type of process variation addressed.
  - ⋆ At maturity level 4, processes are concerned with addressing special causes of process variation and providing statistical predictability of the results.
  - ⋆ Though processes may produce predictable results, the results may be insufficient to achieve the established objectives.
  - ⋆ At maturity level 5, processes are concerned with addressing common causes of process variation and changing the process (that is, shifting the mean of the process performance) to improve process performance (while maintaining statistical probability) to achieve the established quantitative process-improvement objectives.

Process Models and Processes

- One thing is the process model, viz., the graph-like structures shown in, for example, Fig. 4, Figs. 7 and 8, and Fig. 10.
- (These are syntactic structures, but have semantic meanings.)
- Another thing is the actual usage of such models,
  - ⋆ that is, the actual processes that the software developers
  - ⋆ (domain, requirements and software design engineers)
  - ⋆ "steer through"
  - ⋆ when developing domain models, requirements models and software designs.

Graphs and Graph Traversal Traces

- Assume some graph-like, let us call it, process model, see Fig. 11.

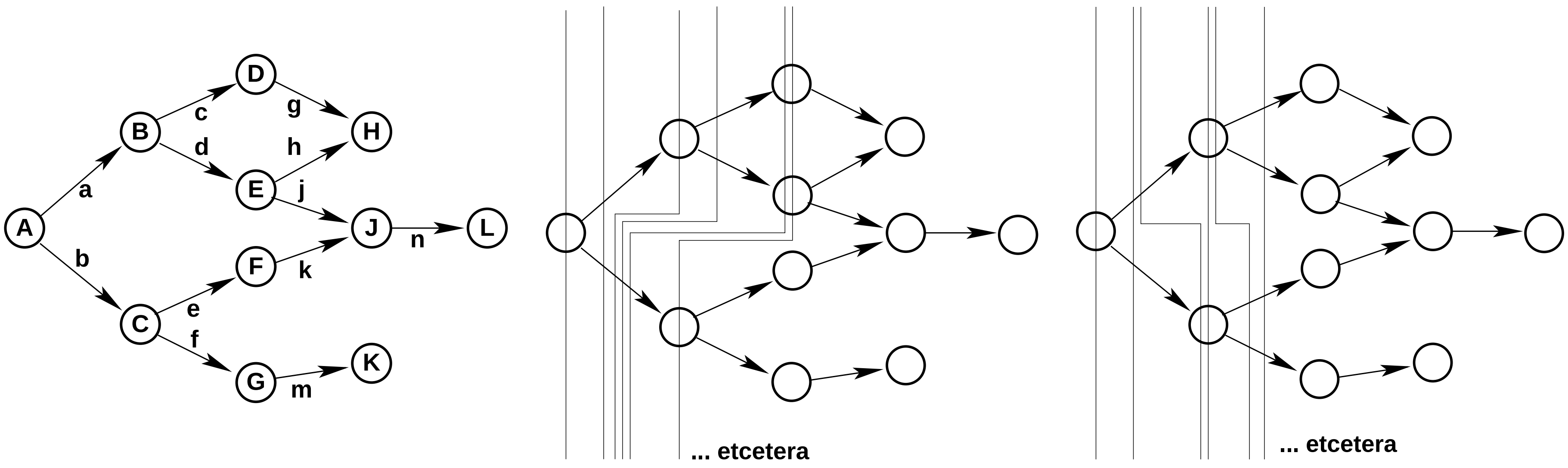

Figure 11: A graph (left) and two (incomplete) traversal traces (center and right)

- So Fig. 11 shows a process model and two traces.
  - ⋆ REDOs, that is, iterations of phases, stages and steps lead to additional traces.
    - ◇ Let us call the totality (set) of these traces for OK traces.
  - ⋆ And "jumping" or just "skipping" phases, stages and steps lead to further additional traces.
    - ◇ Let us call these "jumped" or "skipped" traces for NOK traces.
- A process model thus denotes a possibly infinite set of such traces.

Degrees of Process Model Compliance

- We can now define two notions of process model compliance,
  - ⋆ a syntactic and
  - ⋆ a semantic.
- The **syntactic** notion of **process model compliance** has to do with “the degree” to which an actual process matches a possibly iterated, i.e., an OK trace of a process model.
- The **semantic** notion of **process model compliance** is concerned with adherence to the semantics of boxes and edges.
- We shall not, in this talk define these notions precisely — that should be done in a future talk.

7

Software Process Assessment 2

Syntactic Process Compliance:

- *Given the generic process models diagrammed in Figs. 4, 7, 8 and 10,*
- *and given the project-specific software development graph as exemplified by Fig. 12,*
- *one can now, in a process claimed to adhere to these models and graphs*
- *quite simply assess whether that actual process follows those diagrams.*

Software Process Improvement 2

Syntactic Process Compliance:

- *Adherence to the process model can,*
- *at least "formally" (wrt.), be improved*
- *by actually ensuring that the process steps and stages (or even phases)*
- *that were assessed to not having been performed, that these be performed.*

A “Base 0” for Triptych Developments

- By a triptych development we mean a development which applies the principles, techniques and tools as prescribed by the triptych dogma.
- Either in a systematic, or in a rigorous, or in a formal way.
- **A triptych development process therefore, “by definition” has its base point at level 4 in the CMM scale.**
- This does not mean that a software development process which claims to follow the triptych dogma (or the software house within which that process occurs) at least measures at level 4.
- The dogma sets standards.

Software Process Assessment 3

Planned Syntactic and Semantics Compliance:

- *If a process is assessed (SPA) to be in full compliance,*
- *syntactically and semantically with the process model*
- *then we claim that the software development in this case is at CMM level 4 (or higher).*

Software Process Improvement 3

Planned Syntactic and Semantics Compliance:

- *If it is assessed that a process has not reached CMM level 4,*
- *and that at least CMM level 4 is desired,*
- *then one must first secure syntactic compliance, see process improvement #2 (Slide 46),*
- *thereafter ensure that each of the steps (or stages, or phases)*
  - ⋆ *whose semantic compliance was assessed too low*
  - ⋆ *be redone and according to their semantic intents.*

Proactive Measures

- The above spoke in general about assessment and improvement.
- We are now ready to deal with more specific issues of process assessment and improvement.
- But first we need to refine our notion of process model.

Project Development Graphs

- The process models (i.e., the graphs) are generic.
- They apply to any development — whatever the software.
- They must be instantiated to fit the particular problem frame.
- Figure 12 shows the project development graph that was used in the development of the Danish Ada compiler (1981–1984).

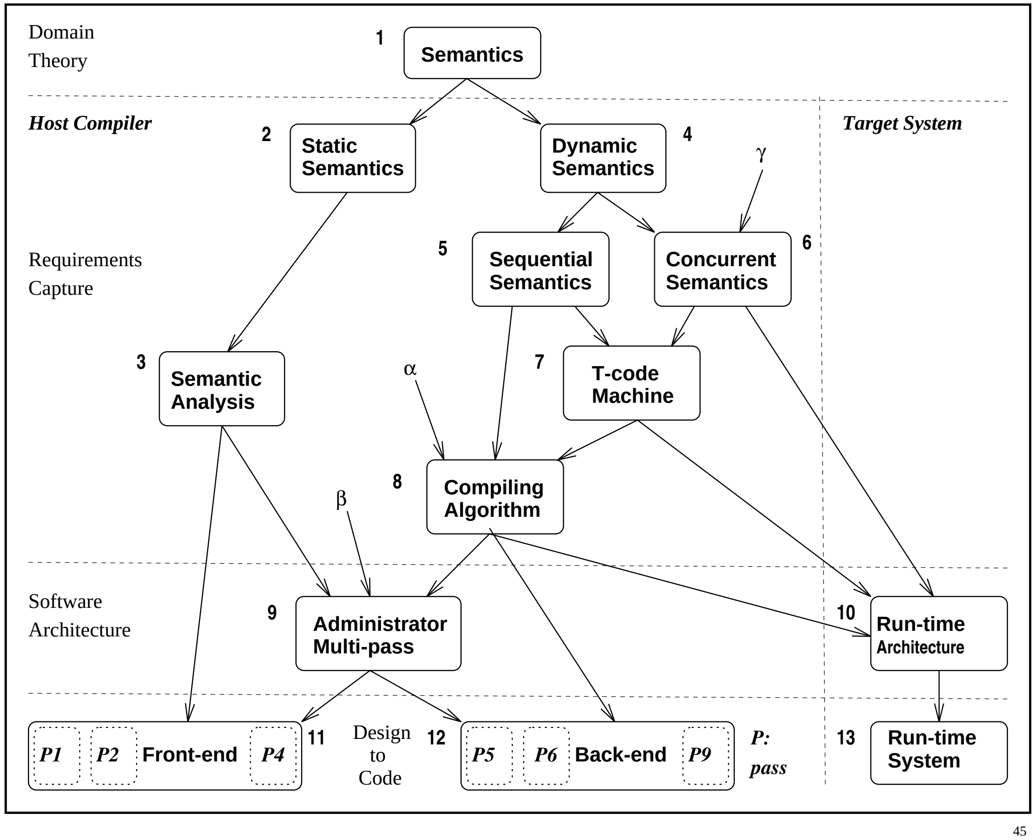

Figure 12: Project development graph: Compiler development

- The graph details the three phase process model.

Management

- So far, in this paper, we have not dealt with management.
- Management[6] is about planning, and monitoring and controlling process resource usage — including the quality of the documents emanating from the use of resources.
- Planning is about scheduling and rescheduling processes and allocating and re- and deallocating resources to (from) processes.
- A primary resource in software development is the set of domain and requirements engineers and the set of software designers.
- Other primary resources are the time, space and tools used by these developers.

[6] We restrict management to the below items. That is: we do not consider product management (which products to develop and in which sequence of deliverables) nor project funding.

Planning — Scheduling and Allocation

- Planning starts with instantiating, selecting, or developing a new, tentative, software development graph and
- detailing (i.e., annotating) it wrt. process model concepts:
  - ⋆ phases (domain, requirements, software design),
  - ⋆ stages (stakeholder identification, acquisition, analysis, description (prescription, specification), verification, model checking, testing, validation, etc.),
  - ⋆ and make allowances for more crucial, detailed steps.

- Based on the resulting software development graph
- management can, in a far more detailed (i.e., granular) way,
- ascribe resource usage (people, time, offices, equipment, software development tools)
- to each box and edge,
- and can schedule these in time and allocate them "in space".

Software Process Assessment 4

Resource Planning:

- *How can one assess a software development project plan (i.e., graph), that is, something which designates something yet to happen?*
- *Well, one can compare to previous software development graphs purporting to cover "similar" (if not identical) development problems and their eventual outcome, that is, the process that resulted from following those graphs.*
- *Based on actual resource useage accounts one can now — "to the best of anyone's ability" — draw a software development graph and ascribe resource consumption estimates (time, people, equipment) to each and every node and edge.*
- *Thus 'assessment' here was "speculated assessment" of an upcoming project.*

Thus, if that 'speculated assessment' of an upcoming project is felt, by the assessors, i.e., the management, to be flawed, to be questionable, then one has to proceed to improvement:

Software Process Improvement 4

Resource Planning:

- *One must first improve the precision with which one designs the domain specific project development graphs.*
- *Then the precision with which we associate resource usage with each box and edge of such a graph. Etcetera.*
- *Some development projects are very much "repeats" of earlier such projects and one can expect improvement in project development graphs for each "repeat".*
- *Other projects are very much tentative, explorative,*
  - ⋆ *that is, are actually applied research projects —*
  - ⋆ *for which one only knows of a project development graph at the end of the project,*
  - ⋆ *and then that graph is not necessarily a "best such"!*

Monitoring & Controlling Resource Usage

- As the project (i.e., the process) evolves
- management can now check a number of things:
  - ⋆ adherence to schedule and allocation, and
  - ⋆ adherence to the syntactic and the semantic notions of process model compliance.

- Most process models do not possess other than rather superficial and then mostly syntactic notions of compliance.
- In the triptych process model semantic compliance is at the very core:
  - ⋆ Every box and every edge of the process models
  - ⋆ have precise syntax and semantics of the
  - ⋆ documents that are the expected results of these (box and edge) activities.

Software Process Assessment 5

Resource Useage:

- *No problems here.*
  - ⋆ *As each step (of the development process) unfolds*
  - ⋆ *one can assess its compliance to estimated plan.*

- Should a resource useage assessment reveal that there are problems
- (for example: all resources used well before completion of step)
- then something must be done:

Software Process Improvement 5

Resource Useage:

- *Well, perhaps not this time around, when all planned resources have already been consumed —*
- *no improvement can undo that — but perhaps "next" time around.*
- *An audit may reveal what the cause of the over-consumption was.*
- *Either a naïve, too low resource estimate, or unqualified staff, or some simple or not so simple mistakes?*
- *Improvement now means: make precautions to avoid a repetition.*

- Resource usage is at a very detailed and accountable level and can thus be better assessed.
- Slips (usually excess usage)
  - ⋆ can be better foreseen and discovered
  - ⋆ and more clearly defined remedies,
  - ⋆ should milestones be missed or usage exceeded,
  - ⋆ can then be prescribed —
  - ⋆ including skipping stages and steps whose omission are deemed acceptable.

- Skipping stages and steps result in complete, perhaps extraneous (ad hoc) processes.
- Given that management has an “ideal” process model and
  - ⋆ hence an understanding of desirable, possibly iterated processes,
  - ⋆ management can now better assess which are acceptable slips.

From Informal to Formal Development

- By process improvement,
  - ⋆ to repeat and to enlarge on our previous characterisation of what is meant by process improvement,
  - ⋆ we understand something which improves the quality of resulting software.
- We "translate" the term 'resulting software' into the term 'resulting documents'.
- These documents can be developed either
  - ⋆ informally (without any use of any formalism other than the final programming language),

  or

  - ⋆ systematically formal,
  - ⋆ or rigorously formal
  - ⋆ or formally formal!

Informal Development

- It is an indispensable property of the triptych approach to software development
  - ⋆ that the formalisable steps domain engineering, requirements engineering and software design
  - ⋆ be pursued in some systematic via rigorous to formal manner.
- Hence the informal aspects of development is restricted to the development of only the informative documents.
- Informative documents are usually "developed" by project leaders and managers.
- Hence an "upper" level of management is process assessing and possibly prescribing process improvements to a "lower" level of management!

Software Process Assessment 6

Informal Development of Informative Documents:

- *We refer to Fig. 2 (Slide 8).*
- *That figure lists the kind of documents to be carefully developed — and hence assessed.*
- *Since no prescribed syntax,*
  - ⋆ *let alone formal semantics can be given for these documents*
  - ⋆ *— whose purpose is mainly pragmatic —*
  - ⋆ *assessment is a matter of style.*
  - ⋆ *It is easy to write non-sensical, "pat" informative documents which do not convey any essence, any insight.*
- *Assessment hence has to evaluate: dose a particular, of the many informative documents listed in Fig. 2, really convey, in succinct form, an essence of the project being initiated?*

Software Process Improvement 6

Informal Development of Informative Documents:

- *If an informative document is assessed*
  - ⋆ *to not convey its intended message succinctly,*
  - ⋆ *with necessary pedagogical and didactical "bravour",*
- *then it must be improved.*
- *Only "seasoned", i.e., experienced managers can do this.*

Systematic, Rigorous and Formal Development

- The development of
  - ⋆ domain description,
  - ⋆ requirements prescription and
  - ⋆ software design

  documents as well as the development of analytic documents
  - ⋆ (tests, verification, model checking and validation)
- can be done in a spectrum from systematically via rigorously to formally.

Software Process Assessment 7

Staff and Tool Qualification:

- *Given the syntax and semantics of the specific step —*
  - ⋆ *in the process model —*
  - ⋆ *of the tasks to be assessed a (syntax and semantics)*
- *a knowledgeable person, a project (task) leader or a manager,*
- *can assess compliance.*
- *That assessment is greatly assisted by the software tools*[7] *that support activities of those tasks:*
- *If they can process the documents*
- *then something seems OK.*
- *If not, assessment will have to be negative.*

[7]These software tools mainly support the use of the main tools, namely the specification languages, their transformation (or refinement) and their proof systems.

- There are now two distinct, "extreme" reasons for a failure to meet assessment criteria —
- with any actual reason possibly being a combination of these two "extremes".
  - ⋆ One is that the quality of the staff performing the affected tasks is not up to expectations.
  - ⋆ The other is that the tools being deployed are not capable of supporting the problem solution task.

Staff Qualification

- If the assessment of
- 'Systematic, Rigorous and Formal Development of Specifications and Their Analysis'
- is judged negative due to inadequate development decisions
- then we suggest the following kind of improvement.

Software Process Improvement 7

Staff Qualification:

- *It is suggested that improvement, when deemed necessary, takes either of three forms:*
  - ⋆ *Either*
    - ◇ *"move" from a systematic to a rigorous level of development,*
    - ◇ *or from a rigorous to a formal level of developement when that is possible*
    - ◇ *and redo the task(s) affected.*

- ⋆ *Or*
  - ◇ *educate and train staff*
  - ◇ *to re-perform the affected task(s) more accurately*
  - ◇ *(while remaining systematic, rigorous, or formal as the case may be.*
- ⋆ *Or*
  - ◇ *replace affected staff with better educated and trained staff*
  - ◇ *and redo the task(s) affected.*

- *These kinds of improvement decisions are serious ones.*

Tools

- There are different categories of tools.
  - ⋆ Tools can serve management:
    - ◇ for the design of software development graphs (a la Fig. 12) and
    - ◇ their "fusion" into the appropriate process model diagrams (a la Fig. 4, Fig. 7 and 8, and Fig. 10)
    - ◇ and for the monitoring and control (i.e., assessment and improvement) of the process with respect to these diagrams.

⋆ And tools can serve developers:

- ◇ syntactic and
- ◇ semantic

description, prescription and software design tools as well as analytic tools:

- ◇ for testing,
- ◇ model checking and
- ◇ verification (proof assistance or theorem provers).

- ⋆ These tools embody, that is, represent the formalisms of the textual or diagrammatic notations used — whether
  - ◇ Alloy,
  - ◇ B,
  - ◇ CafeOBJ ,
  - ◇ Cas,
  - ◇ Duration Calculus,
  - ◇ LSCs,
  - ◇ MSCs,
  - ◇ Petri Nets,
  - ◇ RAISE RSL,
  - ◇ Statecharts,
  - ◇ TLA+,
  - ◇ VDM-SL, or
  - ◇ Z.
- ⋆ Thus the formal notations of the above listed thirteen languages,
  - ◇ whether textual
  - ◇ or diagrammatic,
  - ◇ or combinatons thereof,

  are tools,
- ⋆ as are the software packages that support uses of these linguistic and analytic means.

Tool Qualification

- If assessmment of 'Systematic, Rigorous and Formal Development of Specifications and Their Analysis'
- is judged negative due to inadequate tools
- then we suggest the following kind of improvement:

Software Process Improvement 8

Tool Qualification:

- *Better tools must be selected and applied to the task(s) affected (i.e., judged negatively assessed).*
- *These tools are*
  - ⋆ *either intellectual, that is, the specification languages, whether textual or diagrammatic, and their refinement and proof systems,*
  - ⋆ *or they are the manifest software tools that support the intellectual tools.*
- *These are likewise a serious improvement decisions.*

Hindrances to Process Assessment and Improvement

- What could be "standard" hindrances to assessment and improvement?
- And what could be similar hindrances
  - ⋆ to actually carrying out projects
  - ⋆ according to the triptych process model?

Lack of Knowledge of Methodology

- Both management and development staff must be intimately familiar with
  - ⋆ the triptych process model and its syntactic, semantic and pragmatic implications,
  - ⋆ its need for from systematic via rigorous to formal development,
  - ⋆ its need for the creation, use, maintenance and correlation of myriads of documents, and
  - ⋆ its need for assessment and possible improvement.
- Lack of knowledge of the methodology, ever so sporadically, is a hindrance to proper software development processes.

Generation Gaps

- Classically we see that young candidates join software houses as software engineers,
  - ⋆ fluent in the kind of methods: principles, techniques and tools inherent in the triptych approach.
- They are eager to use these.
- But they are usually stifled:
  - ⋆ their slightly older colleagues as well as their project leaders and managers
  - ⋆ do not possess the same skills,
  - ⋆ or are outright illiterate wrt. the tripych methods: principles, techniques and tools.
- Lack of knowledge of the methodology, across generations of staff,
- is a hindrance to proper software development processes — and even a few years (say ten) count as a generation today.

Lack of Tools

- Above we pointed out that there we intellectual tools and there were software tools that support the use of the intellectual tools.
- Here we mean both.
- On one hand,
  - ⋆ the problem being tackled in a particular software development project
  - ⋆ may be such that there are, as of today, year 2006, no obvious or no good intellectual tools (and a methodological approach,
  - ⋆ i.e., a process model) for the properly assessable and improvable pursuit of such a project.
- On the other hand,
  - ⋆ even when appropriate intellectual tools are (and a process model is) available
  - ⋆ there may not be good manifest,
  - ⋆ that is, software support tools available.
- Lack of tools is a serious hindrance to proper software development processes.

Lack of Acceptance

- By far the most common hindrance to proper software development processes —
  - ⋆ such as suggested by the triptych process model — processes
  - ⋆ that can be properly assessed and
  - ⋆ for which a continuum of improvement possibilties exists —

  is

  - ⋆ (1) the lack of acceptance of what is referred to as "formal methods", and
  - ⋆ (2) the lack of acceptance of the necessity to do proper domain modelling before tackling requirements.
- This is not the time and place to lament on those "facts".

Conclusion

It is time to conclude.

Summary

- We have overviewed a rather comprehensive process model, the triptych model
  - ⋆ which prescribes three development phases:
    - ◇ domain engineering,
    - ◇ requirements engineering and
    - ◇ software design,
  - ⋆ and which, within these prescribes a number of stages and within these again a number os steps.
- Phases, stages and steps may be iterated, and phases, stages and steps, as well as the transition between them results in documents.

- We have modelled process models
  - ⋆ as acyclic graphs which
  - ⋆ denote possibly infinite sets of indefinite length traces of waves,
    - ◇ where a wave is a set of nodes and edges of the graph,
    - ◇ but where subsequences of traces may be repeated (due to process iterations: redoing "previous" tasks).

- We have then identified a class of
  - ⋆ seven software process assessment categories
  - ⋆ and eight software process improvement categories,

  all in relation to the syntax and semantics of the triptych process model.
- Finally we briefly touched upon hindrances to process assessment and improvement.

Future

- This is the first time the author has related the triptych model to SPA and SPI:
  - ⋆ software process assessment and software process improvement,
  - ⋆ and hence to CMM, Watts Humphrey's Capability Maturity Model.
- It has been instructive to do so.
- Clearly,
  - ⋆ for actual projects to apply the triptych approach
  - ⋆ and to carry out the assessments and improvements suggested in this paper,
  - ⋆ more clarifying directions must be given.
  - ⋆ And support tools developed.

Software Procurement
Software

- By software we shall here mean
  - ⋆ not just the executable code and some
  - ⋆ manuals on how to install, use and possibly repair this code,
  - ⋆ but also all the documents that emanates from a full project developing this code.

Procurement

- In software procurement it is therefore natural
  - ⋆ that the procurement includes as large a set of the documents mentioned in those figures,
  - ⋆ and that all these documents have passed an assessment with some positive, CMM level-relatable degree of acceptance.

Acknowledgments

- The speaker thanks
  - ⋆ Kouichi Kishida[8] for challenging him to write this paper for the audience of the Japan Software Process Improvement Consortium,
  - ⋆ Prof. Kokichi Futatsugi for providing the Japanese oral translation of the talk and
  - ⋆ Yasuhito Arimoto for translating English slides to Japanese slides.

[8] It is just simply too bad that K^2 is not here to hear the talk!
But I'll take my revenge:
Giving this talk in a seminar at Todai, Nov. 22nd, in honour of K^2's round birthday!