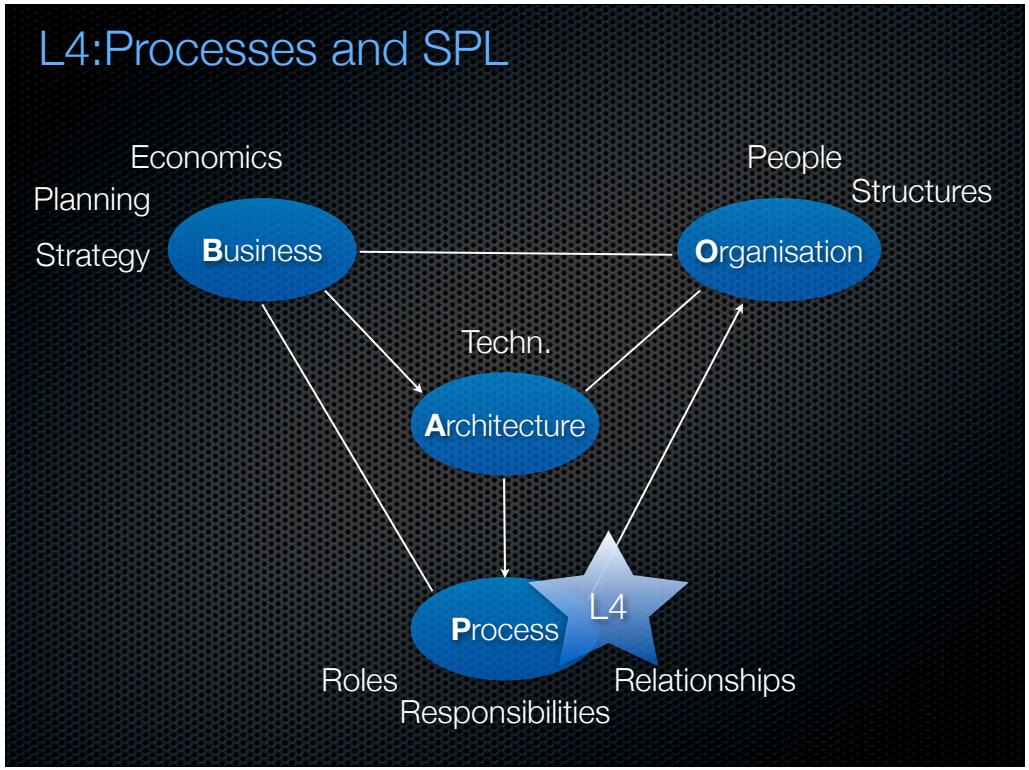
Software Product Line Engineering

L4:Processes and SPL

L5:Organizational Issues

L6:SPI/SPA

Tony Gorschek - tony.gorschek@gmail.com


L4:Processes and SPL
Economics
Planning
Strategy
Business
People
Structures
Organisation
Techn.
Architecture
Process
L4
Roles
Responsibilities
Relationships

Processes

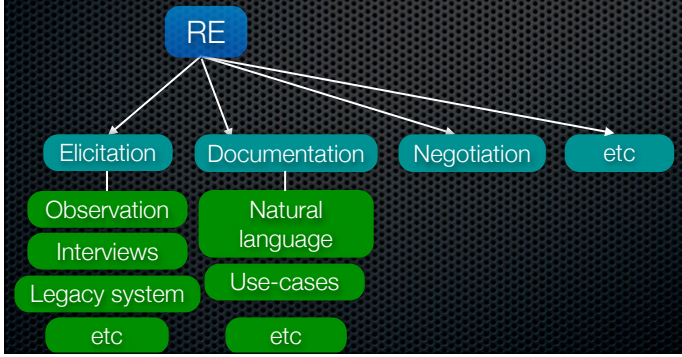
- Software Engineering Process: the total set of software engineering activities needed to transform requirements into software
- Product Development Process: the total set of engineering activities needed to transform requirements into products
  - Software (product) engineering refers to the disciplined application of engineering, scientific, and mathematical principles and methods to the economical production of quality software (products).

Process examples

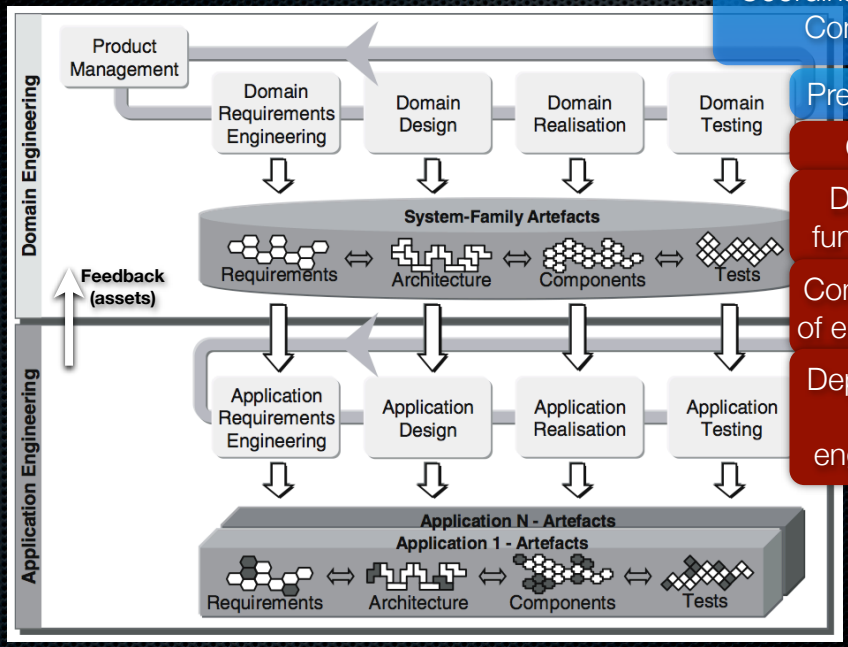
- Requirements Engineering (Main Process Area)
  - Elicitation (Sub-process Area)
    - Task observation (Activity/Action)
- Configuration Management
  - Configuration Item Identification
    - Risk analysis
    - Volatility (change Prone) analysis

Process examples

- Requirements Engineering (Main Process Area)
  - Elicitation (Sub-process Area)
    - Task observation (Activity/Action)
- Configuration Management (MPA)
  - Configuration Item Identification (SPA)
    - Risk analysis (Action), Change Prone analysis (Action)



SPL Process



Coordination and Control

Predictability

Quality

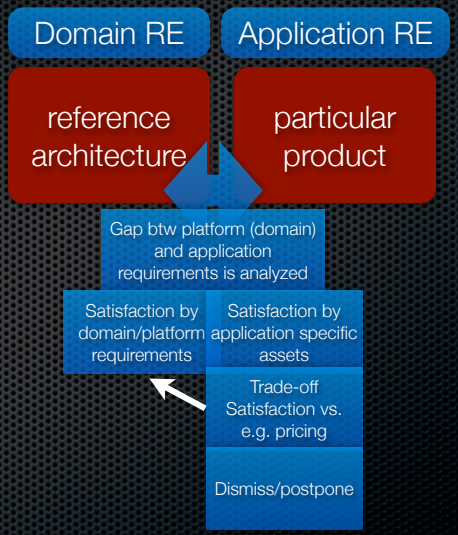
Delivered functionality

Commonality of engineering

Dependency heavy engineering

Requirements Engineering (RE)

- Elicitation
- Documentation
- Analysis and Negotiation
- Validation and Verification
- Management



Elicitation

- **Domain** (Understanding it)
- **Problem (application) domain**
  What's the problem(s) and who can explain it to you
- **History**
  Previous systems / current systems
  Documentation
  Old requirements/design etc.
- **Competitors**
  Have they solved the problem and how?
- **Surrounding environment**
  Other systems, processes which the system should support (and/or processes which the system influences)

Domain | Application

- internal (development org.) stakeholders (e.g. PM, developers, architects, support, STRATEGIES)
- external (customer, domain, environmental, regulatory)

need vs. want
stakeholder weights (politics) and access

- **Stakeholders**
  (management, users, future users, system managers, partners, sub contractors, Law and Policy, customer's customers, domain experts, developers etc)
- Finding them (Stakeholder Identification)
- Getting access to them (Cost, Politics)

PREPARATION

Elicitation techniques

- Interviews
  + Getting to know the present (domain, problems) and ideas for future system
  - Hard to see the goals and critical issues, subjective
- Group interviews
  + Stimulate each other, complete each other
  - Censorship, domination (some people may not get attention)
- Observation (Look at how people actually perform a task (or a combination of tasks) – record and review…)
  + Map current work, practices, processes
  - Critical issues seldom captured (e.g. you have to be observing when something goes wrong), usability issues seldom captured, time consuming
- Task demonstrations (Ask a user to perform a task and observe and study what is done, ask questions during)
  + Clarify what is done and how, current work
  - Your presence and questions may influence the user, critical issues seldom captured, usability problems hard to capture

Elicitation techniques 2

- Questionnaires
  + Gather information from many users (statistical indications, views, opinions)
  - Difficult to construct good questionnaires, questions often interpreted differently, hard to classify answers in open questions and closed questions may be to narrow…
- Use cases and Scenarios (Description of a particular interaction between the (proposed) system and one or more users (or other terminators, e.g. another system). A user is walked through the selected operations and the way in which they would like to interact with the system is recorded)
  + Concentration on the specific (rather than the general) which can give greater accuracy
  - Solution oriented (rather than problem oriented), can result in a premature design of the interface between the problem domain and the solution
- Prototyping
  + Visualization, stimulate ideas, usability centered, (can be combined with e.g. use cases)
  - Solution oriented (premature design), “is it already done?!”

Documentation

- **Natural Language (NL) Specification**
  (most common in industry)
  + Everyone can do it/understand
  + NL is a powerful notation (if used correctly)
  - Imprecise and Quality may vary
  - Use of attributes can improve accuracy
    ID, Title, Desc, Rationale, Source(s), Conflict, Dependencies, Prio. etc
- **Modeling** (where use-cases most common)
  + Relatively easy to do
  + Structure
  + Reuse of effort (e.g. code generation)
  - Imprecise and Quality may vary
  - Solution oriented, don't catch non functional aspects (Quality Requirements)
  - Cost/time

Context Diagrams
Event Lists
Screens & Prototypes
Scenarios
Task Descriptions
Standards
Tables & Decision Tables
Textual Process Descriptions
State Diagrams
State Transition Matrices
Activity Diagrams
Class Diagrams
Collaboration Diagrams
Sequence Diagrams

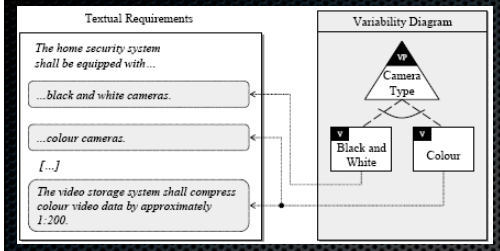
Complete
Correct
Feasible
Necessary
Prioritized
Unambiguous
Verifiable

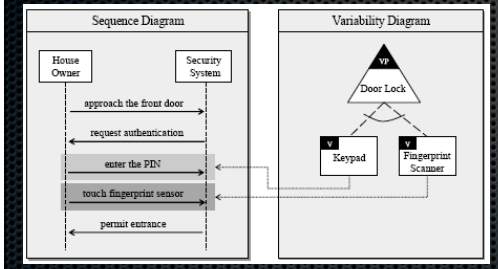
Documentation 2

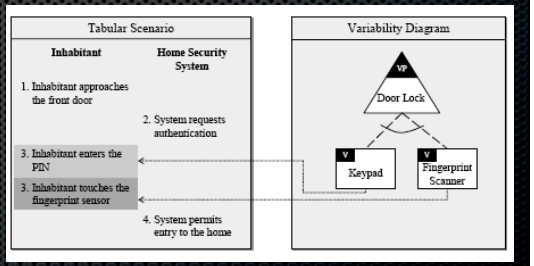
variability has to be mapped to requirements

Decision support: Domain or Application

Influences priority, risk, timeline, cost

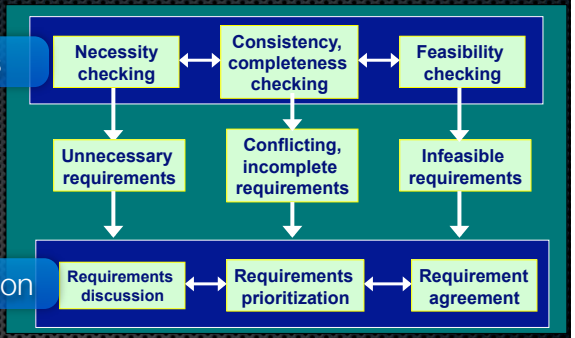






Analysis and Negotiation

Aims to discover problems with requirements and reach agreement that satisfies all stakeholders



- Premature design?
- Combined requirements?
- Realistic within Constraints?
- Understandable?
- Conformance with business goals?
- Ambiguous?
- Necessary requirement?
  - Customer Value
  - Gold Plating?
- Testable?
- Complete?
- Traceable?
- Consistent Terminology?
- Fit Criteria
  - Relevant?
  - Measurable?
- Requirement or Solution?

Techniques

Interaction Matrices
Requirements Classification
Requirements Risk Analysis
Boundary Definition

Verification and Validation (quality assurance)

- Verification is the process of determining that a system, or module, meets its specification
- Validation is the process of determining that a system is appropriate for its purpose

are we building the right system

check if we have elicited and documented the right requirements

Reviews/Inspections
Perspective based reading
Checklist based reading
Test Case Based Inspections
Two Man Inspection
(perspectives and checklist may include product line specific items like variability checks)

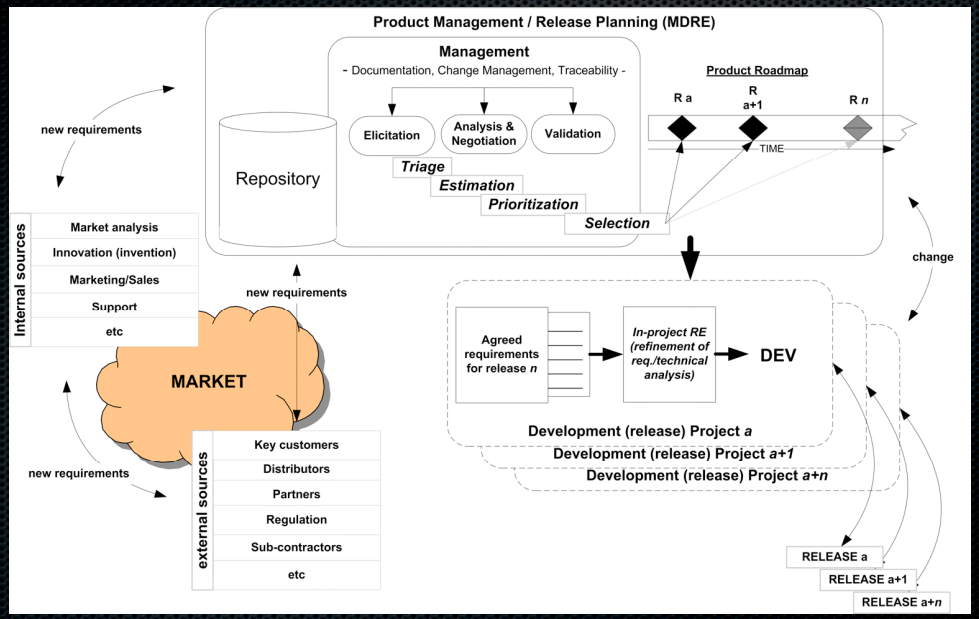
Reviews
Inspections
Checklists
Goal-Means Analysis
Req. Classifications
Prototyping
Simulation
Mock-Up
Test-Cases
Draft User Manual

the earlier you find a problem... errors introduced in the RE process are the most resource intensive to fix (50x more costly to fix defects during test than during the RE)

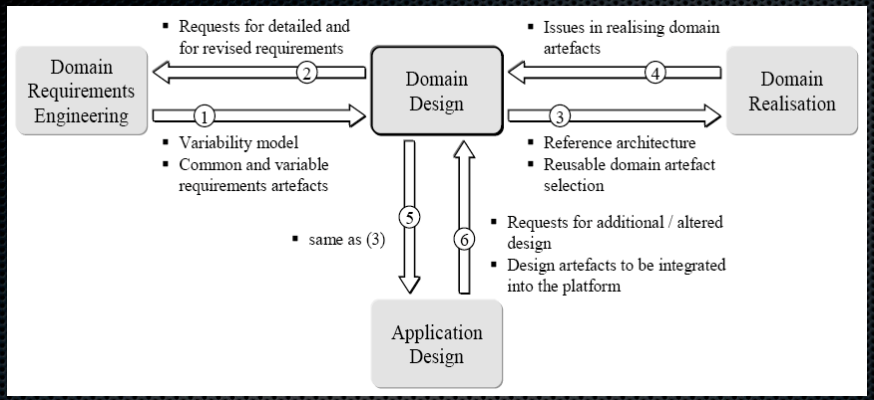
RE Management

- **Definition of the RE process and its interfaces and management of requirements and the requirements process over time**
  - **Configuration Management (!)** what to put under control; change management; version handling
  - **Tool support** tool that supports your process
  - **Traceability policies(!)** source, forward, backward (pre-requisite for reuse)
    - Focal Point, CaliberRM, Serena, Rational Req. Pro
  - **Reuse (!)** the artifacts you are creating may be reused = **quality and cost implications**
  - **Standards and policies (e.g. documentation)** least common denominator (what is good-enough) for RE you have to see beyond your role/needs
    - **Criteria for when to ignore policies**

Product Management



Domain Design

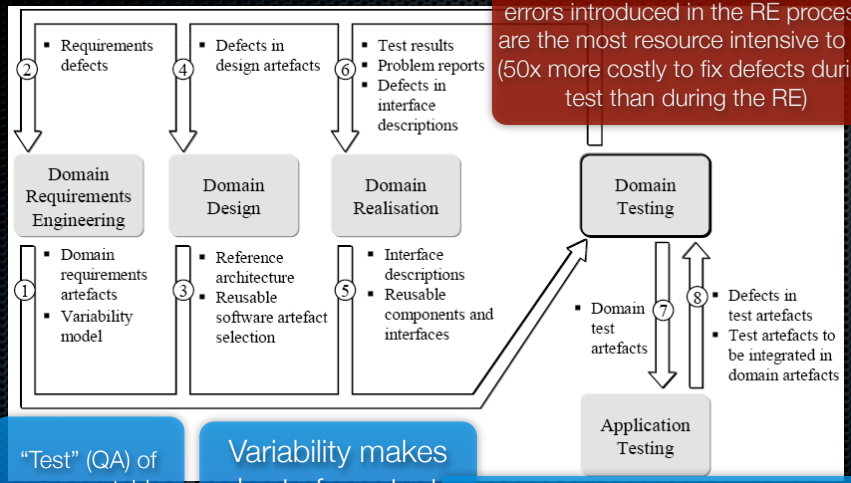


- Based on the reference requirements (delivered by PM and RE) create a reference architecture
  (variability and design covered in different lecture)

Domain Realization

- Make (assets built in-house)
  - control technical but also from a business perspective - is the asset a competitive (innovative asset)
- Buy (bought off-the-shelf)
  - often resource intensive assets (e.g. OS, middleware) but also infrastructure like RUP or CMMI
- Mine (reuse)
  - reuse of existing assets (e.g. other products) - often requires a lot of reengineering
    BUT application specific assets can be used and turned into a common asset
- Commission (3rd party)
  - specification in-house as a order to 3rd party (adherence to specification, specification quality, use of e.g. implementation proposals to assure common understanding)

Domain Testing



the earlier you find a problem... errors introduced in the RE process are the most resource intensive to fix (50x more costly to fix defects during test than during the RE)

"Test" (QA) of non-executables is !critical!

Variability makes brute force test impossible

Test suitable configurations (selected for best ROI) alt.
Use of e.g. stubs (fill on for absent/future plug-ins) BUT COST for creating and maintaining tests and e.g. stubs has to be weighed in (not to mention defects in test artifacts themselves)

Testing Strategy

BFS=Brute Force
PAS=Pure Application Strategy
SAS=Sample Application Strategy
CRS=Commonality and Reuse Strategy

	Time to create	Absent variants	Early validation	Learning effort	Overhead
(BFS)	-	-	+	0	-
(PAS)	0	+	-	+	-
SAS	0	+	+	+	-
CRS	+	+	0	-	+
Combined SAS/CRS	+	+	+	0	0

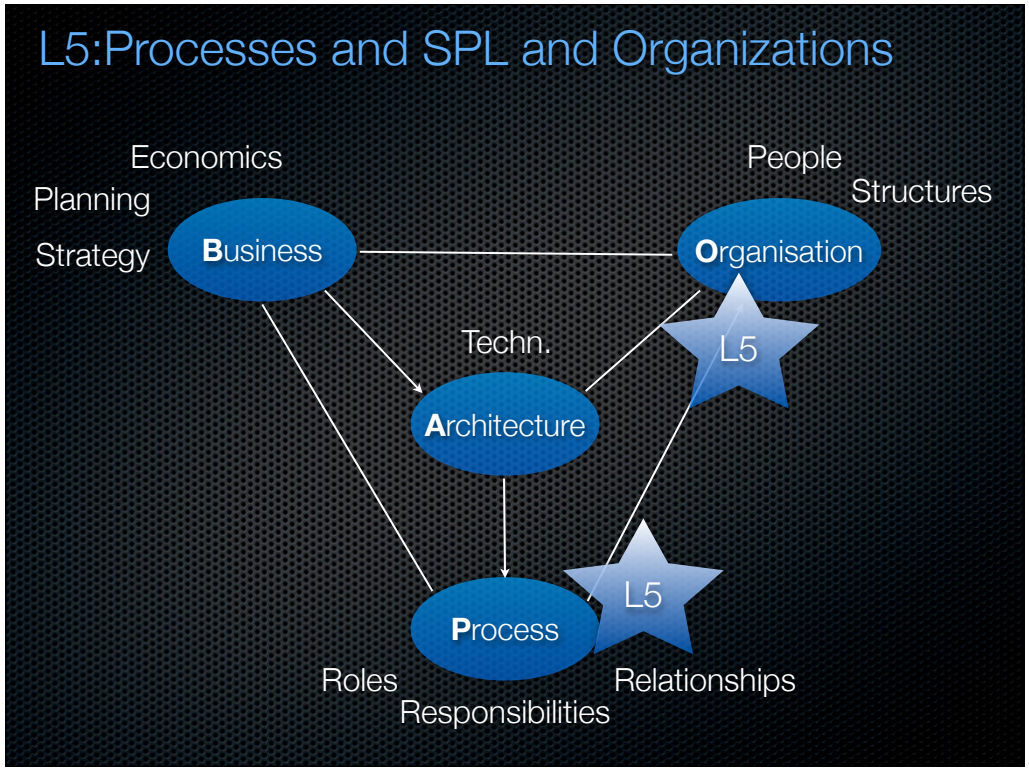
BFS. A "+" indicates that the
strategy yields positive results for a criterion, a "-" indicates that the strategy yields negative results for a criterion, and a "0" indicates that advantages and disadvantages are almost balanced for a criterion. For the BFS, the time to create test artefacts criterion is rated with a "-" due to the large amount of test artefacts that must be created. The learning effort is rated with a "0" as the BFS requires learning how to deal with different configurations, but avoids having to learn how to deal with variability in test artefacts. The inability of the strategy to deal with absent variants leads to a "-" for the absent variants criterion. Early validation gets a "+" as all tests are performed in domain testing. The overhead is rated with a "-" as most configurations are tested unnecessarily.

PAS - pure application strategy
The time to create test artefacts is rated with a "0" as it is roughly equal to the time needed in single-system engineering. As test engineers neither have to deal with absent variants nor with variability, the absent variants criterion and the learning effort are both rated with a "+". Early validation is rated with a "-" since no tests are performed in domain testing. The overhead is rated with a "-" since similar test cases have to be defined for each application.

SAS - *Sample Application Strategy*
Tests created for a specific application (can be reused but with adaptation) average
Absent variants are handled through creation of test apps
Test like normal = not hard to learn
Expensive as test apps have to be built

CRS - Commonality and Reuse Strategy
Domain testing aims at testing common parts and preparing test artefacts for variable parts. Application testing aims at reusing the test artefacts for common parts and reusing the predefined, variable domain test artefacts to test specific applications.
Tests can be reused in app testing = time low
Early validation not always possible as some test only possible after application engineering
Train testers to create test cases that include variability
Overhead low as reuse is possible

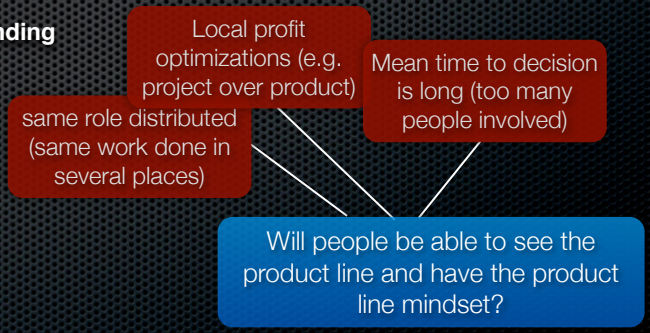
SAS/CRS
The composite strategy enforces the creation of reusable test artefacts in domain testing and the reuse of these artefacts in application testing. This leads to a good rating for the time criterion. In addition, an early validation is performed with fragments of a sample application. This means that no complete application is built, but only parts that are large enough to perform the tests. This indeed implies a minor overhead, but the overhead is significantly lower than the overhead of the SAS


L5:Processes and SPL and Organizations
Economics
Planning
Strategy
Business
People
Structures
Organisation
L5
Techn.
Architecture
L5
Process
Roles
Responsibilities
Relationships

Organization, roles and responsibilities

why should we bother with this...

- **Mapping of activities (actions) and process and roles to organization is critical as it is central to the successful realization and use of a PL**
  - **Amount of people working together (coherence within unit vs. collaboration btw units)**
  - **Accountability and funding**
  - **Decision hierarchy**



Organization, roles and responsibilities

why should we bother with this (2)...

- **Mapping of activities (actions) and process and roles to organization is critical as it is central to the successful realization and use of a PL**
  - **Organizational SIZE is crucial as it speaks to the impact of the organizational structure and the role and responsibilities division on the product line...**

Small organization has “closeness” and familiarity that can compensate for inadequacies, LARGE organizations DO NOT

“not my job”

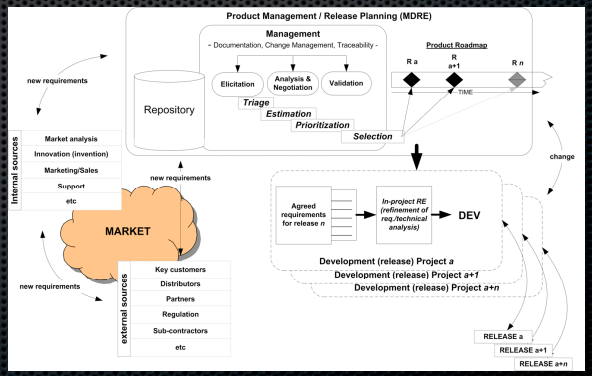
Personal mind-set, and motivational structure plays a crucial role if a PL succeeds or not, much more so than having a perfect architecture or variability analysis

Imbalance in the organization (e.g. domination of application engineering over domain engineering)

What are individual engineers good at (like to do), skill set! E.g. Domain Eng. (high quality components and maintenance) vs. App. Eng. (build apps fast w. given components)

Roles and responsibilities

- **Product Manager (PM)**
  - **Planning and evolution of the complete range of products (present and future) taking features and BUSINESS value into consideration**
  - **Business value -> Business owner, Features -> marketing and sales**
  - **Domain requirements engineering -> evolution of the features (commonality and variability)**
  - **PM initiates application development and coordinates with the application requirements engineer**



Roles and responsibilities 2

- **Domain Requirements Engineer**
  - Development and maintenance of the requirements that are relevant for the whole range of products (domain), i.e. the development of common and variable requirements incl. a variability model (in accordance with the roadmaps and plans of the PM)
  - Estimation and feasibility feedback
  - Common and variable req. + variability model -> input to domain architect
- **Domain Architect**
  - Development and maintenance of the reference architecture for the complete set of products
  - Collaborates a lot with the domain requirements engineer
  - The common and variable parts of the arch. are provided to the domain asset manager who performs management on variants and versions
  - Reference architecture -> input to domain developer (includes the selection of reusable domain components and interfaces)
  - The domain architect validates that the designs of the reusable assets fulfill/ adhere to the reference arch.
  - To enable configuring, the domain arch. determines what configuration mechanisms should be used to build end products.
  - Domain architect validates application architectures - adherence to domain arch. + reference arch -> is used by the application architectures

Roles and responsibilities 4

- **Domain Developer**
  - Development and maintenance of reusable components and interfaces for the complete range of products
  - Development of configuration mechanisms (e.g. through parameters, on model/design level, on CM level (e.g. versions) etc) to support the variance of the systems in the product line
- **Domain Tester**
  - Development and maintenance of reusable test assets for the complete range of products
  - Testing of integrated products, but also integration and system tests on domain assets, and prepare common and variable test assets to be used by the application tester (make sure to plan what has to be tested from a domain perspective in the individual applications)
  - Domain tester -> input to RE (testability etc), -> to PM regarding costs, -> to architect and domain developer as to testability on domain level
- **Domain Asset Manager**
  - Maintaining versions and variants of all domain assets! (everything from requirements to test cases and executables)
  - Traceability and configuration control (-> e.g. versions of individual artifacts to application configurations are kept traceable and under CM control)
  - Large potential of overhead!

Roles and responsibilities 4

- **Application Requirements Engineer**
  - Development and maintenance of the requirements for a single product
  - Use present requirements, if not available create new application specific ones that are validated against the PM
  - Submit suggestions for candidate domain requirements
  - Application RE -> supplies selected requirements application architect and developer, and asset manager gets list for CM purposes

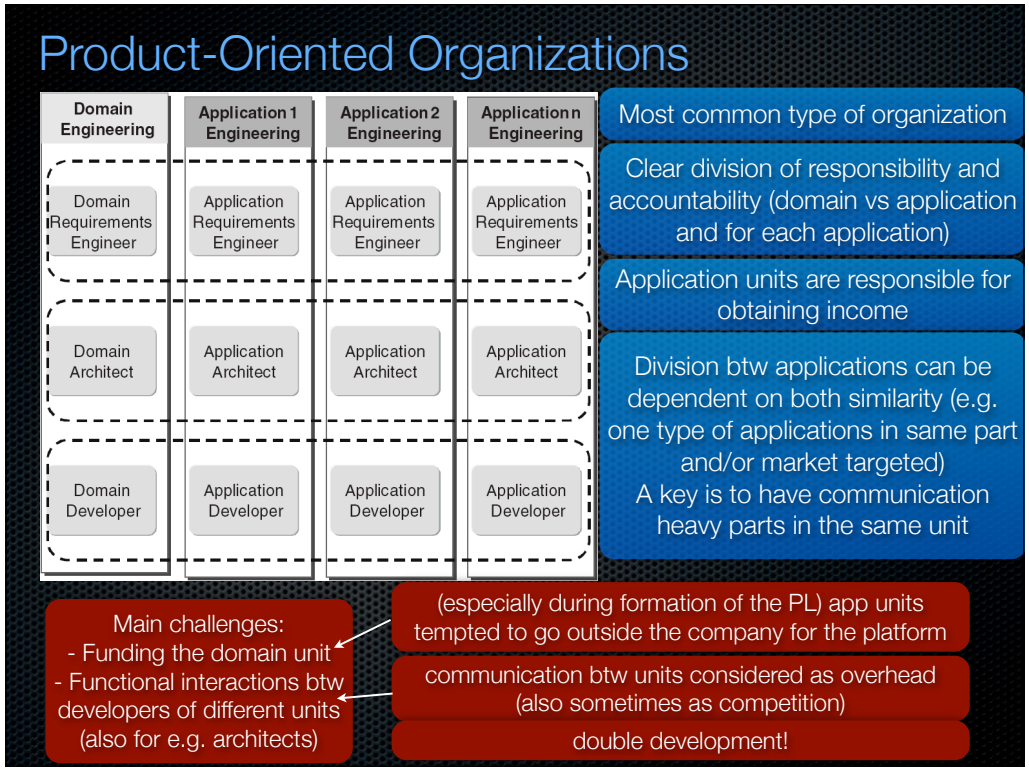
- **Application Architect, Developer, Tester**
  - Specific application
  - Reuse what is possible from the domain level, develop what is needed for the application level
  - Validate against Domain PM and Architect as to adherence to domain assets and architecture
  - Suggest additions (alterations for new variants) to domain level artifacts
  - Early estimation of impact and cost (short and long-term) - not only development but product line impact and cost...

Organizational structures

- The way people interact can be captured in communication patterns. The patterns determine what kinds of mechanisms are used for communication and by whom
- Communication patterns are influenced by organizational structure, as it dictates what information needs to be communicated to whom, and who is concerned with what part (functionality wise) and aspect (life cycle perspective)
- Organizational structures for PL are linked with roles and responsibilities:
  - Domain and Application engineering - go through a development life-cycle (sequence or in parallel)
  - Interactions btw domain and application engineering are on functional level (requirements, design, realization, test level)
  - Domain asset manager interacts with most engineering roles
  - Product Manager provides input to domain engineering and initiates application engineering

domain and application engineering and their interaction influence organizational structure the most

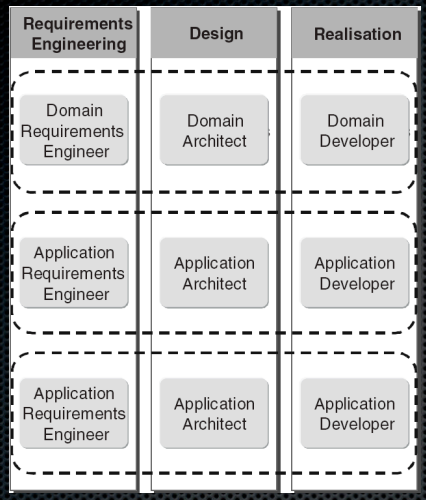
PM, Asset manager, testing lead to additional structure



Funding: budgetpressure... application units tempted to choose other company to provide domain (base)...
(especially initially when forming the PL, then after the domain part is so adapted to the apps that the apps cant find a better match

Interactions: communication btw units -> overhead, addition of additional structure - can be compensated by accepting some overhead + formation of functional units

Process-Oriented Organizations



Functional hierarchy is prime!

Functional interaction is facilitated

Flexible allocation of resources depending on need (btw application but also btw domain and application)

People develop similar functionality for different products:
- Easier to ensure integrity of architecture
- Focus on reusability as it benefits you...

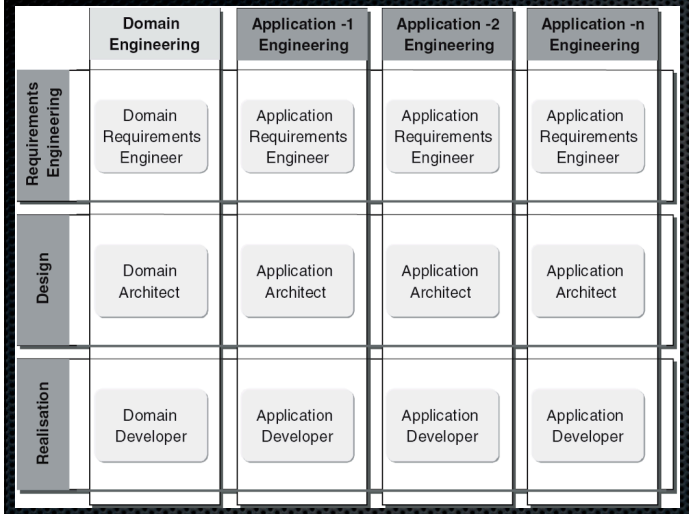
more common in smaller organizations where communication is less of a problem

Main challenges:
- Different phases of engineering are not close
- Domain engineering spread out

communication btw units and planning is necessary

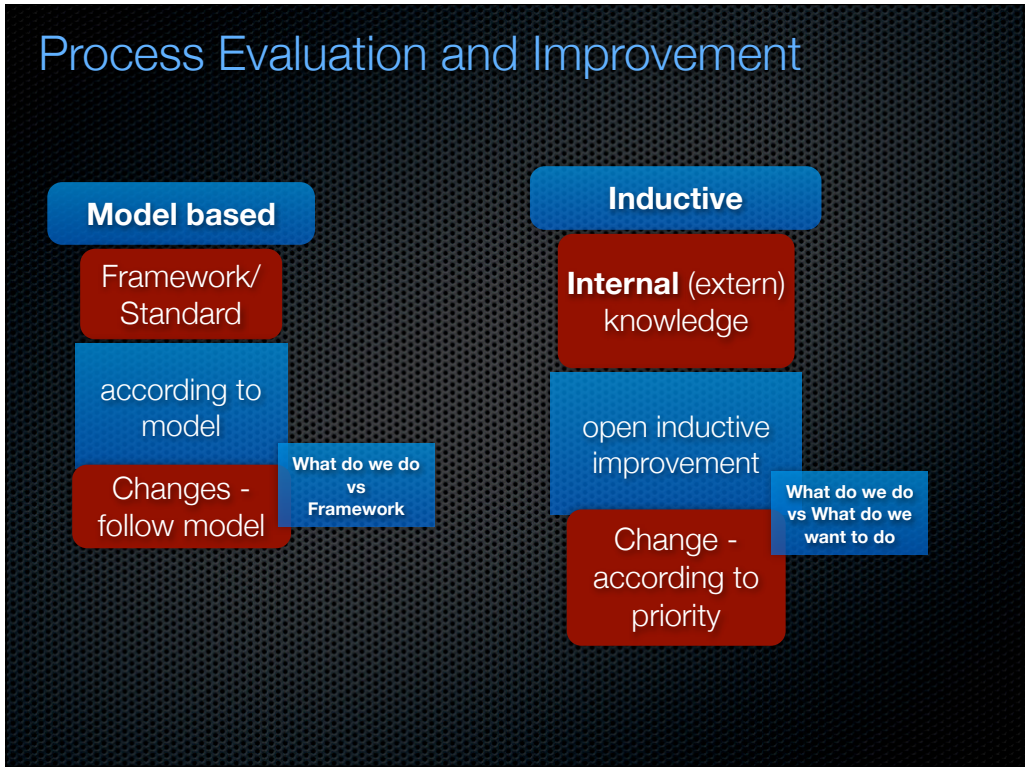
accountability (especially for domain assets is not clear)

Matrix Organizations



Compromise btw product and process focus

Main challenges:
- Scattered focus
- Complex management


Process Evaluation and Improvement
Model based
Framework/ Standard
according to model
Changes - follow model
What do we do vs Framework
Inductive
Internal (extern) knowledge
open inductive improvement
Change - according to priority
What do we do vs What do we want to do

Process Evaluation and Improvement 2

Model based

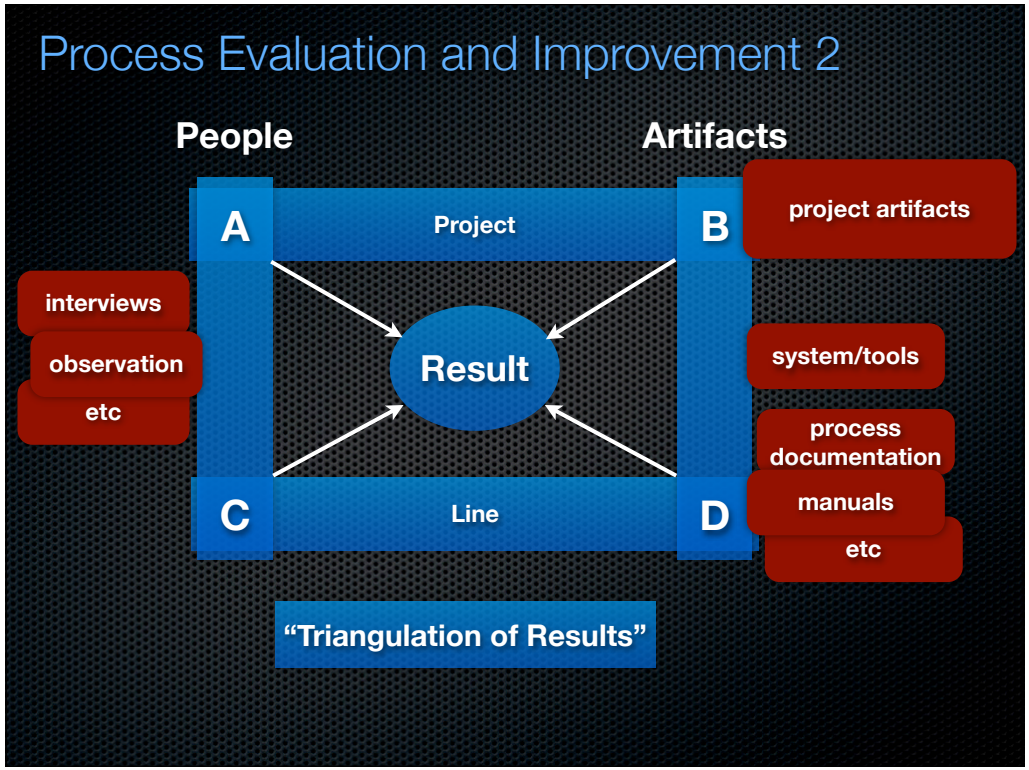
+ external knowledge
+ pre-packaged
+ best practices
- top down
- fit (generic)
- superfluous parts
- priority set

CMM/CMMI
SPICE
ISO

Inductive

+ adapted to the organization
+ only what is needed
+ org. priority
+/- learning process
+ up-down, down-up
- internal knowledge
- larger demands on internal commitment

QIP
PDCA
iFLAP


Process Evaluation and Improvement 2
People
Artifacts
A
Project
B
project artifacts
interviews
observation
etc
Result
system/tools
process documentation
manuals
etc
C
Line
D
“Triangulation of Results”

Family Evaluation Framework (FEF)

- Focuses on the evaluation of product lines (focus on aspects relevant to PLs)

BAPO view

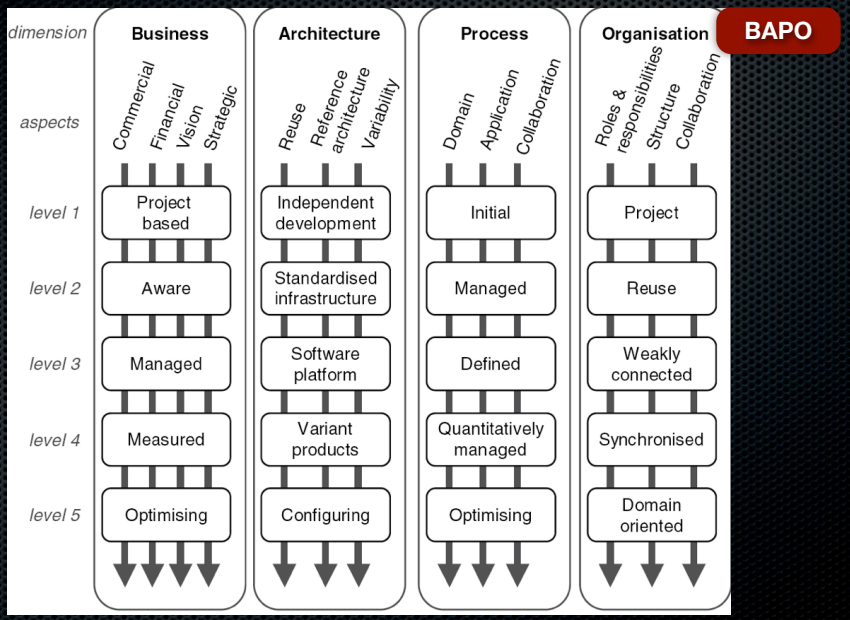
- FEF should be used to evaluate product line organizations (or product line “like” organizations...)

companies that have nothing like a product line = FEF might be a wrong fit

For the case study in this course, see FEF (available on course homepage!) - more detailed than the BAPO paper...

http://trind.dyndns.org/~feldt/cth/sple/papers/linden_2005_fef_intro_and_overview.pdf

Family Evaluation Framework (FEF) 2



Family Evaluation Framework (FEF) 3

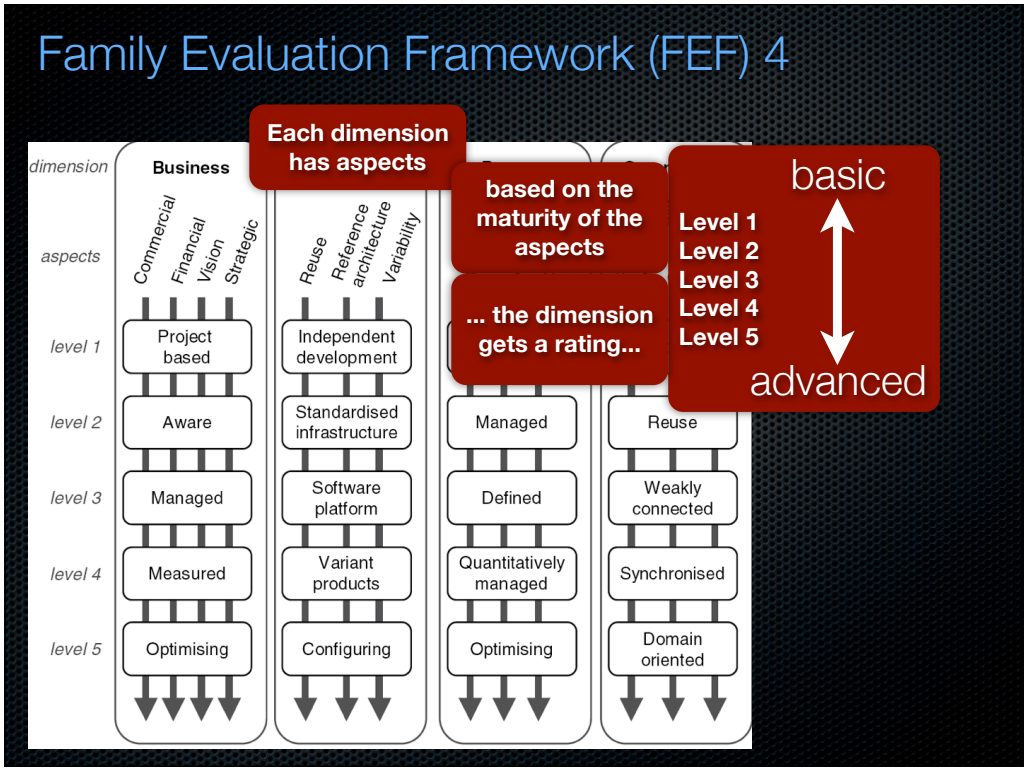
- Business: business involvement in the SPL engineering and variability management. Business relationships between domain and application engineering, and the cost, profits, market value, and planning of variability.

- Architecture: domain and application architecture relations and how they are related via variability.

- Process: process usage and process maturity (use e.g. CMMI)

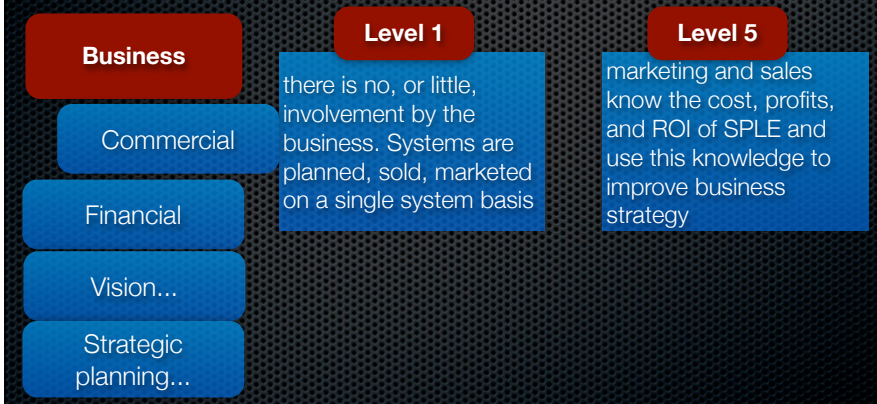
- Organization: effectiveness and distribution of domain and application engineering over the organization. Coordination, communication, how well is the organization suited to PL engineering and to the company

Family Evaluation Framework (FEF) 4

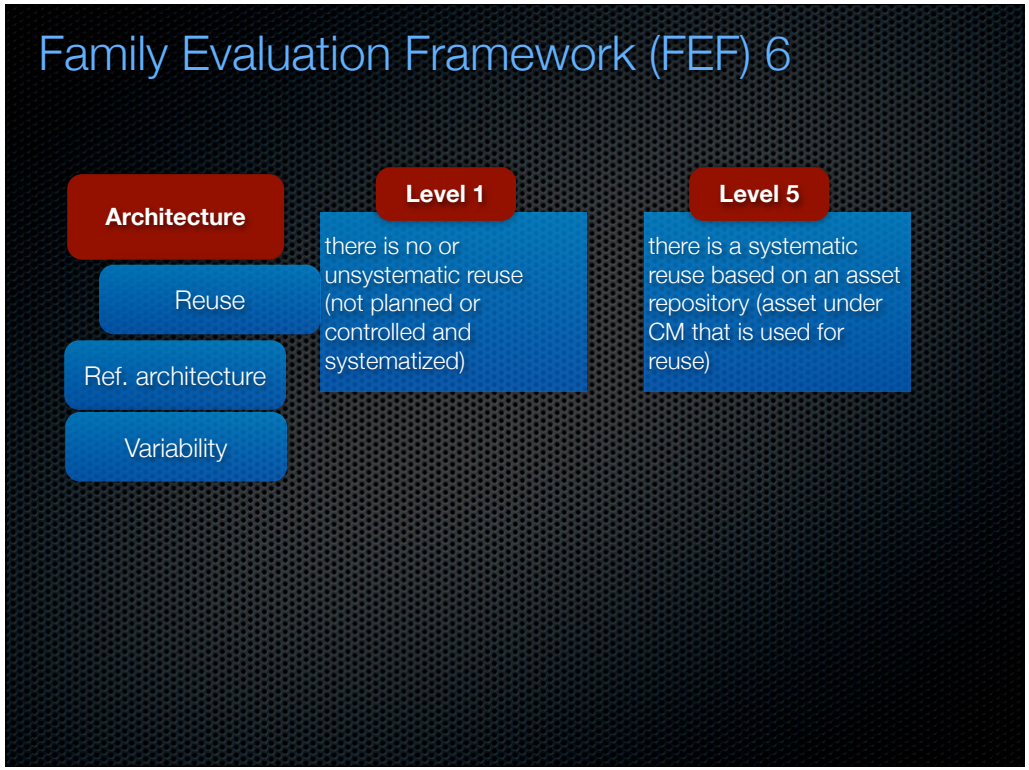


Family Evaluation Framework (FEF) 5

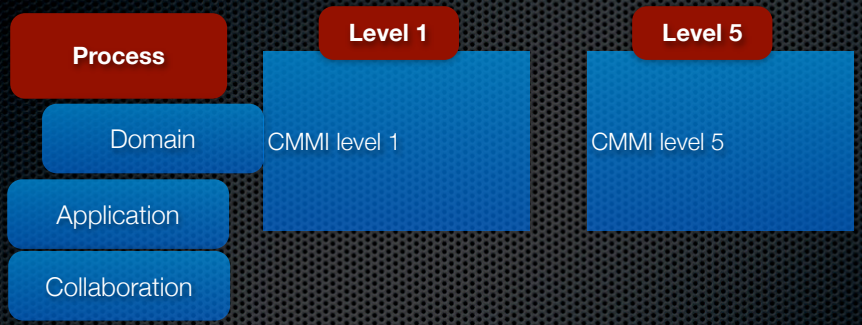
- For each level FEF gives a characterization of the maturity for each aspect.



Family Evaluation Framework (FEF) 6



Family Evaluation Framework (FEF) 7

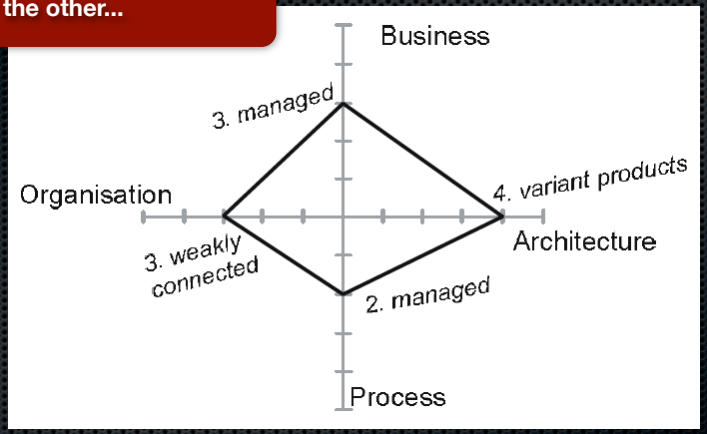


CMMI is used to evaluate the processes used, FEF uses parts of CMMI (and Level 1 in FEF does not always correspond to CMMI Level 1!)

http://www.sei.cmu.edu/cmmi/

Family Evaluation Framework (FEF) 8

balance, one dimension influences the other...



Case study

- Do the evaluation (or suitability analysis) according to relevant framework (see ass. desc.)
- The interview questions, design (e.g. selection of whom you talk to) and how these questions relate to the framework should be mapped.
- The subjects answers (raw data) should also be turned in (appendix).
- Your interpretations of the answers should be a part of the report, e.g. why you judge a certain level
- Some aspects are more suited to other data sources than interviews, but you may use interviews. Bonus if you use triangulation (e.g. confirm in other sources, e.g. two interviews or one interview and documentation)
  - E.g. ask about reuse, get an answer that indicated Level 5, then you look at their asset management and control that the opinion of the interview subject corresponds to reality.
  - E.g. 2: ask two different developers (separate interviews) about reuse, compare answers.
- The interviews you design should be semi-structured to reflect FEF, but do not be leading. Ask follow-up questions to be sure you understand enough to make judgement.