2IS55 Software Evolution

Tests

Alexander Serebrenik

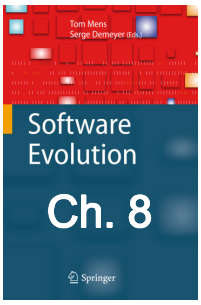
TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

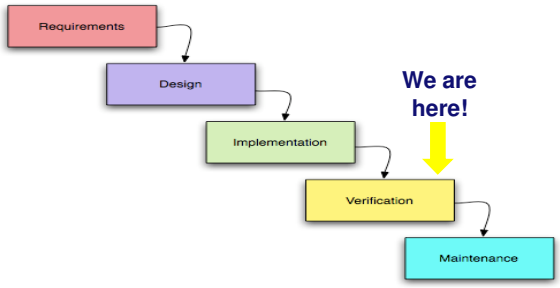
Assignments

- Assignment 6 (Architects and developers)
  - Deadline: Today
    - Some of you got deadline extension till Wednesday
- Assignment 7 (Testing)
  - On Peach
  - Deadline: May 31
    - No class on May 24 (Whit Monday)
- Assignment 8 (Reengineering)
  - To be published on May 31
  - Due to June 21

Sources



Waterfall model [Royce 1970]



Establishing correctness of the program

- Formal verification
  - Model checking, theorem proving, program analysis
  - Additional artefacts: properties to be established
  - Optional artefacts: models
- Testing
  - Additional artefacts: test cases/scripts/programs
  - Optional artefacts: drivers/stubs
- Co-evolution problem: additional (and optional) artefacts should co-evolve with the production code

Different flavours of tests

Testing	Kind of software		
	Management IS	Systems software	Outsourced projects
Unit	10	10	8.5
Integration	5	5	5
System	7	5	5
Acceptance	5	2.5	3

- Effort percentage (staff months) [Capers Jones 2008]
- Evolution research so far focused on unit testing
  - Highest percentage in testing
  - Best-suited for automation

Unit testing

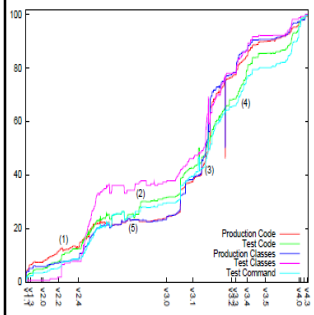
- Test code is also code
  - Recent: unit testing frameworks become popular
- For JUnit code
  - Fixture: common part for multiple tests
    - @Before: set-up, resource claim
    - @After: resource release
  - @Test
- Traditional metrics can be computed
- Compare the evolution of the production code metrics and test code metrics

Examples of co-evolution scenarios [Zaidman et al. 2008]

pLOC	tLOC	pClasses	tClasses	tCommands	interpretation
↗	→				pure development
→	↗				pure testing
↗	↗				co-evolution
→	↗	→	→		test refinement
→	→	↗	↗		skeleton co-evolution
	→		↗		test case skeletons
	→			↗	test command skeletons
→	↘				test refactoring

- p – production code
- t – testing code
- Commands – methods with @Test annotation

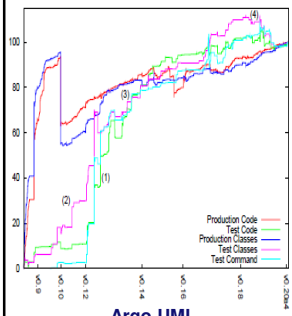
Co-evolution patterns in Checkstyle



Checkstyle, %of maximum

1. Test reinforcement: ↑ #test classes
2. Test refinement
3. Intensive development – testing backlog
4. Back to synchronous testing
5. Test refactoring (↓ tLOC)

Co-evolution patterns in ArgoUML



Argo UML

1. No correspondence between the production code and the test code: pure testing phase
2. Test skeleton introduction
3. Test refinement
4. Test refactorings

"Initial hill" – changes in the VCS leading to code duplication

The diagrams seem to suggest

- Correlation between the size of the test suite size and the production code size
  - Reminder: McCabe's complexity is related to the expected testability effort
  - We are looking at the actual testability effort…

r_s	dLOCC	dNOTC
DIT	-.0456	-.201
FOUT	.465	.307
LCOM	.437	.382
LOCC	.500	.325
NOC	.0537	-.0262
NOF	.455	.294
NOM	.532	.369
RFC	.526	.341
WMC	.531	.348

- Dependent variables
  - dLOCC – LOC per test class
  - dNOTC – number of test cases
- Independent variables
  - FOUT – Class-out
  - WMC – WMC/McCabe

Quantity vs. Quality

- So far: Quantity (tLOC, tClasses, tCommands)
  - BUT how good are the tests?
- Coverage: measure of test quality
  - % program components "touched" by the tests
  - Variants
    - Statement coverage
    - Function/method coverage
    - Module/class coverage
    - Block coverage
      - Block: sequence of statements with no jumps or jumps' targets

Condition coverage vs. Decision coverage

- Condition coverage
  - Every boolean subexpression has been evaluated to true and to false
- Decision coverage
  - In every decision (if/loop) both the true and the false branch have been tested
- Does condition coverage imply decision coverage?
- Does decision coverage imply condition coverage?

Condition coverage vs. decision coverage

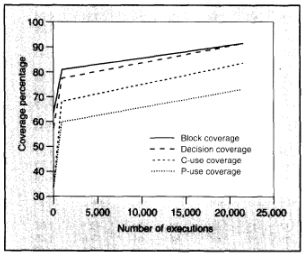
```
int foo(int a, int b) {
  int c = b;

  if ((a>5) && (b>0)) {
    c = a;
  }

  return a*c;
}
```

- { foo(7,-1), foo(5,2) } covers all conditions but not all decisions
- { foo(7,-1), foo(7,1) } covers all decisions but not all conditions

Path coverage

- Path coverage: all possible paths through the given program
  - Unrealistic: n decisions ⇒ up to 2^n different paths
  - Some paths are infeasible
    - Whether a path is infeasible is undecidable
- Coverage implications: path ⇒ decision ⇒ statement
- Special paths: from definition (i = 1) to use (x += i)
  - c-use if the use is a computation (x += i)
  - p-use if the use is a predicate (x < i)

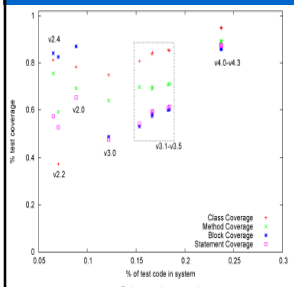
The more you test the better the coverage



Horgan, London, Lyu

- Average over 12 competing versions of the same software
- Coverage increases
  - 100% is still a dream even after more than 20,000 tests!

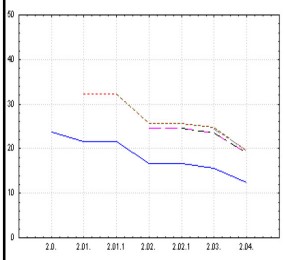
What about evolution of test coverage?



Checkstyle
Abscisse tLOC/(tLOC+pLOC)

- High class coverage (>80% and >95% for 4.*)
  - Exception: 2.2
- 2.*
  - ⇐: pLOC increases faster than tLOC
  - drop in coverage values: major reengineering
- 3.0-4.0: increase for all forms of coverage

Function coverage in bash

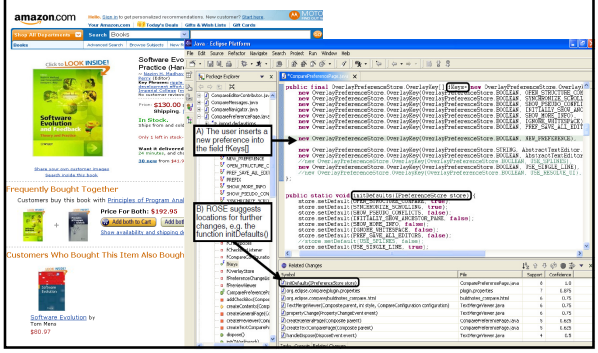


Bash
Elbaum, Gable, Rothermel

- Retrospective analysis: tests for version i were rerun for all versions j, j>i
- Function coverage
- BUT #functions increases and coverage is percentage
  - Consider only functions present in all Bash versions

Closer look at changes

- Remember eROSE? [Zimmermann et al. 2004]



Association Rule Mining

- eROSE is based on detecting frequent sets and association rules, i.e., elements that often are changed together
  - Popular technique: Apriori algorithm (see Lecture 7)
- Tests are code, so [Lubsen, M.Sc. thesis]
  - Distinguish tests/production classes based on their names
    - Drop files that are neither source nor test (makefiles, images, etc.)
  - Use Apriori to mine association rules

Quality of rules: A ⇒ B (A, B – sets)

- Support $|A \wedge B| = P(A,B)$
- Confidence $|A \wedge B| : |A| = P(B|A)$
- Additional values:
  - Lift $P(A,B)/(P(A)*P(B))$
    - Lift = 1 if A and B are independent
    - High Lift – A and B are "indeed" related
  - Leverage $P(A,B) - (P(A)*P(B))$
    - Values > 0 are desirable
  - Conviction $P(A)*P(\text{not } B)/P(A,B)$
    - NB: Asymmetric
- Strong rule: high confidence (and lift) and reasonable support.

Apriori?

- Support $|A \wedge B| = P(A,B)$
  - Support of {X} ⇒ {Y,Z} equals to support of {X,Y} ⇒ {Z} and of {Y} ⇒ {X,Z}, etc.
- First look for sets with a reasonable support, than for the rules with high confidence (lift)
  - Determine frequent item sets
    - If A ∪ B is frequent, then A and B are frequent
  - Generate association rules from frequent item sets

Apriori: frequent item sets

- C_k: Candidate item set of size k
- L_k: frequent item set of size k
- L_1= {frequent items};
- For (k = 1; L_k!=∅; k++) do begin
  - $C_{k+1} = L_k \bowtie L_k$
    - Drop those elements that have a non-frequent subset
  - for each transaction do
    - increment the count of all candidates in C_{k+1} that are contained in t
    - L_{k+1}= candidates in C_{k+1} with minSupport
  - end
- return $\cup L_k$;

Efficiency bomb
Can be improved…

From sets to rules

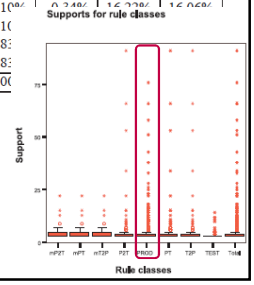
- Given frequent item set X
- Naïve:
  - For each partition Y, Z such that Y ∪ Z = X
    - If confidence(Y⇒Z) < threshold reject;
    - Else report "Y⇒Z"
- However, confidence is $|Y \wedge Z| : |Y|$, so
  - confidence((Y1 ∪ Y2) ⇒ Z) ≥ confidence(Y1 ⇒ (Y2 ∪ Z))
  - Hence, if confidence((Y1 ∪ Y2) ⇒ Z) < threshold, no need to calculate confidence(Y1 ⇒ (Y2 ∪ Z))!
  - Not all partitions of X should be considered
  - Start with one-element long Z…

Rule categorization

- Categorize rules A⇒B (A, B – classes):
  - PROD: A and B are production classes
  - TEST: A and B are test classes
  - P&T pairs:
    - P2T, T2P
    - mP2T, mT2P: matched pairs {*C.java ⇒ CTest.java*}
- Are there any other types of rules we've missed?

Empirical evaluation

Rule Class	Checkstyle	A.I	A.II	B.I	B.II	C.I	C.II
ALL (N)	58566	101896	14590	8820	219248	27308	498
PROD	98,86%	35,15%	49,64%	39,00%	99,12%	51,84%	40,96%
TEST	0,48%	26,11%	9,95%	24,81%	0,20%	9,99%	16,87%
P&T	0,67%	38,75%	40,25%	36,19%	0,69%	32,44%	32,13%
P2T	0,33%	19,37%	20,12%	18,1			
T2P	0,33%	19,37%	20,12%	18,1			
mP2T	0,09%	0,78%	0,74%	0,8			
mT2P	0,09%	0,78%	0,74%	0,8			
UNDEF	0,00%	0,00%	0,16%	0,0			

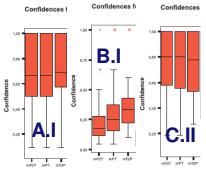


- Checkstyle:
  - Large number of commits with many production classes
    - Classes are together by chance
    - Support is very low
  - Commits on test classes involve only few of them

Empirical evaluation

Rule Class	Checkstyle	A.I	A.II	B.I	B.II	C.I	C.II
ALL (N)	58566	101896	14590	8820	219248	27308	498
PROD	98,86%	35,15%	49,64%	39,00%	99,12%	51,84%	40,96%
TEST	0,48%	26,11%	9,95%	24,81%	0,20%	9,99%	16,87%
P&T	0,67%	38,75%	40,25%	36,19%	0,69%	32,44%	32,13%
P2T	0,33%	19,37%	20,12%	18,10%	0,34%	16,22%	16,06%
T2P	0,33%	19,37%	20,12%	18,10%	0,34%	16,22%	16,06%
mP2T	0,09%	0,78%	0,74%	0,83%	0,01%	0,78%	4,82%
mT2P	0,09%	0,78%	0,74%	0,83%	0,01%	0,78%	4,82%
UNDEF	0,00%	0,00%	0,16%	0,00%	0,00%	5,73%	10,04%

- A.I, A.II, C.I and C.II (synchronous co-evolution)
  - the ratios correspond to the effort distribution.
  - the confidence of typical rules is not low.



Questions

- Apriori algorithm usually works for A and B as sets of elements rather than individual elements:
  - Age > 52, CurrentAcc = true ⇒ Income > 43759, SavingsAcc = true
- Why did Lubsen consider only pairs of classes?

Assignment 7 (2 students, due to May 31)

- Take a system using JUnit (e.g., Shindig)
- Based on the log-file mine association rules
  - Preprocess the log
  - Miners: Weka, XLMiner, …
  - Algorithms: Apriori, FPGrowth, …
  - Parameters: "pairs of classes", minConfidence, minSupport, …
- Evaluate the results
  - Distribution over the PROD,TEST,P&T,UNDEF
  - Distribution of quality values (e.g. lift) for each category
    - Visualize: box plot, histogram, kernel density est.
    - Aggregate: Gini/Theil or fitted distribution

More than JUnit

- There exist JUnit-like systems for
  - Server-side code: Cactus http://jakarta.apache.org/cactus/
  - Web-applications: HttpUnit http://sourceforge.net/projects/httpunit/
  - Popularity?
  - No research so far (AFAIK)

Conclusions

- Verification ⇒ Testing ⇒ Unit testing
  - Other options are practically unstudied
- Unit testing – another group of code files
  - Traditional metrics are applicable
    - Correlation, co-evolution patterns
  - Coverage metrics
  - Association rules

2IS55 Software Evolution

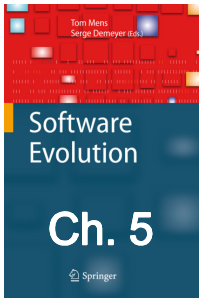
Implementing evolution: Reengineering

Alexander Serebrenik

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Sources



Slides of Rainer Koschke (in German) http://www.iste.uni-stuttgart.de/ps/Lehre/reengineering/neu/transformationen.pdf

So far…

- We assumed that the evolution has already taken place.
- Remainder of the semester: how to implement evolution
  - Reengineering of legacy systems
    - Towards OO, aspects, services
  - Refactoring and its impact
  - Database migration

Evolution strategies

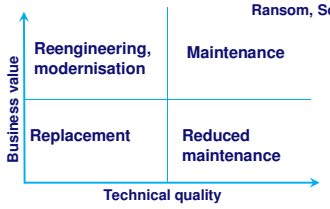
- Refactor
- Reengineer
  - E.g., using models (see next slides)
- Re-implement

Questions

- How can one decide which strategy to follow?
- How can/should one implement the chosen strategy?
- What impact does the implementation have on the co-evolving artifacts?

First look at reengineering decision making

Ransom, Sommerville, Warren



- Both technical and business aspects
- Scale is rather vague

Value-Based Decision Model [Visaggio 2000]

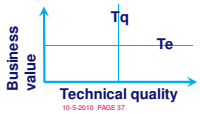
- Metrics to assess technical quality and business value

Examples	Business value	Technical quality
Objective	• Input volume • %input that can be automatically processed	• Constants • OS calls • DB queries/update
Subjective (expert opinion)	• Importance • Fitness for purpose	• Adaptability • Comprehensibility • Correctness • Efficiency

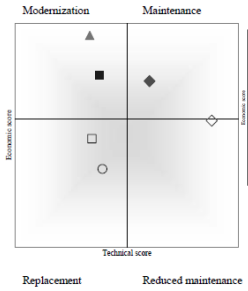
- Each metrics has a threshold B and a weight w (importance)

Value-Based Decision Model

- Divide the system in logical subcomponents
  - Different subcomponents ⇒ different evolution strategies
- Calculate the metrics for each subcomponent
- Aggregate them using thresholds and weights
  - Technical quality of component i $Sq_i = \sum_j \frac{B_j}{m_{ij}} w_j$ where $Tq = \sum w_j$ distinguishes high quality components from the low quality components
  - Similar formula can be given for business values

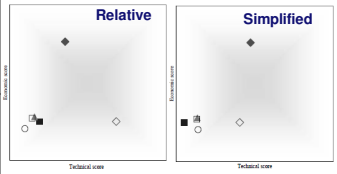


Empirical validation of VDM [Tilus et al.]



Gray – system A, white – system B

- Alternative approaches:



- Agreement: ○ and □ should be replaced, ◆ should be maintained
- Disagreement: experts always preferred VDM

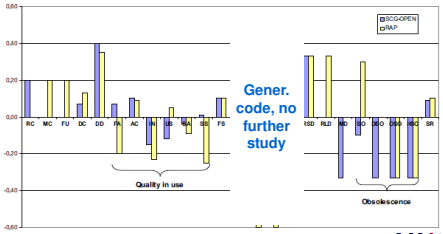
Problem with VDM: To the man with a hammer, everything looks like a nail

- Different problems require different solutions
- Critique table [Aversano et al.]
- If a problem (left) is detected, consider using technique (up)

			Reverse engineering	Redocumentation	Reformatting	Control restructuring	Data restructuring	Modularization	Security management	Data migration	User interface migration	Language migration	Platform migration	Architecture migration	Reengineering	Encapsulation	Evolutive maintenance	Corrective maintenance
			X	X				X										
BUSINESS VALUE	ECONOMIC VALUE	Redevelopment cost																
		Maintenance cost								X		X	X	X	X			
		Future utility														X	X	
	DATA VALUE	Data criticism								X								
		Data dependence								X								
		Data quality					X			X								
	QUALITY IN USE	Functional adequacy															X	
		Accuracy													X			X
		Interoperability									X	X	X	X				
		Usability									X						X	
		User satisfaction									X						X	
		Safety							X									
	SPECIALIZATION VALUE	Specialization level															X	
TECHNICAL VALUE	MAINTAINABILITY	Complexity				X						X			X			
		Size				X	X					X			X			
		Analyzability	X	X	X	X	X	X				X						
		Structuredness				X						X				X		
	DEGRADATION	Responsiveness degradation				X								X				
		Reliability degradation													X			
		Maintainability degradation	X	X	X	X	X	X							X			
	OBSOLESCENCE	SW obsolescence											X		X			
		DB obsolescence								X				X				
		OS obsolescence												X				
		HW/SW infrastructure obsolescence											X	X				
	RELIABILITY	SW reliability													X			X

Critique table in practice

- Two case studies: metrics values and desired thresholds obtained by interviews
- Gap analysis: (metrics – threshold)
- Negative bars are important!



Gener. code, no further study

Selecting all relevant techniques can be too much!

			Reverse engineering	Redocumentation	Reformatting	Control restructuring	Data restructuring	Modularization	Security management	Data migration	User interface migration	Language migration	Platform migration	Architecture migration	Reengineering	Encapsulation	Evolutive maintenance	Corrective maintenance
BUSINESS VALUE	ECONOMIC VALUE	Redevelopment cost	X	X				X										
		Maintenance cost								X		X	X	X	X			
		Future utility														X	X	
	DATA VALUE	Data criticism								X								
		Data dependence								X								
		Data quality					X			X								
	QUALITY IN USE	Functional adequacy															X	
		Accuracy													X			X
		Interoperability									X	X	X	X				
		Usability									X						X	
		User satisfaction									X						X	
		Safety							X									
	SPECIALIZATION VALUE	Specialization level															X	
TECHNICAL VALUE	MAINTAINABILITY	Complexity				X						X			X			
		Size				X	X					X			X			
		Analyzability	X	X	X	X	X	X				X						
		Structuredness				X						X				X		
	DEGRADATION	Responsiveness degradation				X								X				
		Reliability degradation													X			
		Maintainability degradation	X	X	X	X	X	X							X			
	OBSOLESCENCE	SW obsolescence										X		X				
		DB obsolescence								X				X				
		OS obsolescence											X					
		HW/SW infrastructure obsolescence											X	X				
	RELIABILITY	SW reliability													X			X

Language migration can improve interoperability but it also affects many "good" attributes

			Reverse engineering	Redocumentation	Reformating	Control restructuring	Data restructuring	Modularization	Security management	Data migration	User interface migration	Language migration	Platform migration	Architecture migration	Reengineering	Encapsulation	Evolutive maintenance	Corrective maintenance
BUSINESS VALUE	NOMIC VALUE	Redevelopment cost	X	X				X										
		Maintenance cost								X		X	X	X	X			
		Future utility														X	X	
	A VALUE	Data criticism								X								
		Data dependence								X								
		Data quality					X			X								
	LITY IN USE	Functional adequacy															X	
		Accuracy													X			X
		Interoperability									X	X	X	X				
		Usability									X						X	
		User satisfaction									X						X	
		Safety							X									
	IALIZATION VALUE	Specialization level															X	
TECHNICAL VALUE		Complexity				X						X			X			
		Size				X	X					X			X			
		Analyzability	X	X	X	X	X	X				X						
		Structuredness				X						X				X		
	DEGRADATION	Responsiveness degradation				X								X				
		Reliability degradation													X			
		Maintainability degradation	X	X	X	X	X	X							X			
	OBSOLESCENCE	SW obsolescence										X		X				
		DB obsolescence								X				X				
		OS obsolescence											X					
		HW/SW infrastructure obsolescence											X	X				
	RELIABILITY	SW reliability													X			X

Answer:

			Reverse engineering	Redocumentation	Reformating	Control restructuring	Data restructuring	Modularization	Security management	Data migration	User interface migration	Language migration	Platform migration	Architecture migration	Reengineering	Encapsulation	Evolutive maintenance	Corrective maintenance
BUSINESS VALUE	ECONOMIC VALUE	Redevelopment cost	X	X				X										
		Maintenance cost								X		X	X	X	X			
		Future utility														X	X	
	DATA VALUE	Data criticism								X								
		Data dependence								X								
		Data quality					X			X								
	QUALITY IN USE	Functional adequacy															X	
		Accuracy													X			X
		Interoperability									X	X	X	X				
		Usability									X						X	
		User satisfaction									X						X	
		Safety							X									
	SPECIALIZATION VALUE	Specialization level															X	
TECHNICAL VALUE	MAINTAINABILITY	Complexity				X						X			X			
		Size				X	X					X			X			
		Analyzability	X	X	X	X	X	X				X						
		Structuredness				X						X				X		
	DEGRADATION	Responsiveness degradation				X								X				
		Reliability degradation													X			
		Maintainability degradation	X	X	X	X	X	X							X			
	OBSOLESCENCE	SW obsolescence										X		X				
		DB obsolescence								X				X				
		OS obsolescence											X					
		HW/SW infrastructure obsolescence											X	X				
	RELIABILITY	SW reliability													X			X

So far: business value and technical quality

- Whatever decision taken, it should fit the goals of the organization

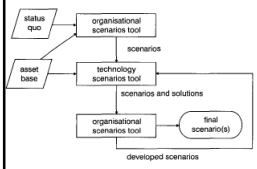
	Scenario 1: Status quo	Scenario 2: Automate	Scenario 3: Infomate	Scenario 4: Transform	...	Scenario n
Boundary						
Vision						
Logic						
Structure						
Roles						
View of information						
Costs						
Benefits						
Risks						

What does the organization want to achieve?

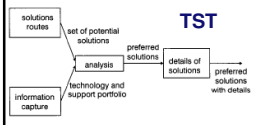
- Organizational scenarios tool
  - costs: major costs, both financial and nonfinancial
  - benefits: both financial and nonfinancial
  - risks: major sources of risk
- Automate: replace with IT
- Infomate: IT augments manual process
- Transform: IT restructures the process

SABA: Bennett, Ramage, Munro

Scenarios and more



TST



- Scenarios state what should be achieved
- TST helps to refine archetypal solutions:
  - leave, discard, rebuild, reengineer
  - based on the specifics of the organization assets:
    - software, staff and process
- "Tools" are not really tools but methods

Validation

	Internal	External
SABA [Bennett, Ramage, Munro]	Has been developed gradually, iteration may be used to explore the consequences	OST: empirically tested
Critique tables [Aversano, Esposito, Mallardo,Tortorella]	Unknown	Two case studies
VDM [Visaggio]	Longitudinal data gathering during method development	Observations of a real-world renewal project (653 programs)

Evolution strategy choice: Summary

- Should take both technical quality and business value into account
  - But also customer satisfaction [Sahin, Zahedi]
- Should suggest a specific technique rather than a generic approach
- Should be iterative [Ransom, Sommerville, Warren]