Systematic Software Improvement

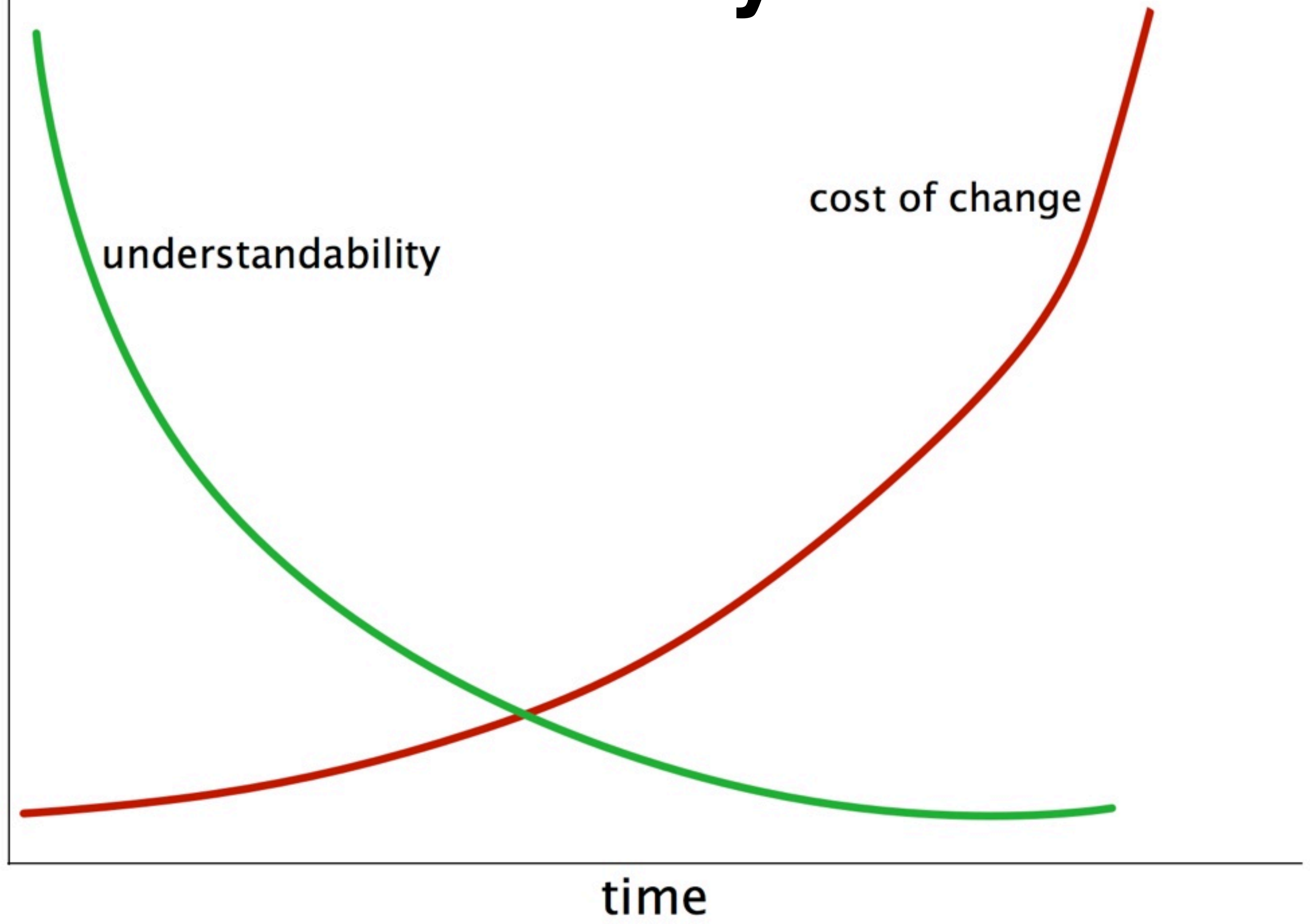
Alexander Heusingfeld

Dr. Gernot Starke

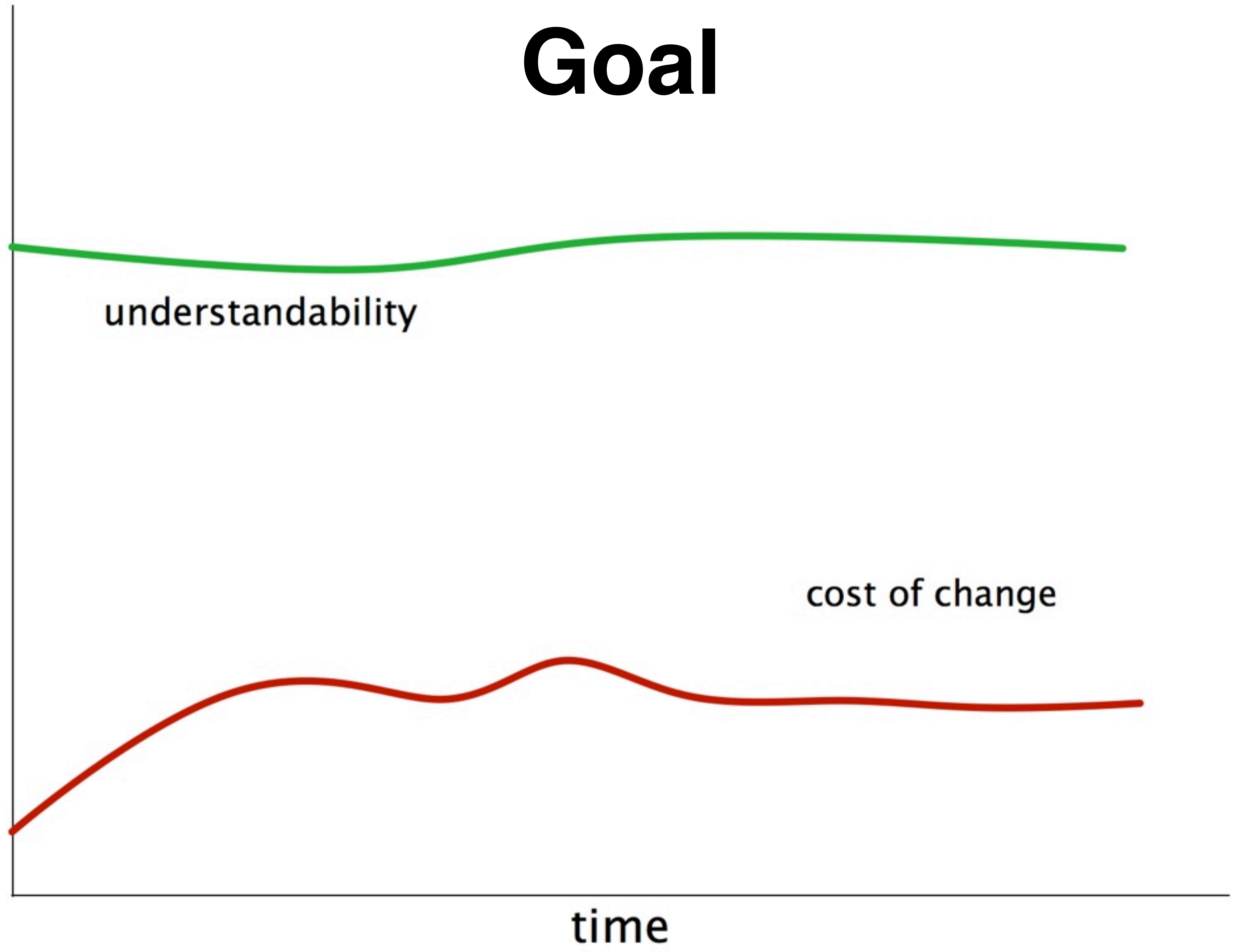
innoQ

aim42

Reality



Goal



Thesis:

Software

Education focused

of systems

on „build-from-scratch“

Thesis:

Business requires more maintenance competence

Thesis:

of Systems

Improvement

of single classes

is more than Refactoring

„Große“ Umbauten bedeuten (oft):

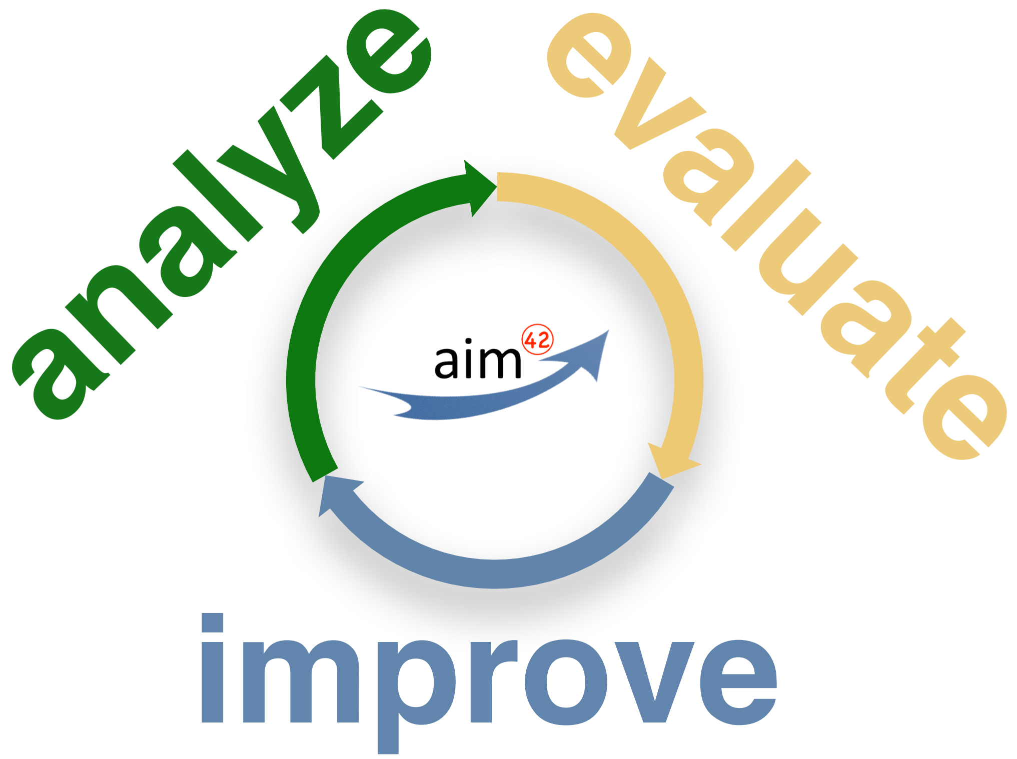
- Umbau DB-Struktur, Datenmigration
- Austausch von Software-Infrastruktur
  (z.B. Frameworks)
- umfangreiche Änderung interner Abläufe
- massive Änderung interner Schnittstellen

Thesis:

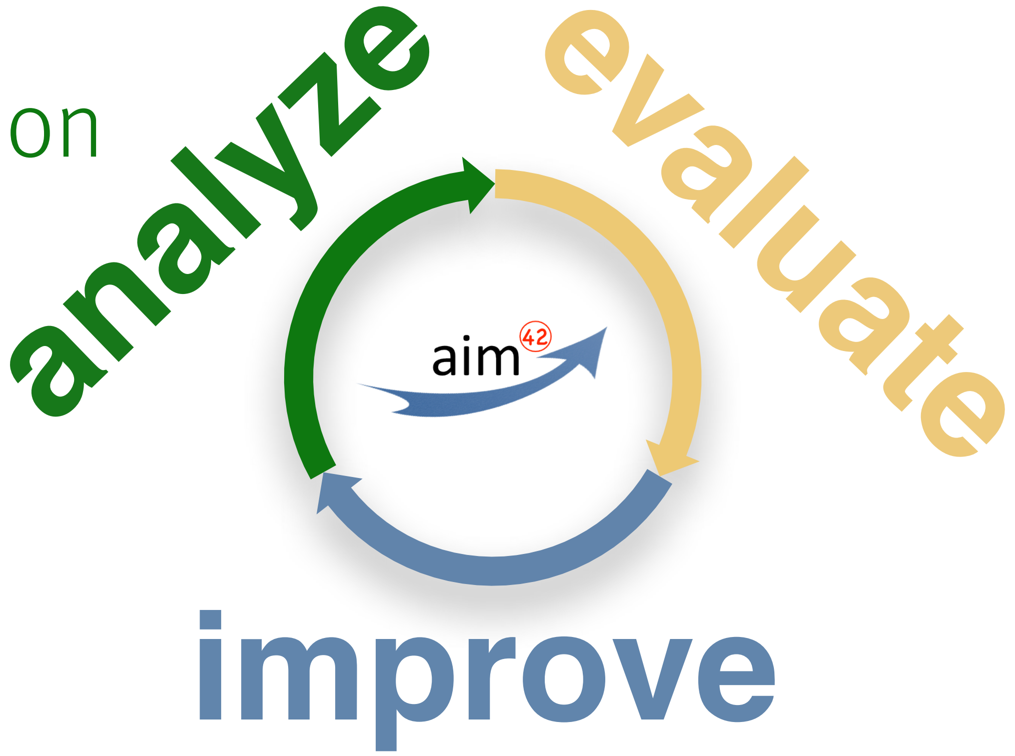
Management responsible for budget ignores architecture principles

aim42

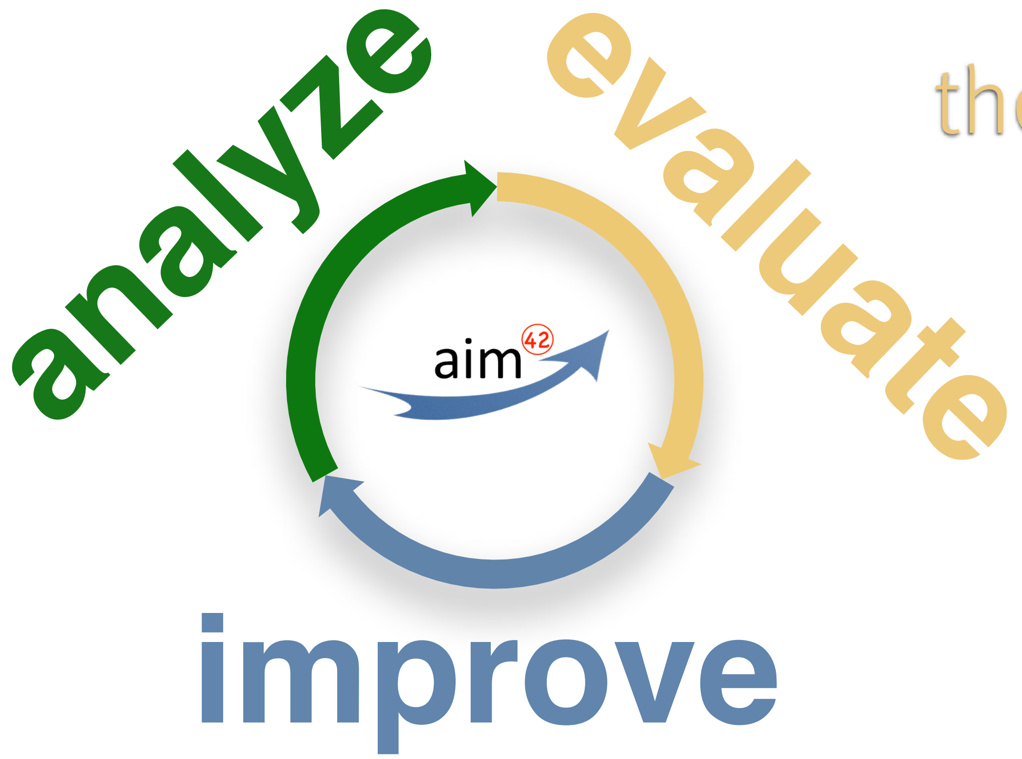
Architecture Improvement Method

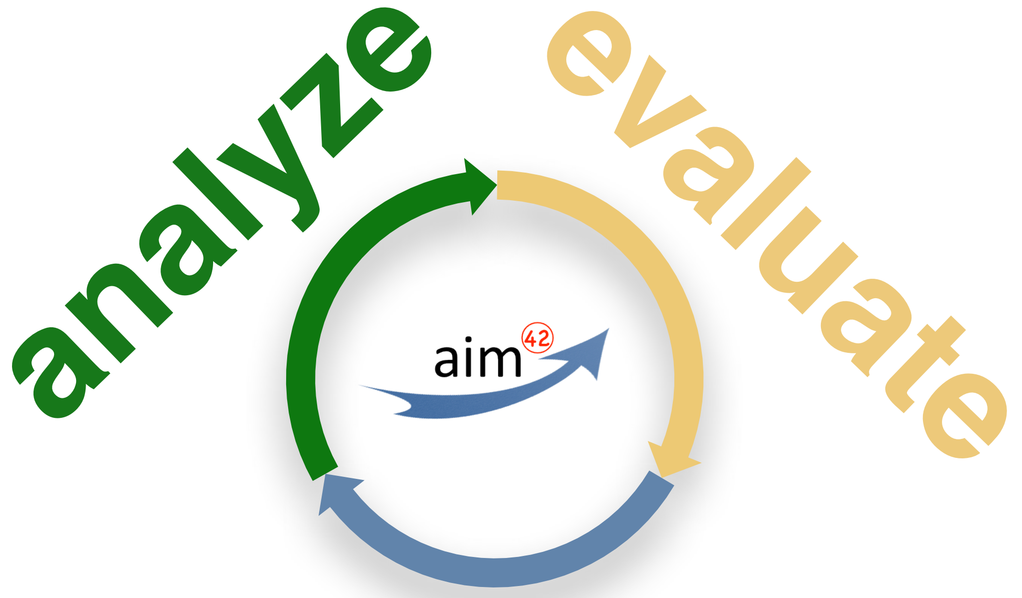

analyze
evaluate
improve
aim42

- architecture
- code
- runtime
- organization



determine „value“ of
problems / risks /
issues <u>and</u>
their remedies

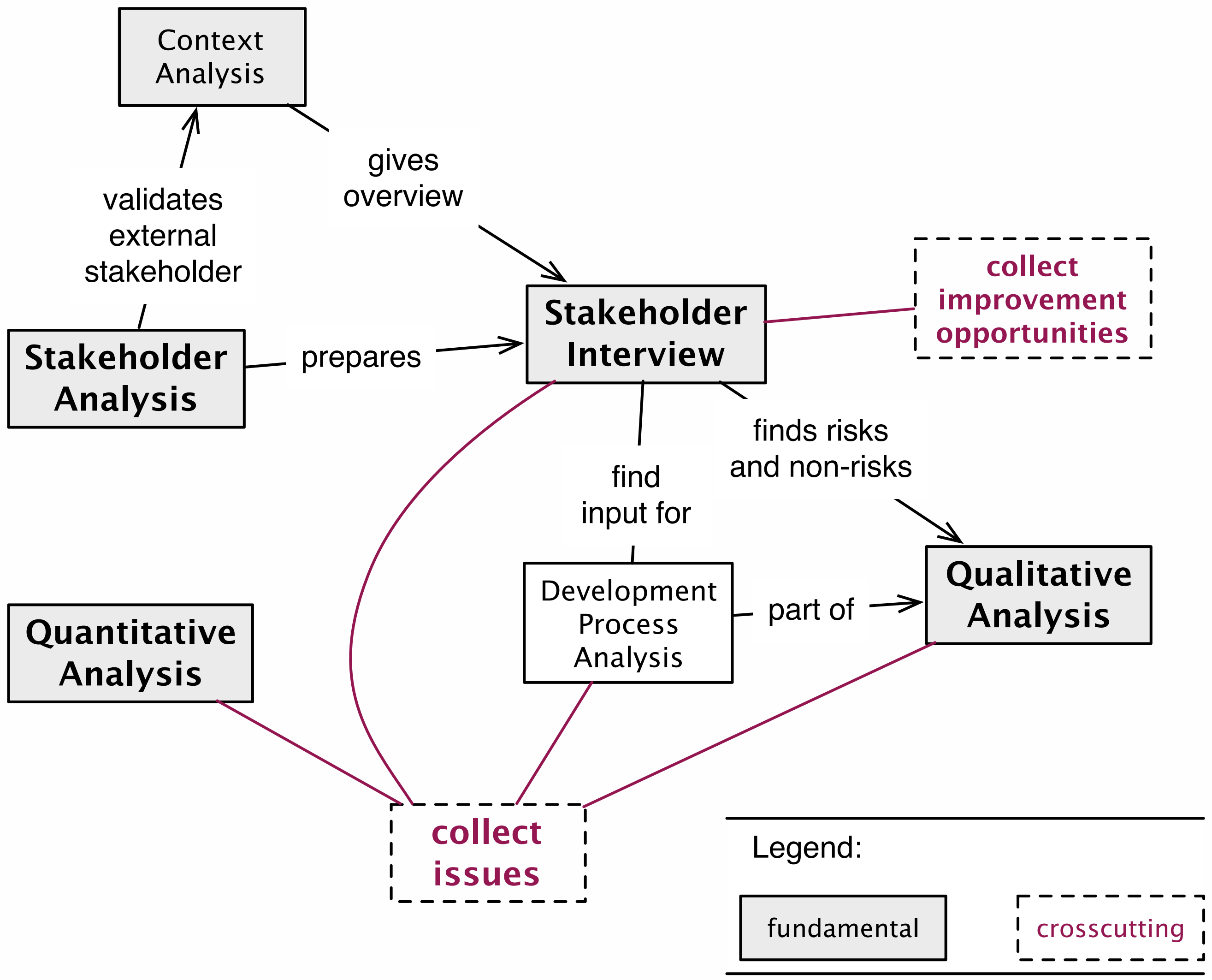




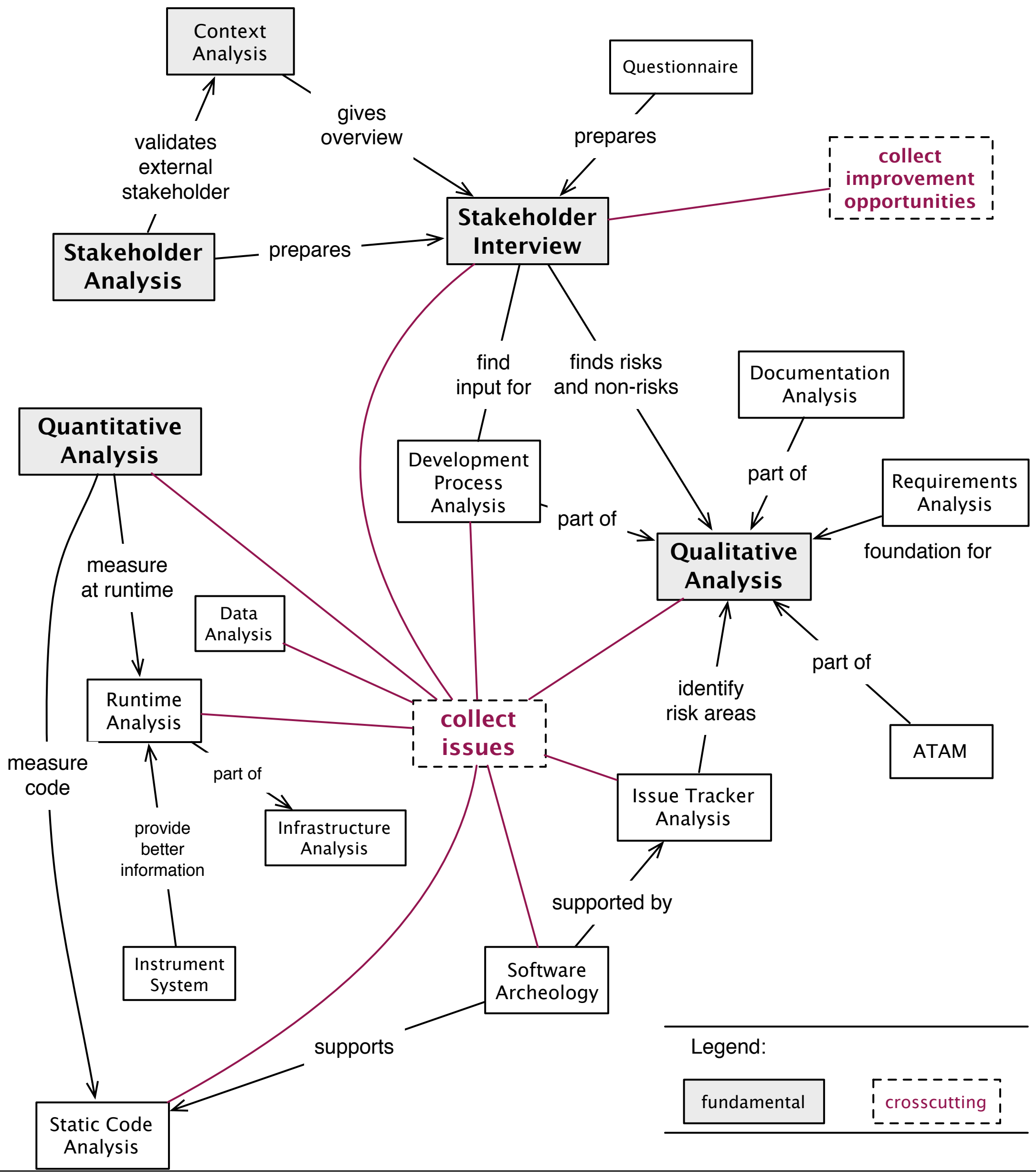
improve

- define improvement strategy
- refactor
- re-architect
- re-organize
- remove debt

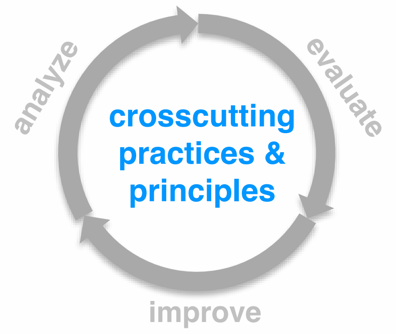
„Analysis“ Overview

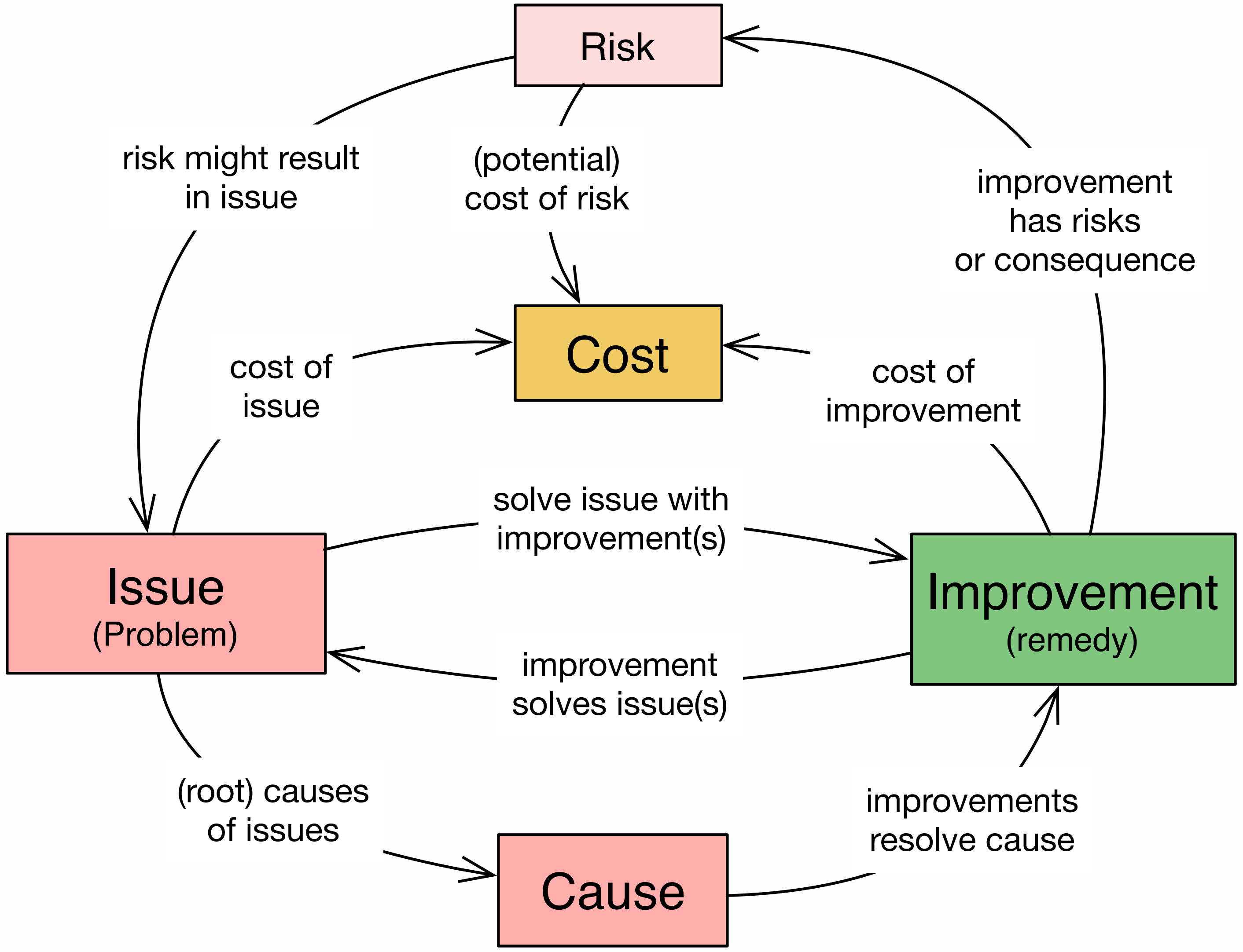


„Analysis“ Details



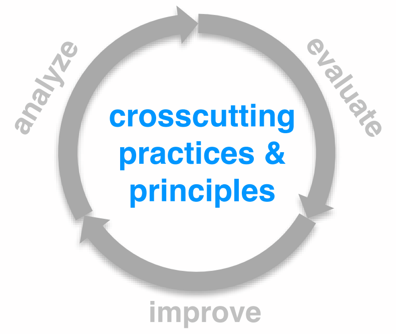
Common Wording

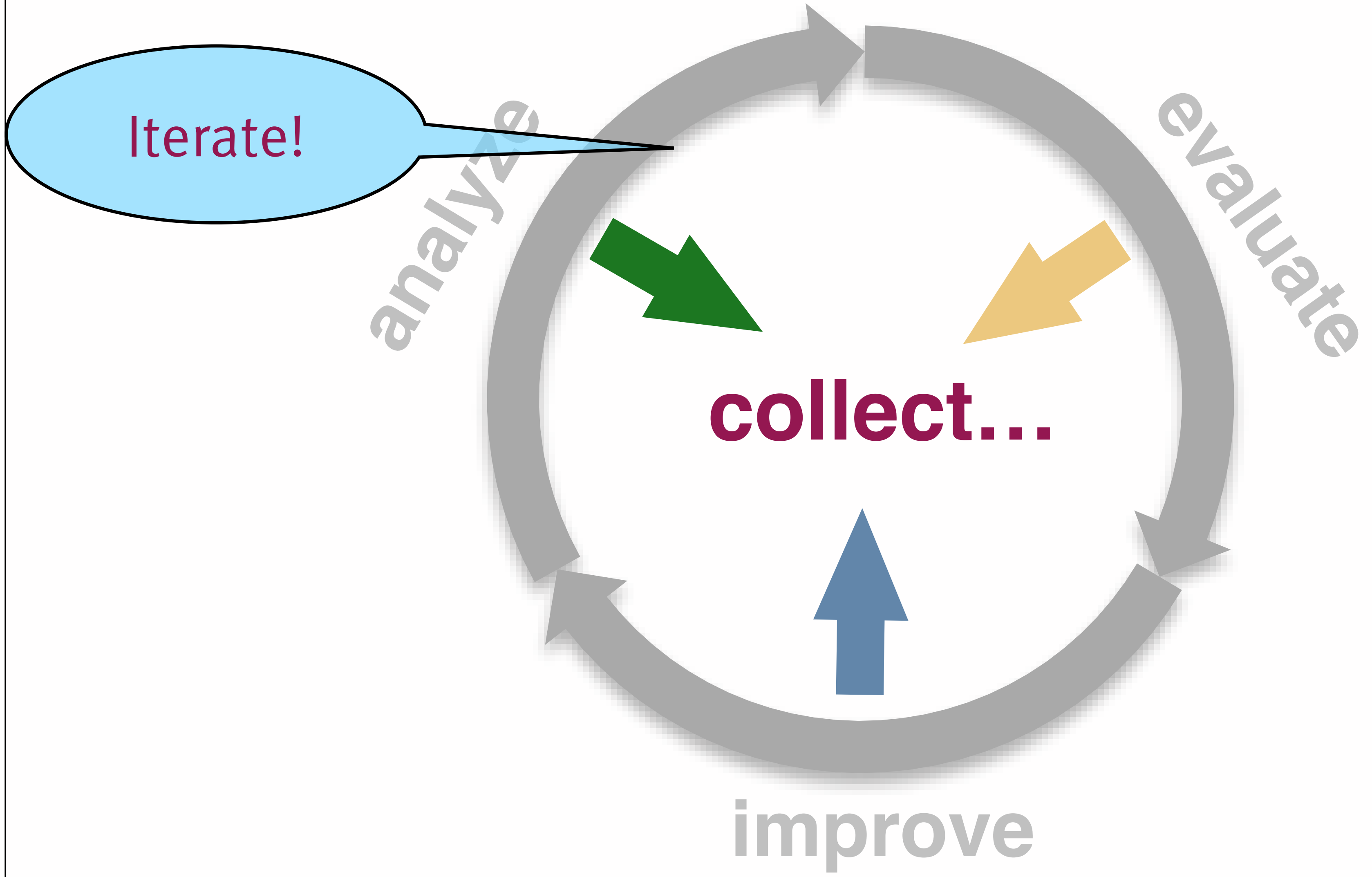




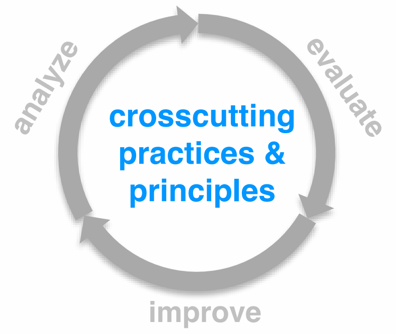
Risk
risk might result in issue
(potential) cost of risk
improvement has risks or consequence
Cost
cost of issue
cost of improvement
solve issue with improvement(s)
Issue
(Problem)
Improvement
(remedy)
improvement solves issue(s)
(root) causes of issues
improvements resolve cause
Cause

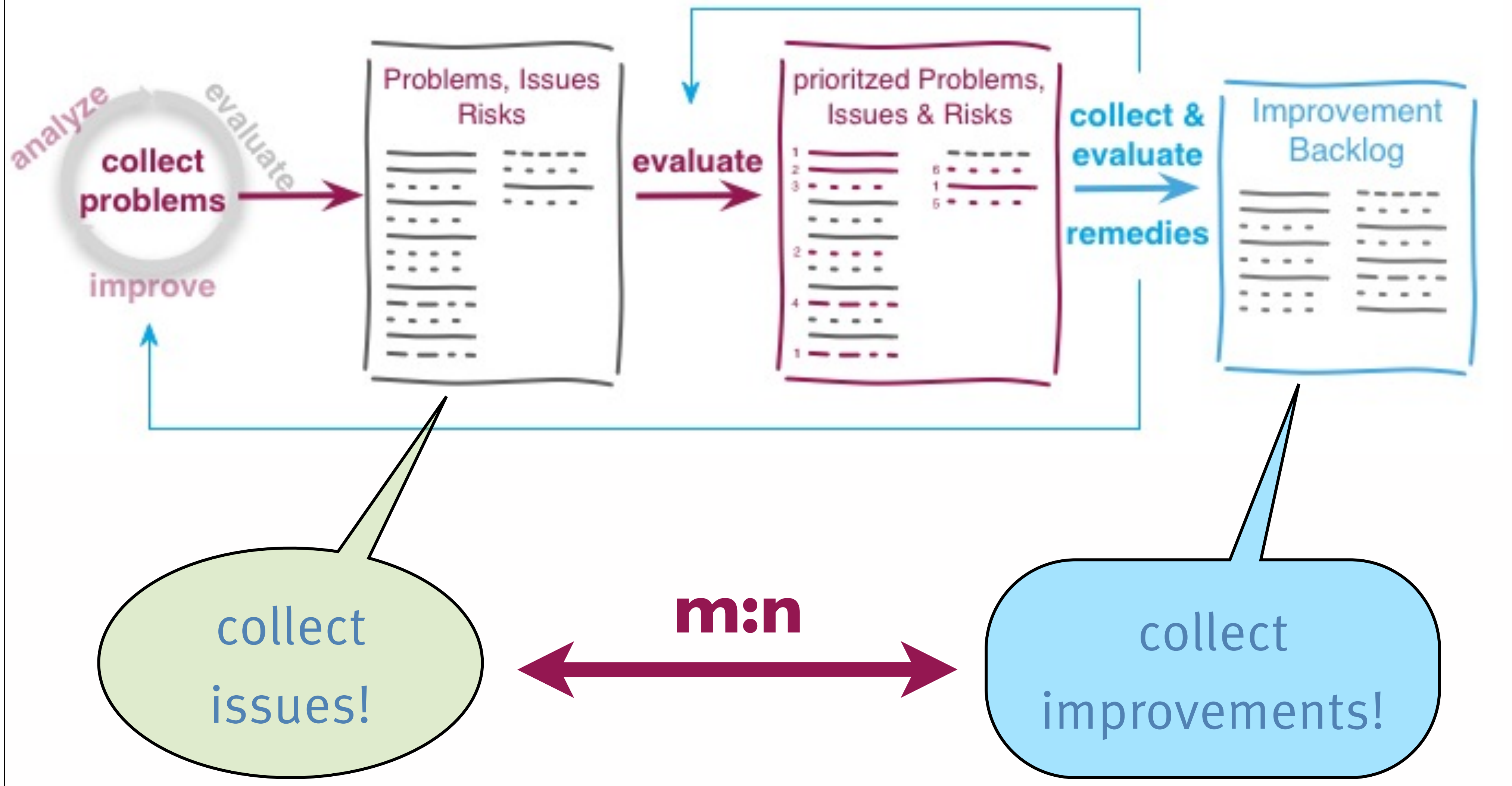
Groundwork (1)



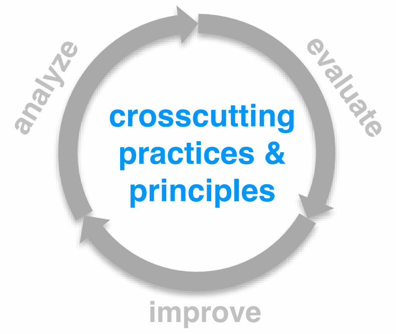


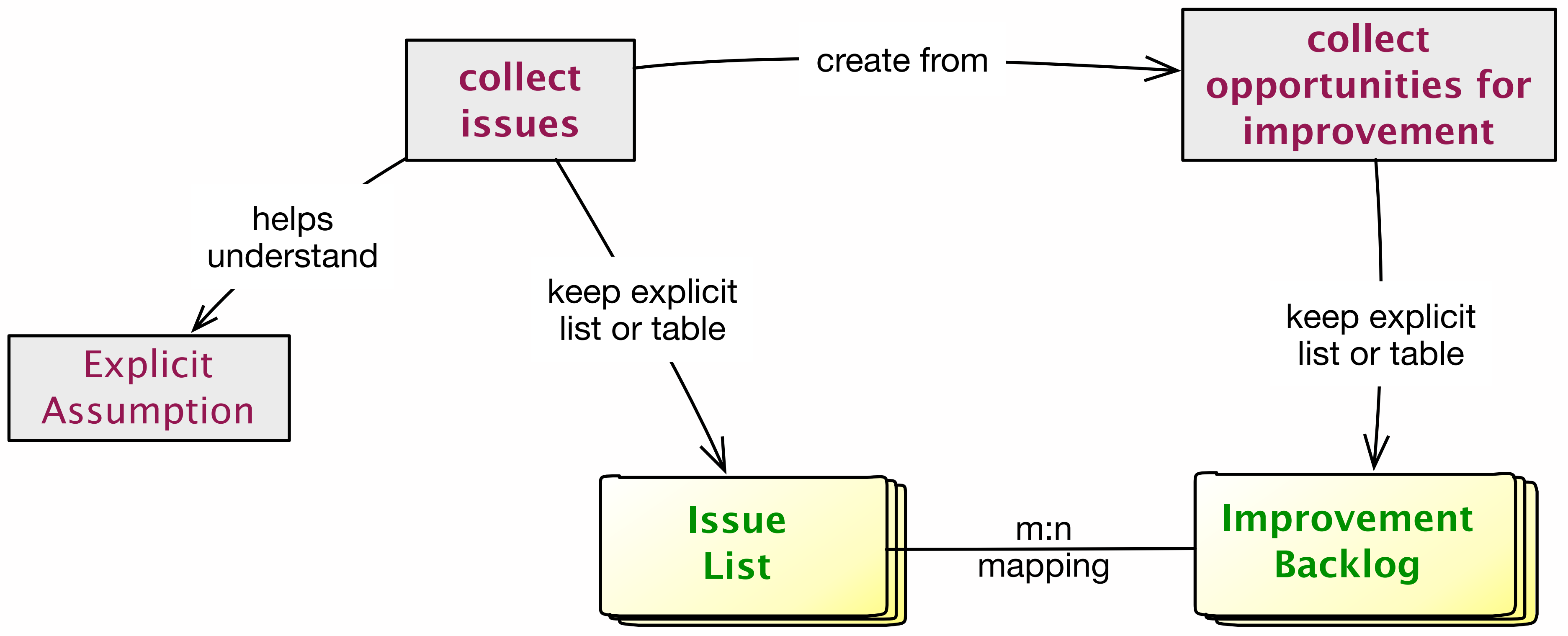
Groundwork (2)



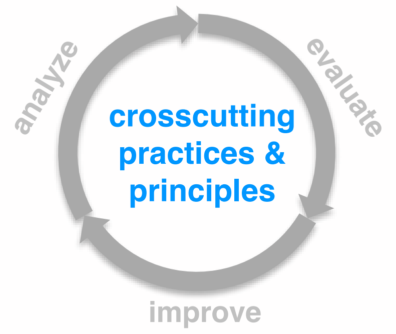


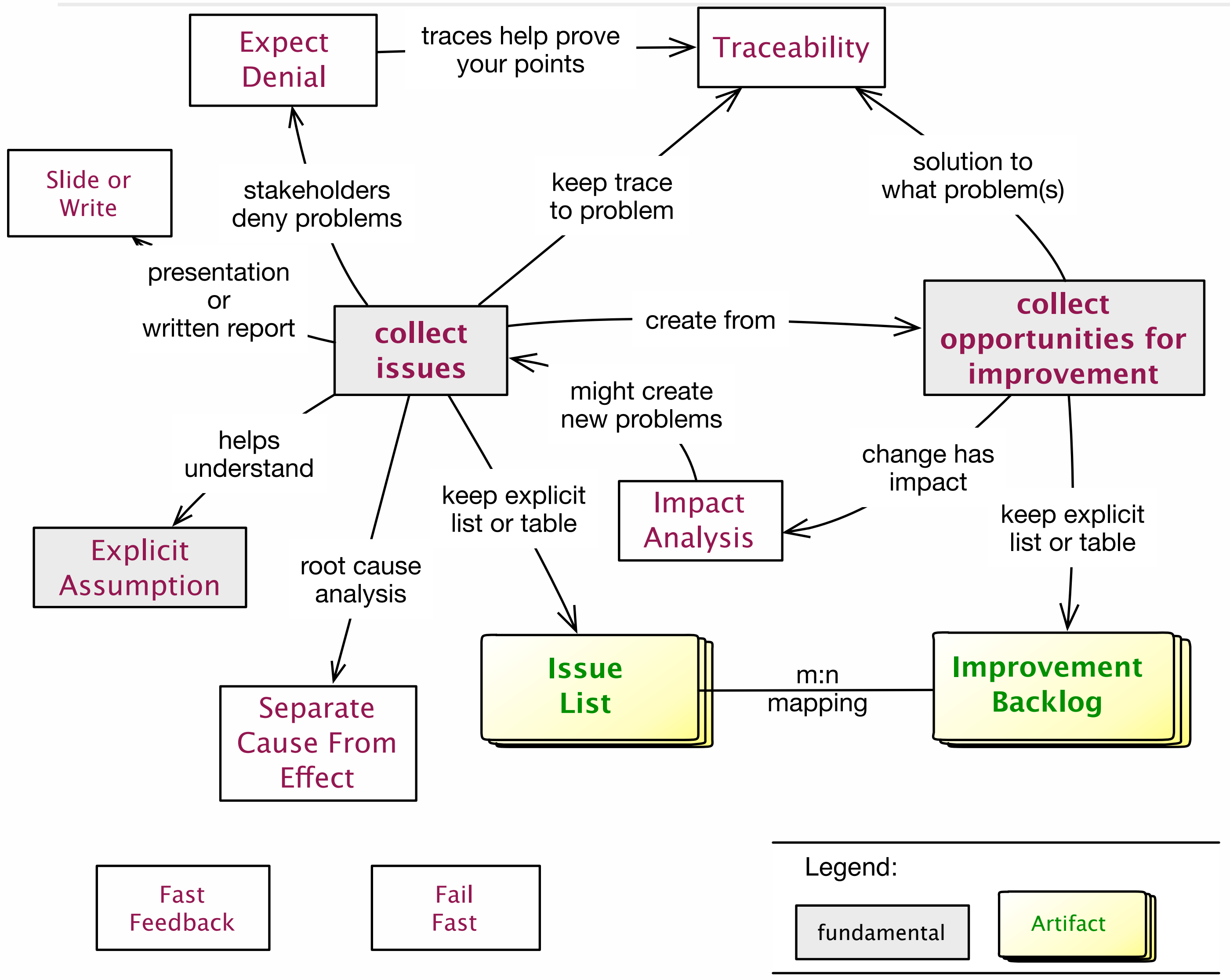
Groundwork (3)



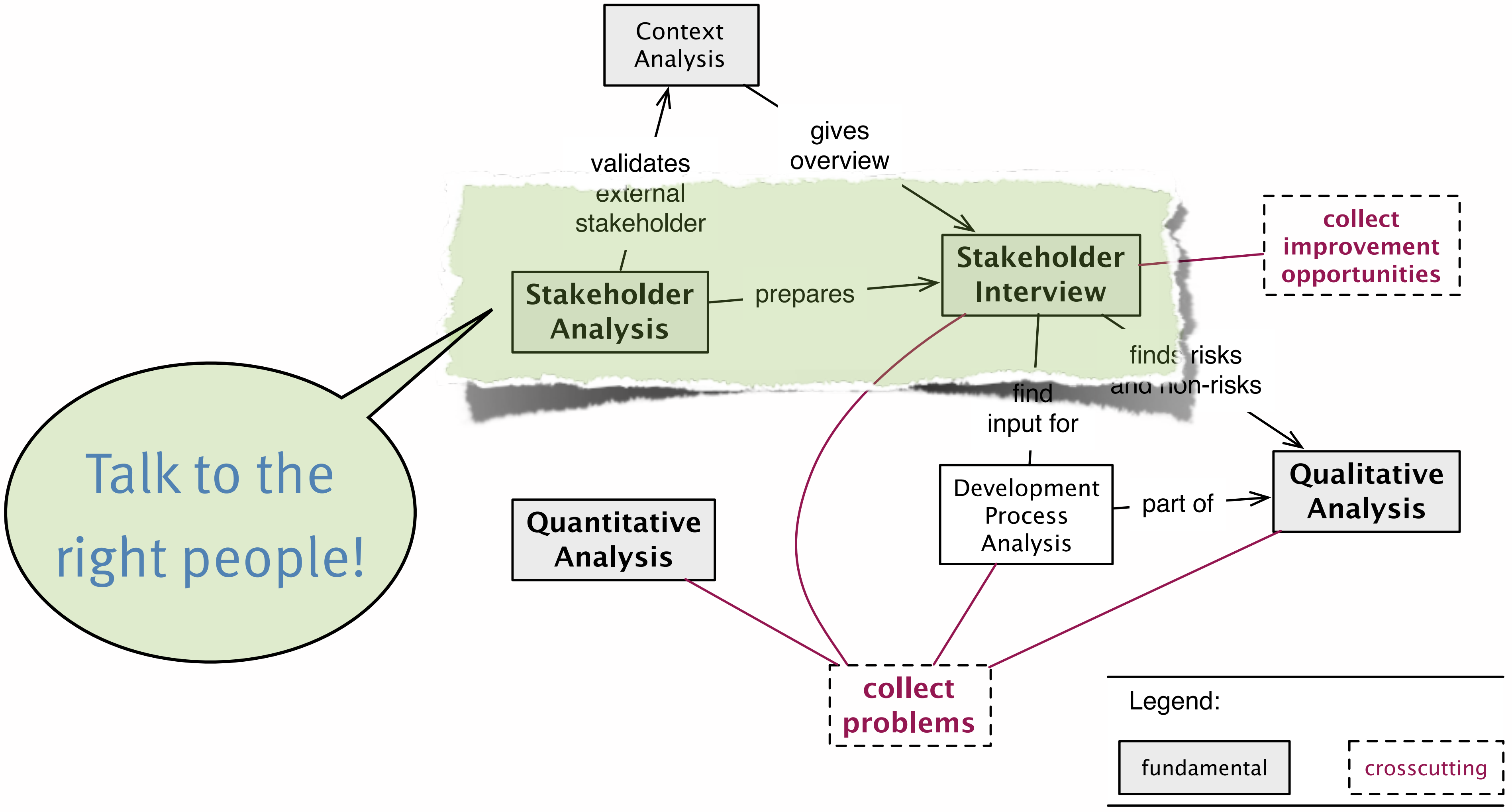


Groundwork (4)

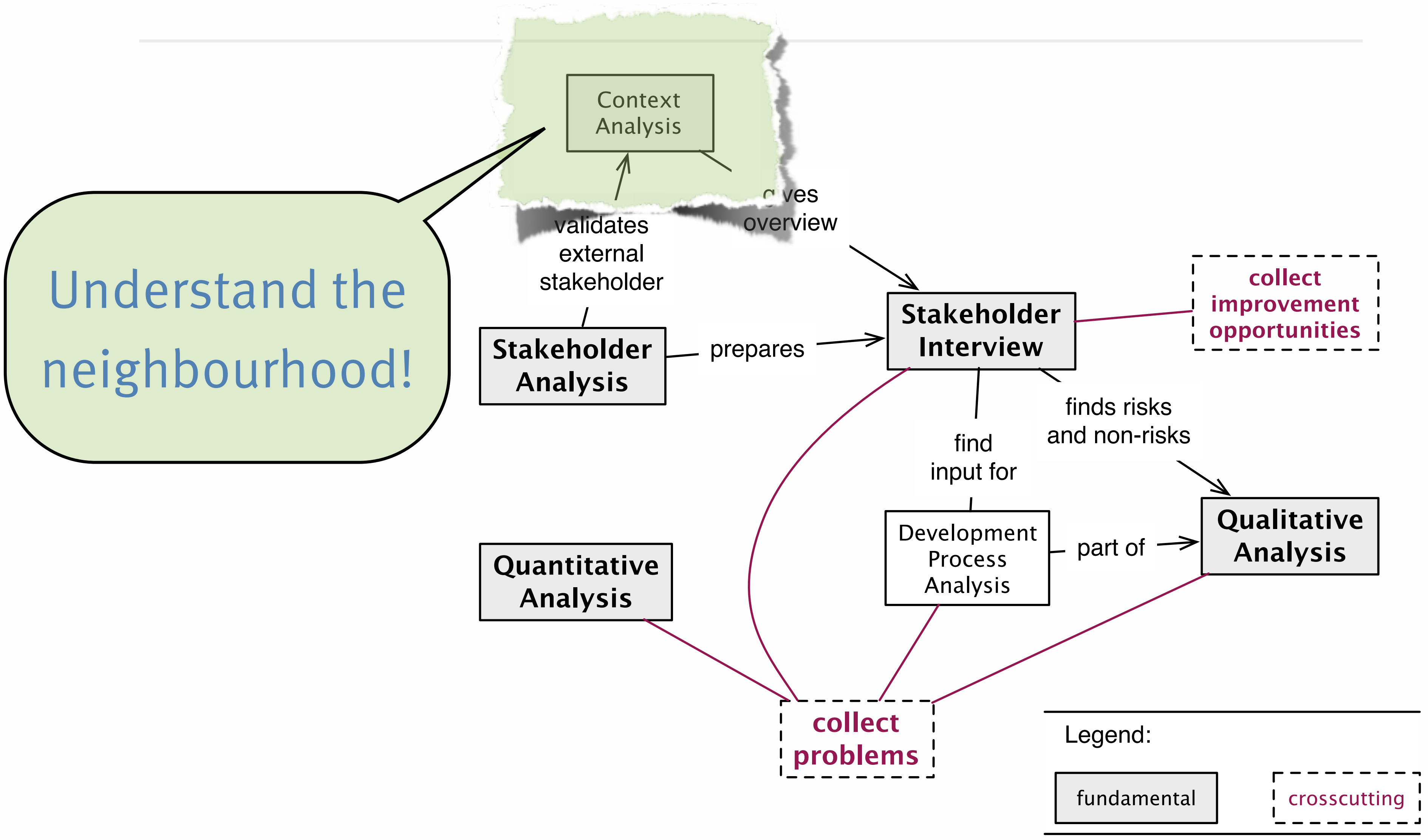




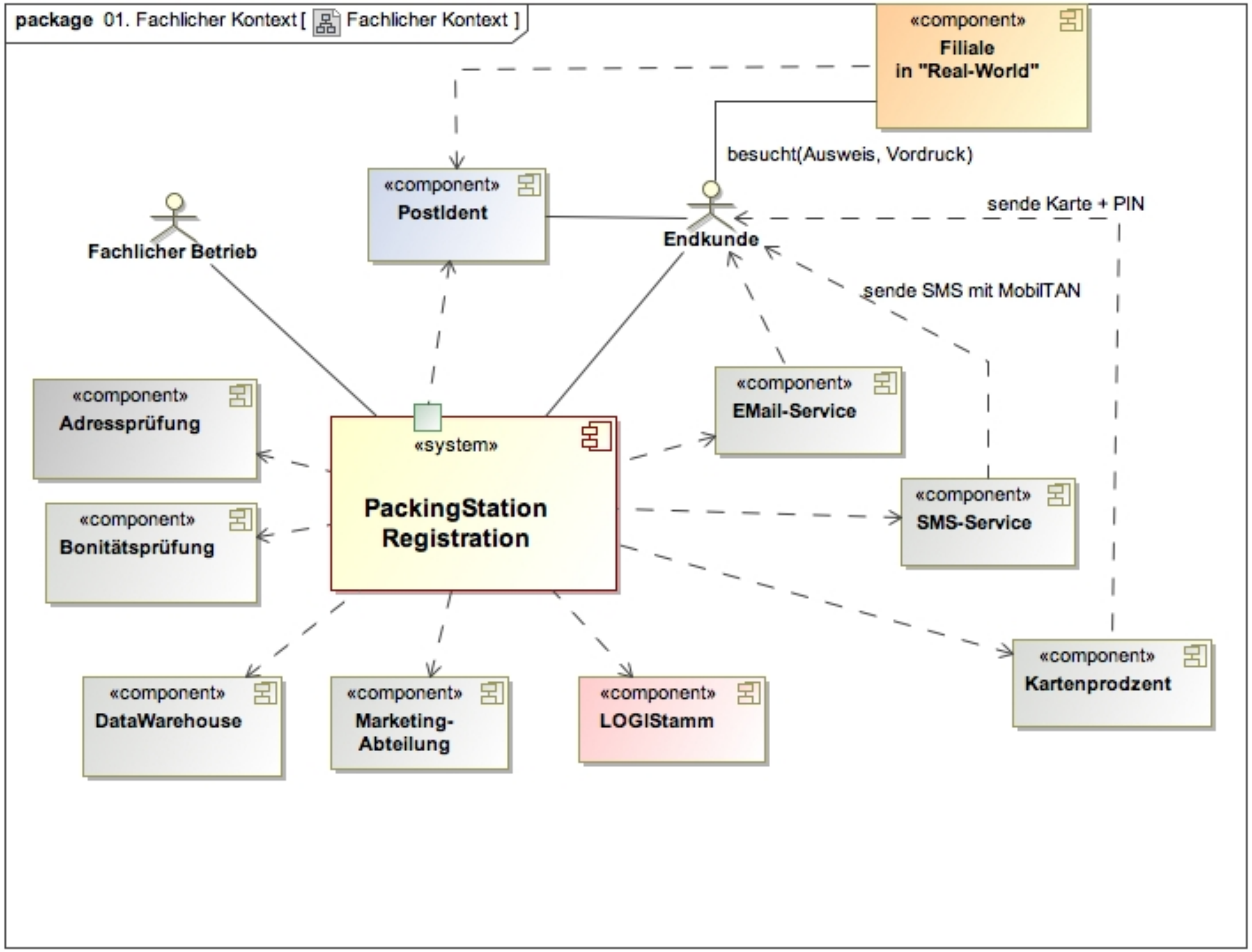
„Analysis“ Overview



„Analysis“ Overview

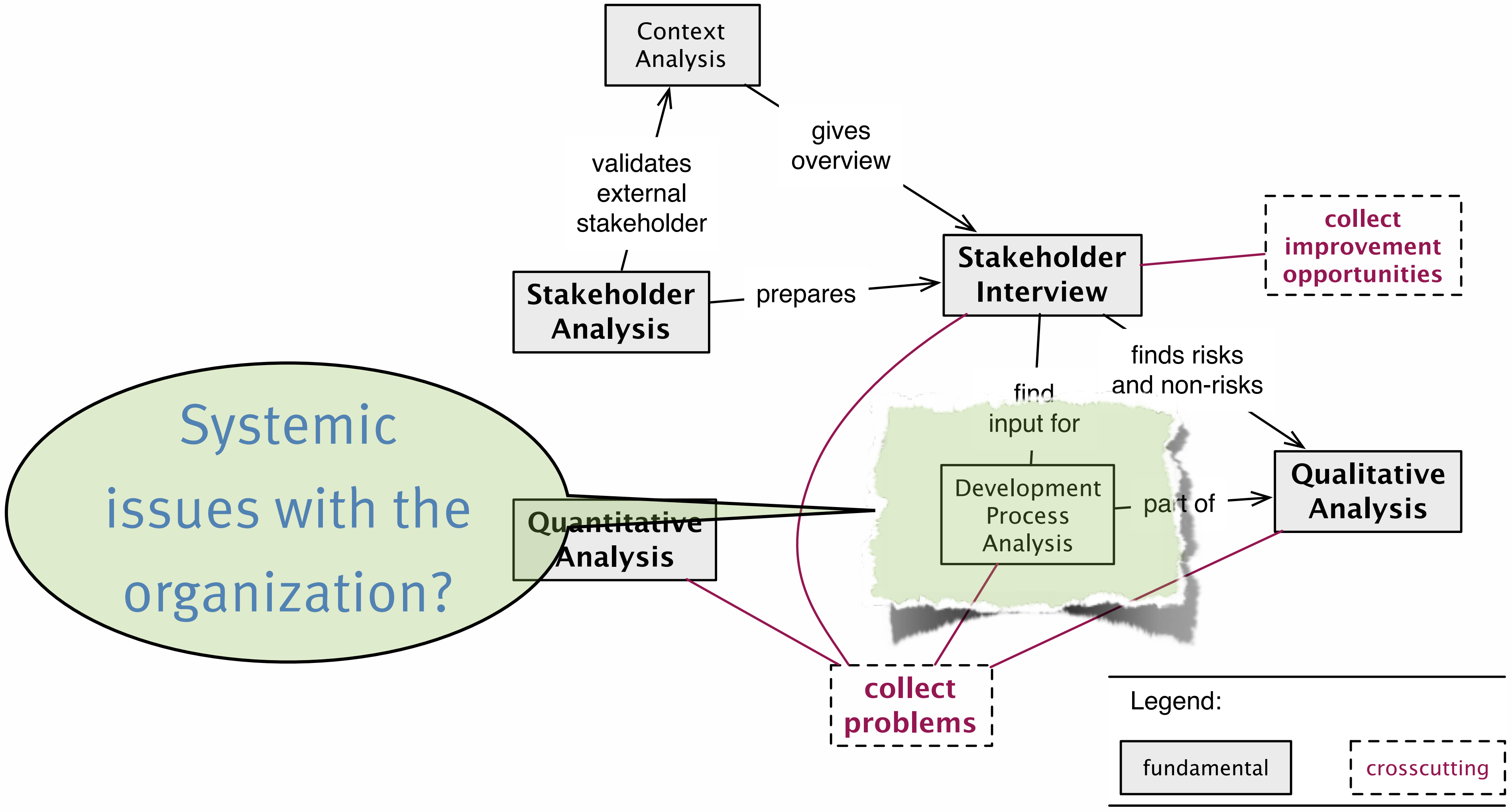


Context Example

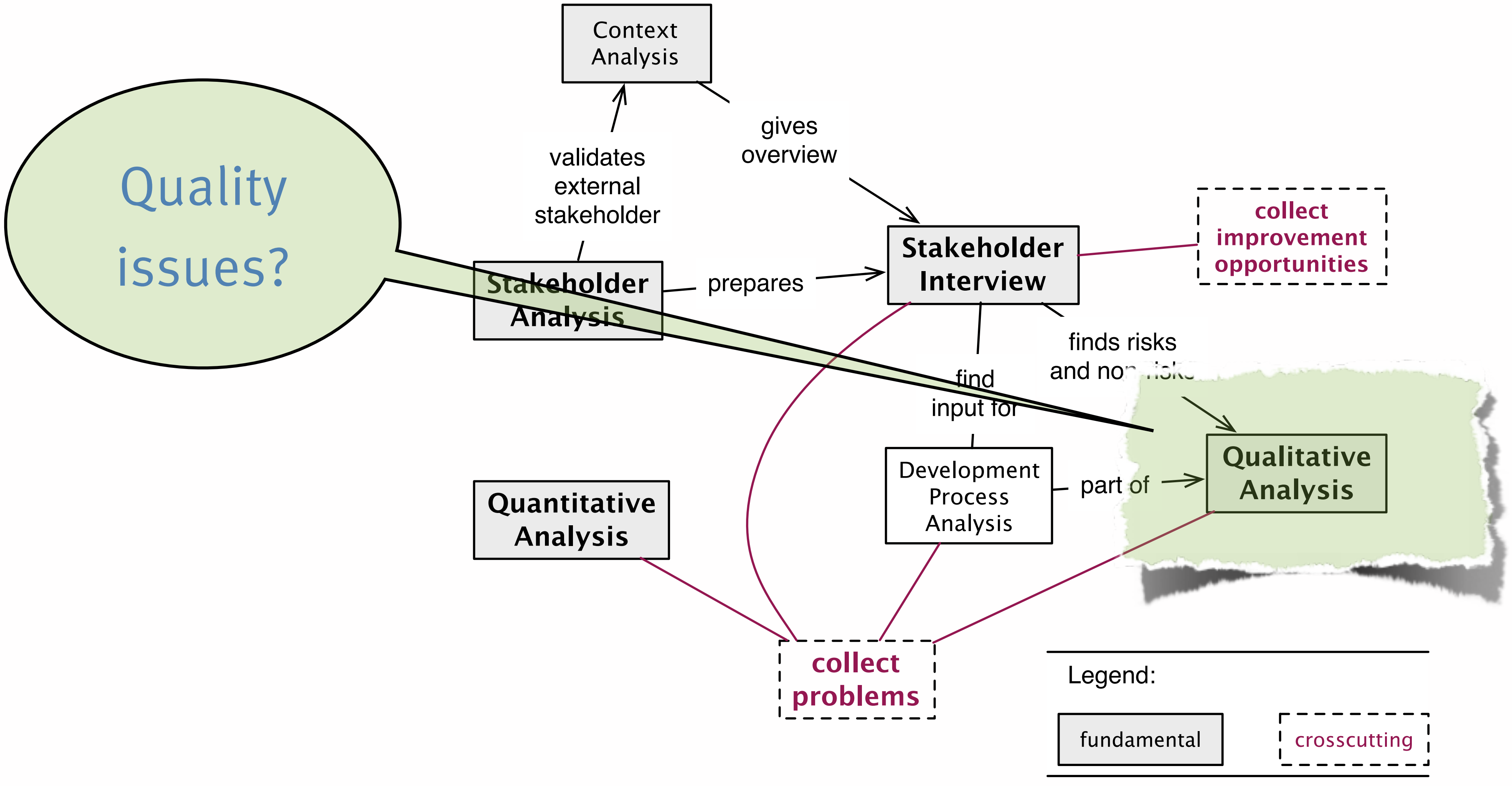


package 01. Fachlicher Kontext [Fachlicher Kontext]
«component» Filiale in "Real-World"
besucht(Ausweis, Vordruck)
«component» PostIdent
Fachlicher Betrieb
Endkunde
sende Karte + PIN
sende SMS mit MobilTAN
«component» EMail-Service
«component» Adressprüfung
«system» PackingStation Registration
«component» SMS-Service
«component» Bonitätsprüfung
«component» Kartenprodzent
«component» DataWarehouse
«component» Marketing-Abteilung
«component» LOGIStamm

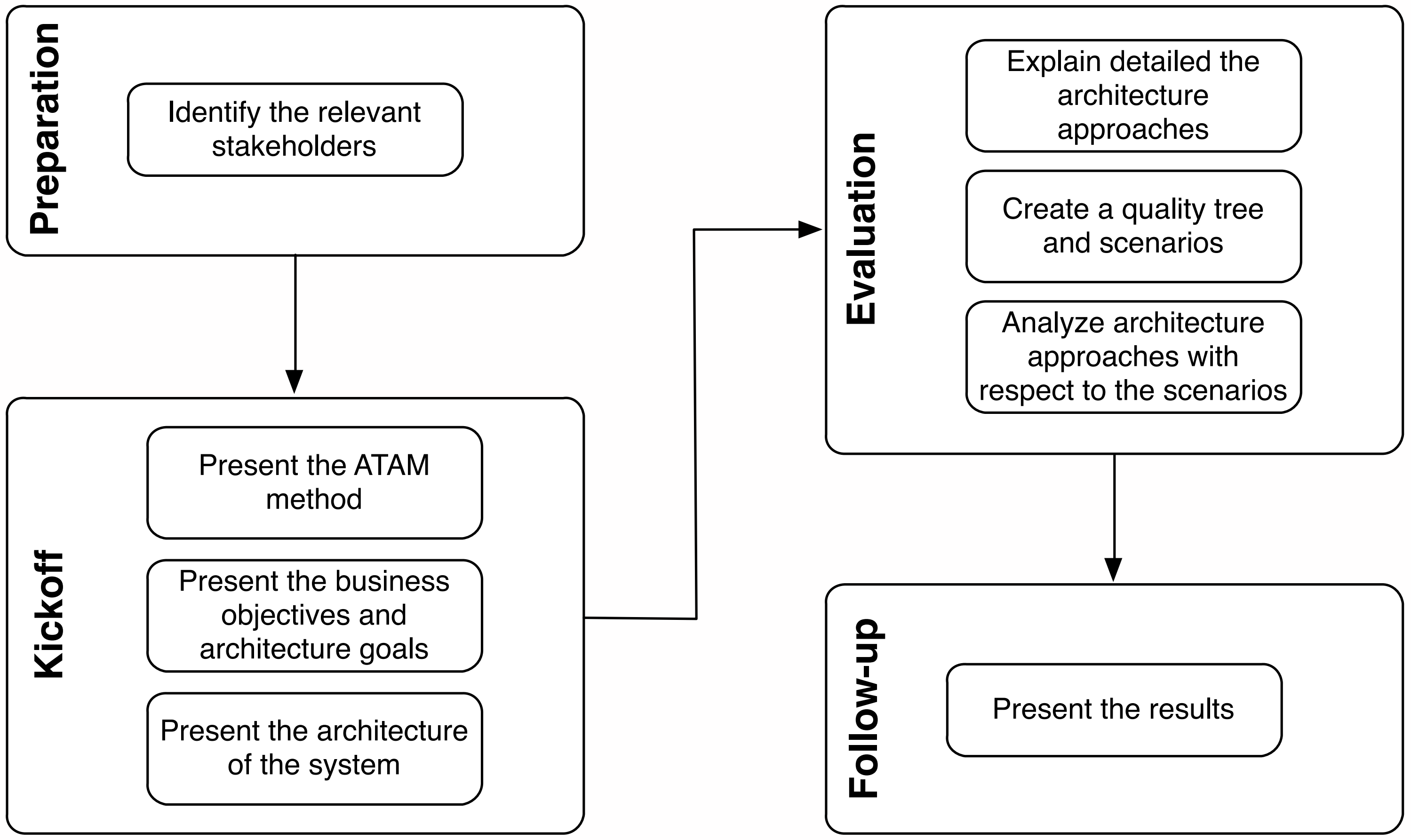
„Analysis“ Overview



„Analysis“ Overview

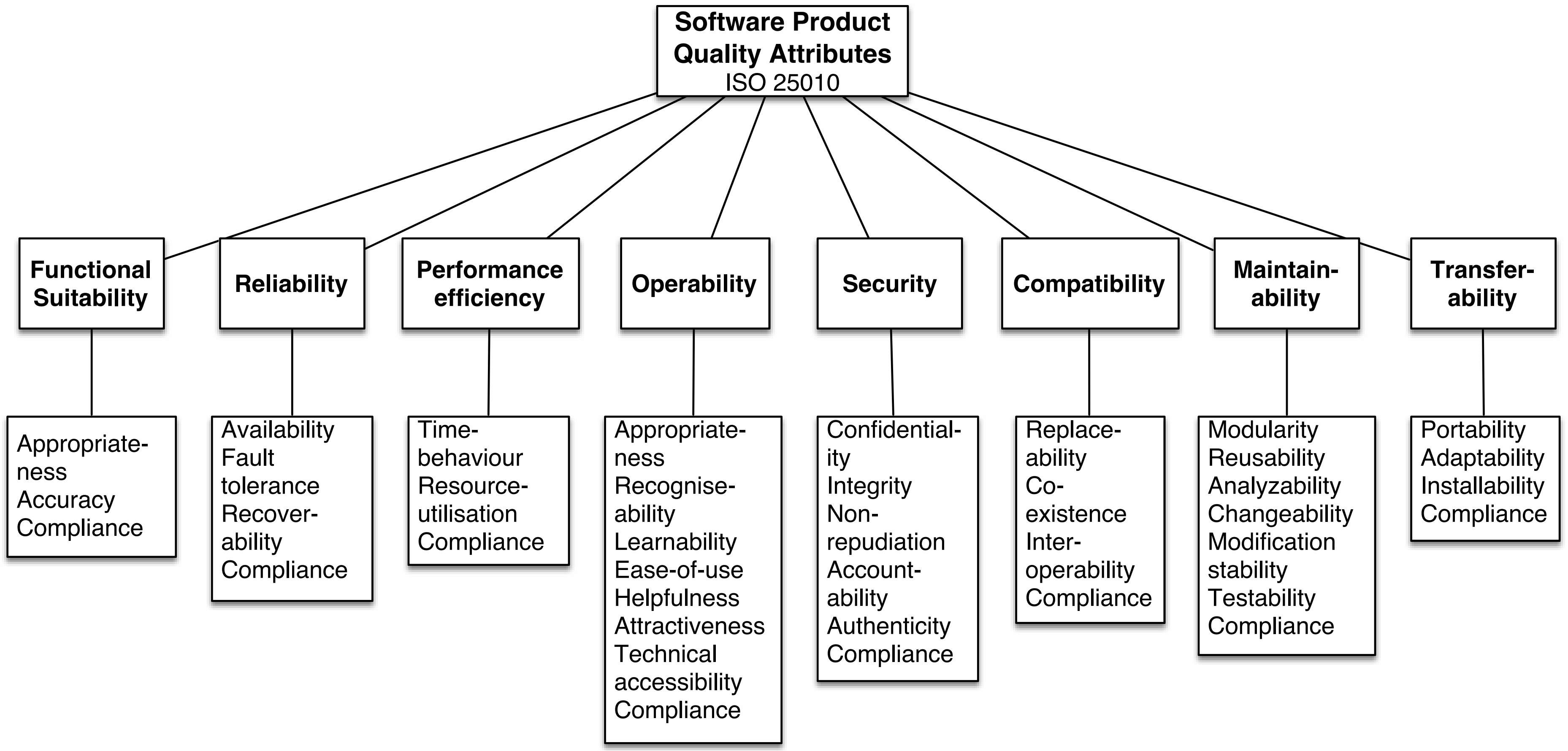


Qualitative Analysis



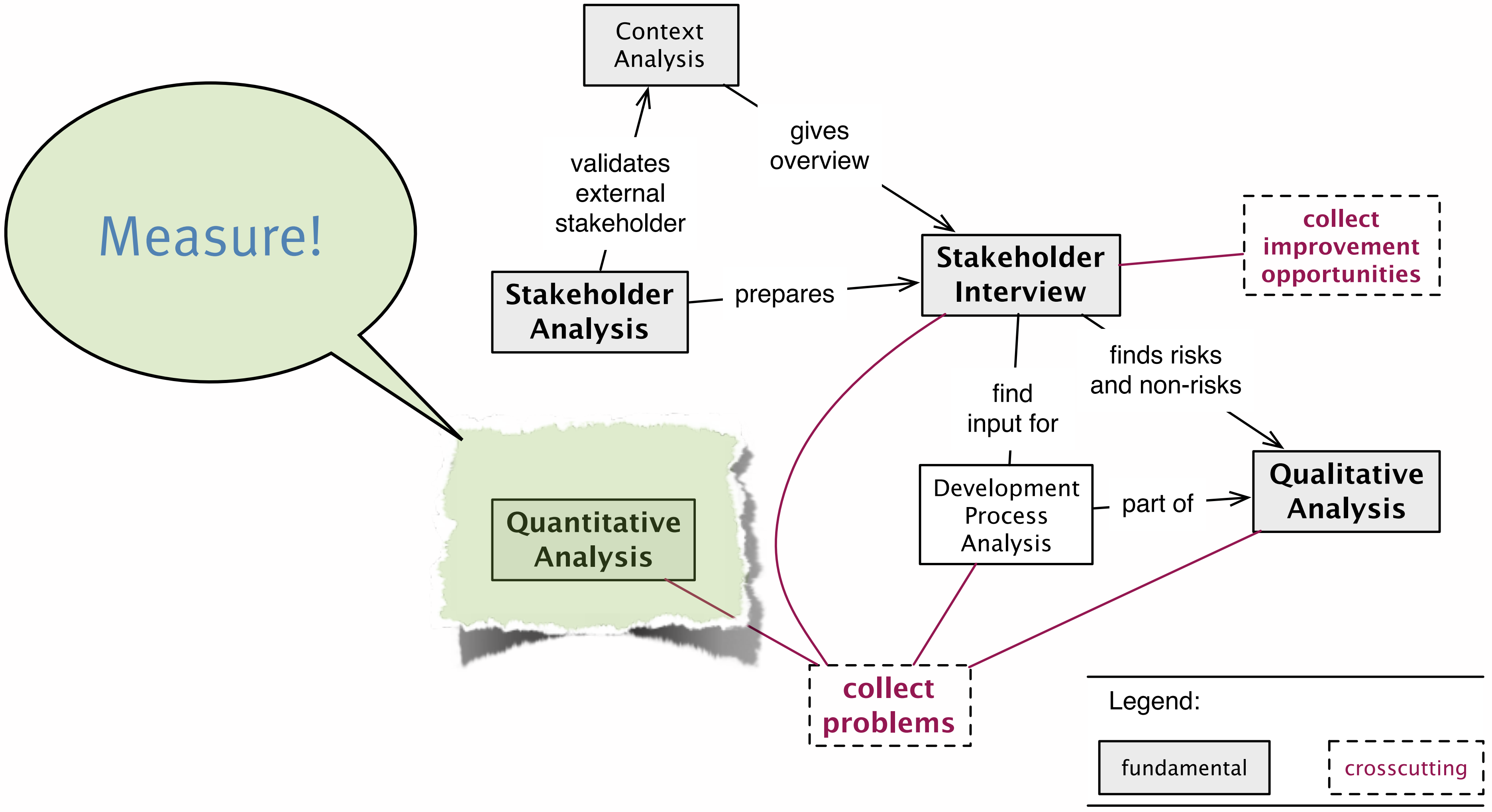
Preparation
Identify the relevant stakeholders
Kickoff
Present the ATAM method
Present the business objectives and architecture goals
Present the architecture of the system
Evaluation
Explain detailed the architecture approaches
Create a quality tree and scenarios
Analyze architecture approaches with respect to the scenarios
Follow-up
Present the results

Qualitative Analysis



Software Product Quality Attributes ISO 25010
Functional Suitability
Reliability
Performance efficiency
Operability
Security
Compatibility
Maintain-ability
Transfer-ability
Appropriate-ness Accuracy Compliance
Availability Fault tolerance Recover-ability Compliance
Time-behaviour Resource-utilisation Compliance
Appropriate-ness Recognise-ability Learnability Ease-of-use Helpfulness Attractiveness Technical accessibility Compliance
Confidential-ity Integrity Non-repudiation Account-ability Authenticity Compliance
Replace-ability Co-existence Inter-operability Compliance
Modularity Reusability Analyzability Changeability Modification stability Testability Compliance
Portability Adaptability Installability Compliance

„Analysis“ Overview



Stakeholder Analysis

Identify the right people!

top-management, business-management, project-management, product-management, process-management, client, subject-matter-expert, business-experts, business-development, enterprise-architect, IT-strategy, lead-architect, developer, tester, qa-representative, configuration-manager, release-manager, maintenance-team, external service provider, hardware-designer, rollout-manager, infrastructure-planner, infrastructure-provider, IT-administrator, DB-administrator, system-administrator, security- or safety-representative, end-user, hotline, service-technician, scrum-master, product-owner, business-controller, marketing, related-projects, public or government agency, authorities, standard-bodies, external service- or interface providers, industry- or business associations, trade-groups, competitors

Role / Name	Description	Intention	Contribution	Contact

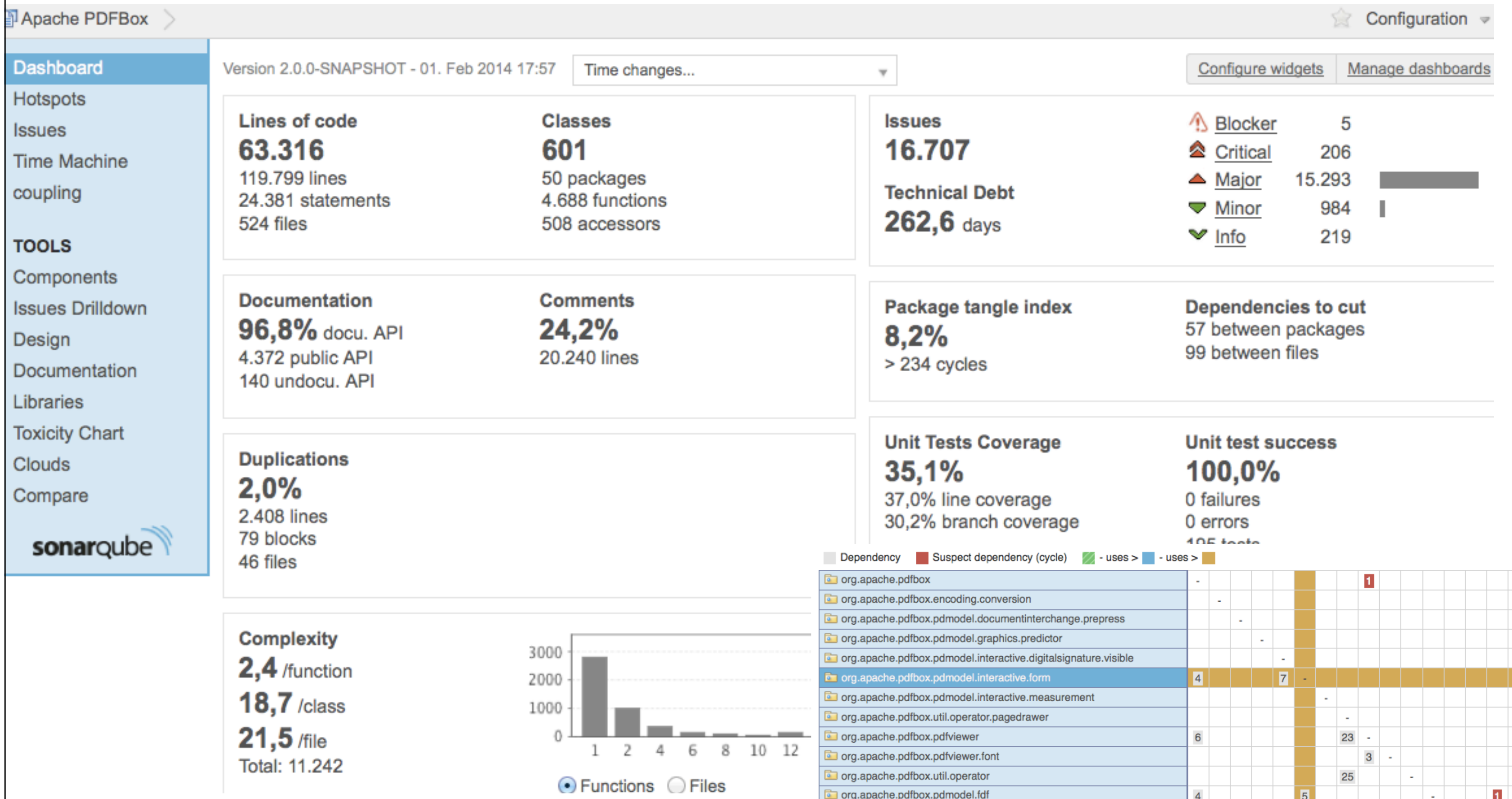
Stakeholder Analysis (II)

who MIGHT have problems
or know things...

- use (pre-interview) questionnaire

- conduct personal interviews:
  e.g. what are your top-3 issues with...
  1. the system
  2. the development / maintenance process
  3. operation / infrastructure of the system
  4. ...

Static Code Analysis (here: SonarQube dashboard / Apache PDFbox)



Apache PDFBox

Configuration

- Dashboard
- Hotspots
- Issues
- Time Machine
- coupling

TOOLS

- Components
- Issues Drilldown
- Design
- Documentation
- Libraries
- Toxicity Chart
- Clouds
- Compare

sonarqube

Version 2.0.0-SNAPSHOT - 01. Feb 2014 17:57 | Time changes... | Configure widgets | Manage dashboards

Lines of code
63.316
119.799 lines
24.381 statements
524 files

Classes
601
50 packages
4.688 functions
508 accessors

Issues
16.707

Technical Debt
262,6 days

Blocker	5
Critical	206
Major	15.293
Minor	984
Info	219

Documentation
96,8% docu. API
4.372 public API
140 undocu. API

Comments
24,2%
20.240 lines

Package tangle index
8,2%
> 234 cycles

Dependencies to cut
57 between packages
99 between files

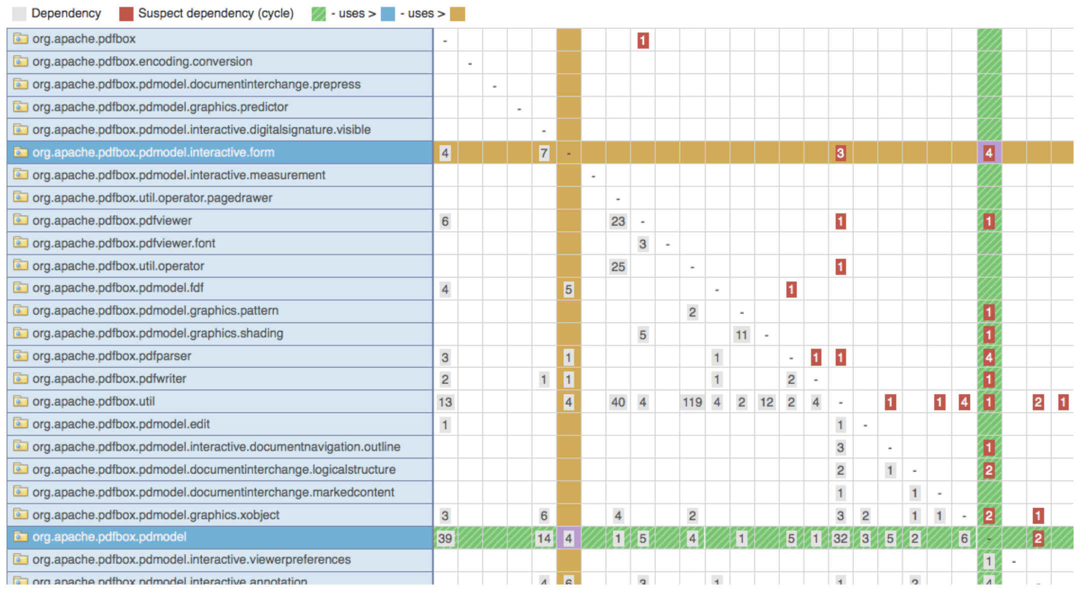
Duplications
2,0%
2.408 lines
79 blocks
46 files

Unit Tests Coverage
35,1%
37,0% line coverage
30,2% branch coverage

Unit test success
100,0%
0 failures
0 errors

Complexity
2,4 /function
18,7 /class
21,5 /file
Total: 11.242

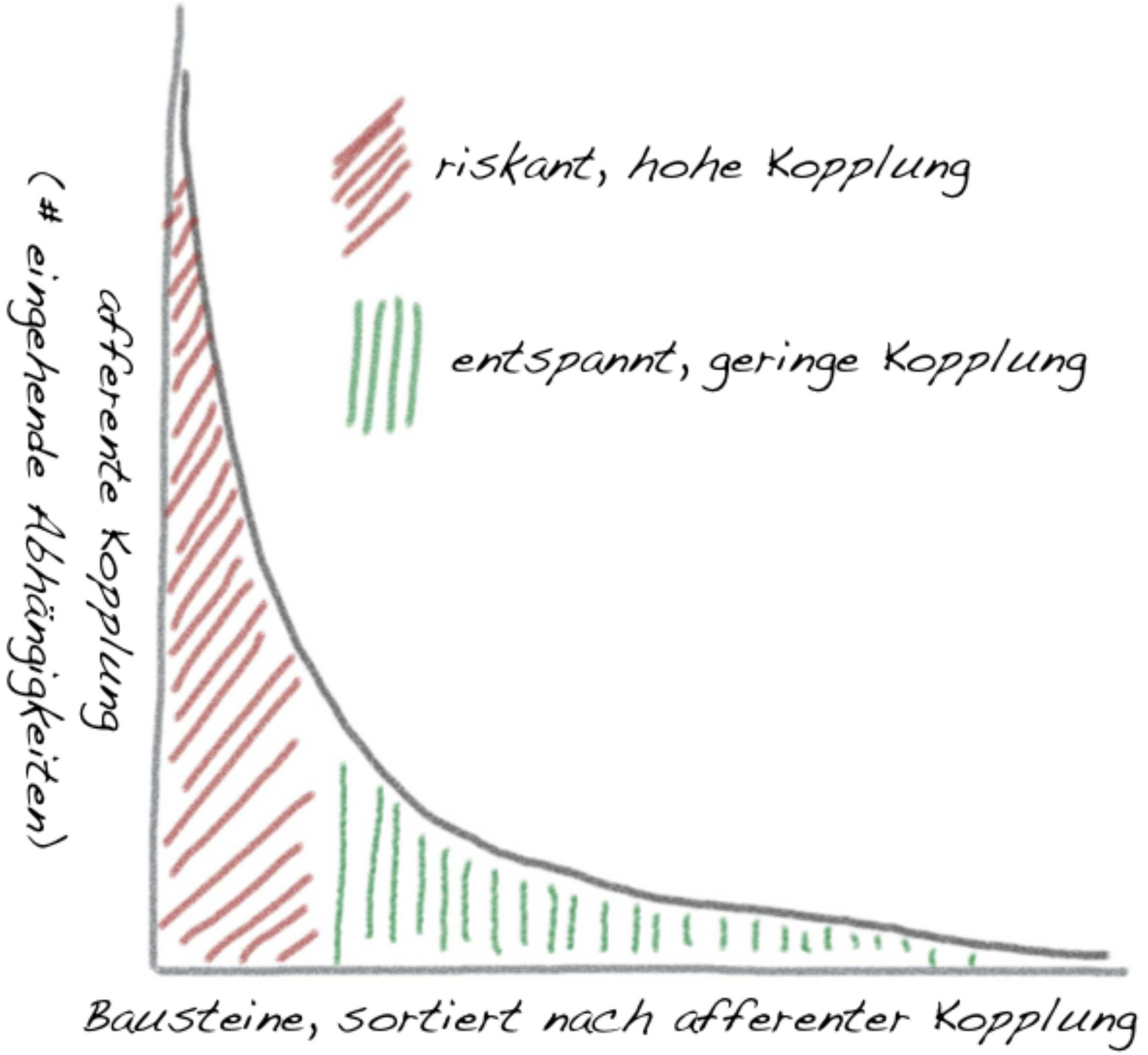
[● Functions] ○ Files



⬜ Dependency 🟥 Suspect dependency (cycle) 🟩 - uses > 🟦 - uses > 🟧

org.apache.pdfbox	-								🟥 1																	
org.apache.pdfbox.encoding.conversion		-																								
org.apache.pdfbox.pdmodel.documentinterchange.prepress			-																							
org.apache.pdfbox.pdmodel.graphics.predictor				-																						
org.apache.pdfbox.pdmodel.interactive.digitalsignature.visible					-																					
org.apache.pdfbox.pdmodel.interactive.form	4				7	-											🟥 3						🟥 4			
org.apache.pdfbox.pdmodel.interactive.measurement							-																			
org.apache.pdfbox.util.operator.pagedrawer								-																		
org.apache.pdfbox.pdfviewer	6							23	-								🟥 1						🟥 1			
org.apache.pdfbox.pdfviewer.font									3	-																
org.apache.pdfbox.util.operator								25			-						🟥 1									
org.apache.pdfbox.pdmodel.fdf	4					5						-			🟥 1											
org.apache.pdfbox.pdmodel.graphics.pattern											2		-										🟥 1			
org.apache.pdfbox.pdmodel.graphics.shading									5				11	-									🟥 1			
org.apache.pdfbox.pdfparser	3					1						1			-	🟥 1	🟥 1						🟥 4			
org.apache.pdfbox.pdfwriter	2				1	1						1			2	-							🟥 1			
org.apache.pdfbox.util	13					4		40	4		119	4	2	12	2	4	-		🟥 1		🟥 1	🟥 4	🟥 1		🟥 2	🟥 1
org.apache.pdfbox.pdmodel.edit	1																1	-								
org.apache.pdfbox.pdmodel.interactive.documentnavigation.outline																	3		-				🟥 1			
org.apache.pdfbox.pdmodel.documentinterchange.logicalstructure																	2		1	-			🟥 2			
org.apache.pdfbox.pdmodel.documentinterchange.markedcontent																	1			1	-					
org.apache.pdfbox.pdmodel.graphics.xobject	3				6			4			2						3	2		1	1	-	🟥 2		🟥 1	
org.apache.pdfbox.pdmodel	39				14	4		1	5		4		1		5	1	32	3	5	2		6	-		🟥 2	
org.apache.pdfbox.pdmodel.interactive.viewerpreferences																							1	-		
org.apache.pdfbox.pdmodel.interactive.annotation					4	6			3			1					1			2			4		-	

Static Code Analysis (here: afferent coupling)



Perishable Food Packaging

- Embedded software + information systems
- Regulated domain -> safety critical

- Goal: Decrease SW development cost

Food: Analysis

- Stakeholder analysis and -interviews
- Development Process Analysis
- Qualitative Analysis + View-Based-Understanding
- Quantitative Analysis, Static Code Analysis

- Central problem areas:
  - Lack of overview („knowledge islands“)
  - Low code quality
  - ad-hoc development: No systematic processes

Food: Root-Cause Analysis

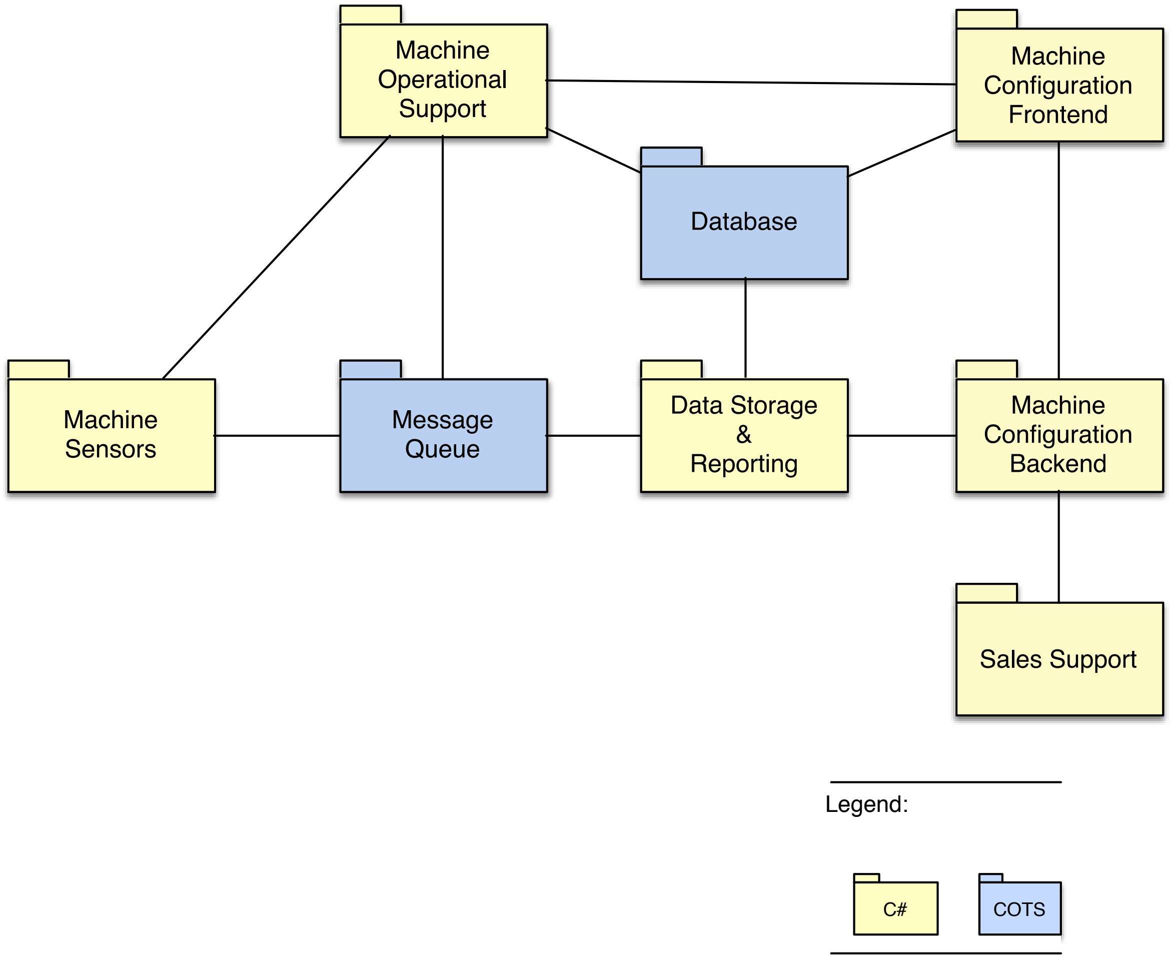
- Company focus primarily on hardware
- Software development scattered in various departments
- No (planned) software architecture

Food: Analysis (excerpt)

issue (problem)	description	problem-cost
time-to-market	> 6 month (!) from business or government requirement to production	sales loss might be > 1M$
production log data loss	architecture does not ensure complete production logs - data records might get lost! Large volumes of perishable food could be at risk	> 10-100k $ per incident
scattered knowledge + low code quality	no synergy effects, no conceptual integrity, no re-use between departments, ...	>5-50k $ per maintenance update
self-developed OR-mapper	expensive maintenance, high know-how requirements, high deviation in performance	5-10k $ per maintenance update

Food: System Overview

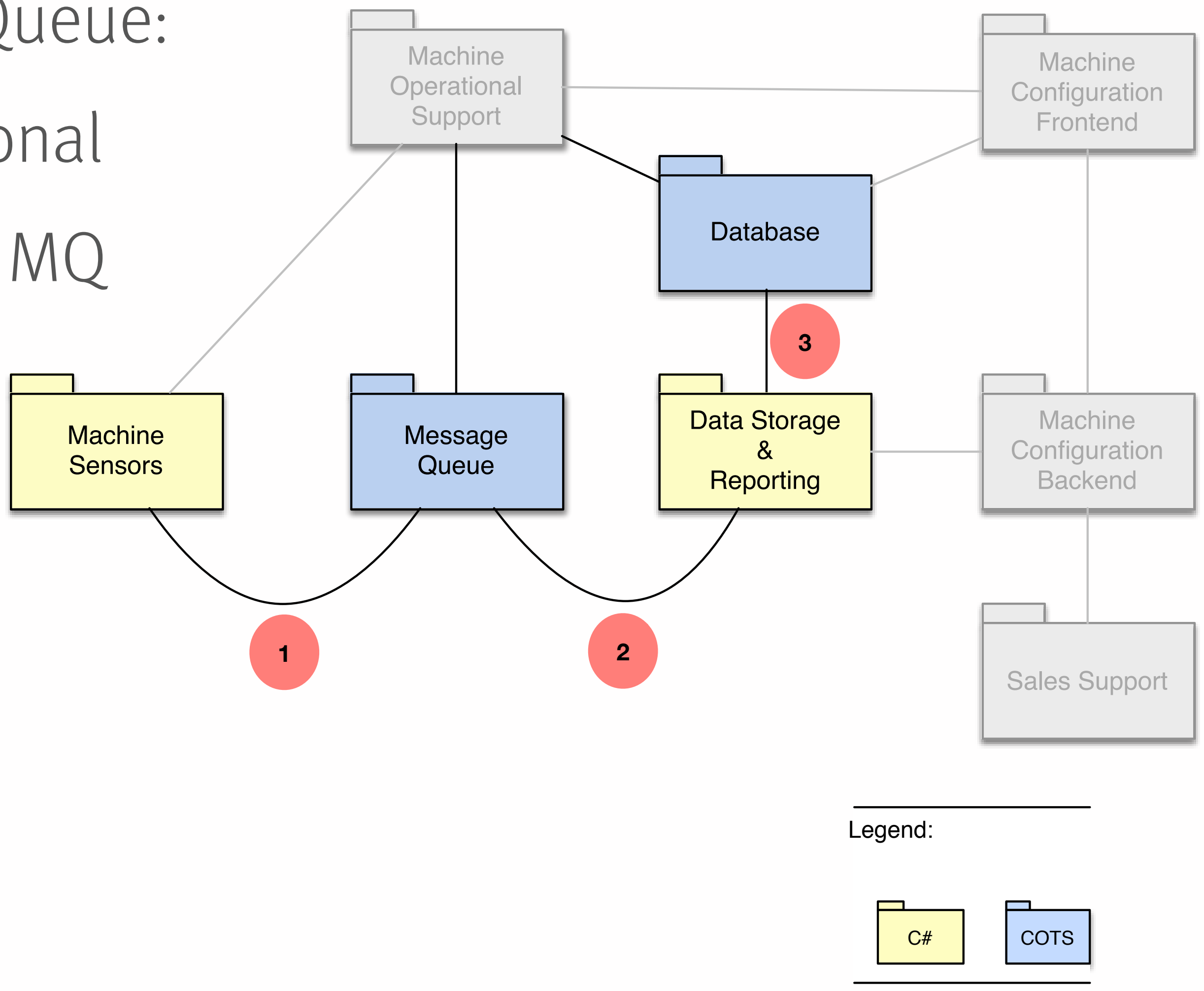
- C# / .NET as development & production platform


Machine Operational Support
Machine Configuration Frontend
Database
Machine Sensors
Message Queue
Data Storage & Reporting
Machine Configuration Backend
Sales Support
Legend:
C#
COTS

Food: Safety Risk

Wrong usage of Message Queue:

- 1.-3. has to be transactional
- Reporting „commits“ to MQ after 2! (too early!)
- Problem in reporting leads to lost data!



EU Telecom Provider

- Business Intelligence Portfolio to support Marketing & Sales

Telco: Analysis

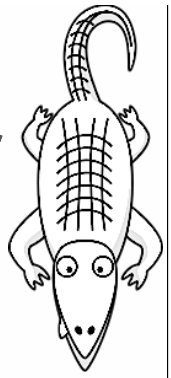
- View-Based-Understanding
- Data Analysis
- (few) stakeholder interviews

- Central problem areas:
  - BI Reporting highly fragmented & diverse
  - Report implementation details driven by business experts (provided data models + SQL query details as „requirements“)
  - Implementation partially based upon proprietary meta-model

Telco: Analysis (excerpt)

problem / risk	description	problem-cost
high development cost	business benchmarks showed development to be overly expensive (and slow)	per report-type 50-200%
non-transparent software and data architecture	of >50 developers and BI experts, only very few understood whole DWH	
vendor-lock-in	proprietary tools implemented to process (proprietary) meta-model, high yearly license cost,	50 k€ license fee / yr, O(1000) dev-hrs wasted
developer exodus	core developers upset as company announced large outsourcing deal, (nearly) annihilating internal development	6-18 month without new business features

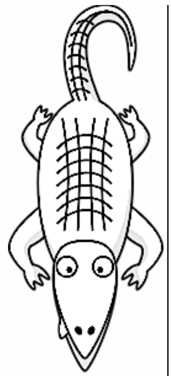
Croc: Sales & ERP Provider



Company name changed due to anonymity requirements!

- Niche provider for sales & ERP „standard“ solution
- Origin in „perishable“ market - but growing
  - 80% of clients: low-margin-high-volume
  - 20% of clients: low-volume-very-high-margin
- Original idea: Universal-Core + Configuration
- Starting point:
  low (dev + runtime) performance

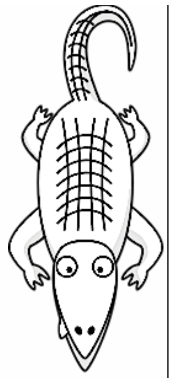
Croc: Analysis



- Brief stakeholder analysis and -interviews
- Static Code Analysis
- Runtime Analysis
- Data Analysis (including data model)

- Central problem areas:
  - Excellent code quality („clean code“) - but very few unit tests
  - Extremely high configurability of everything
  - >150 developers with extremely different options

Croc: Analysis (2)



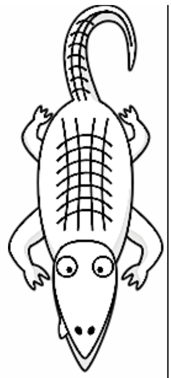
„Configuration is the sequel to programming, with unsuitable means“

› Configuring UI structure, UI behavior, workflows, business and validation rules, reports and interfaces

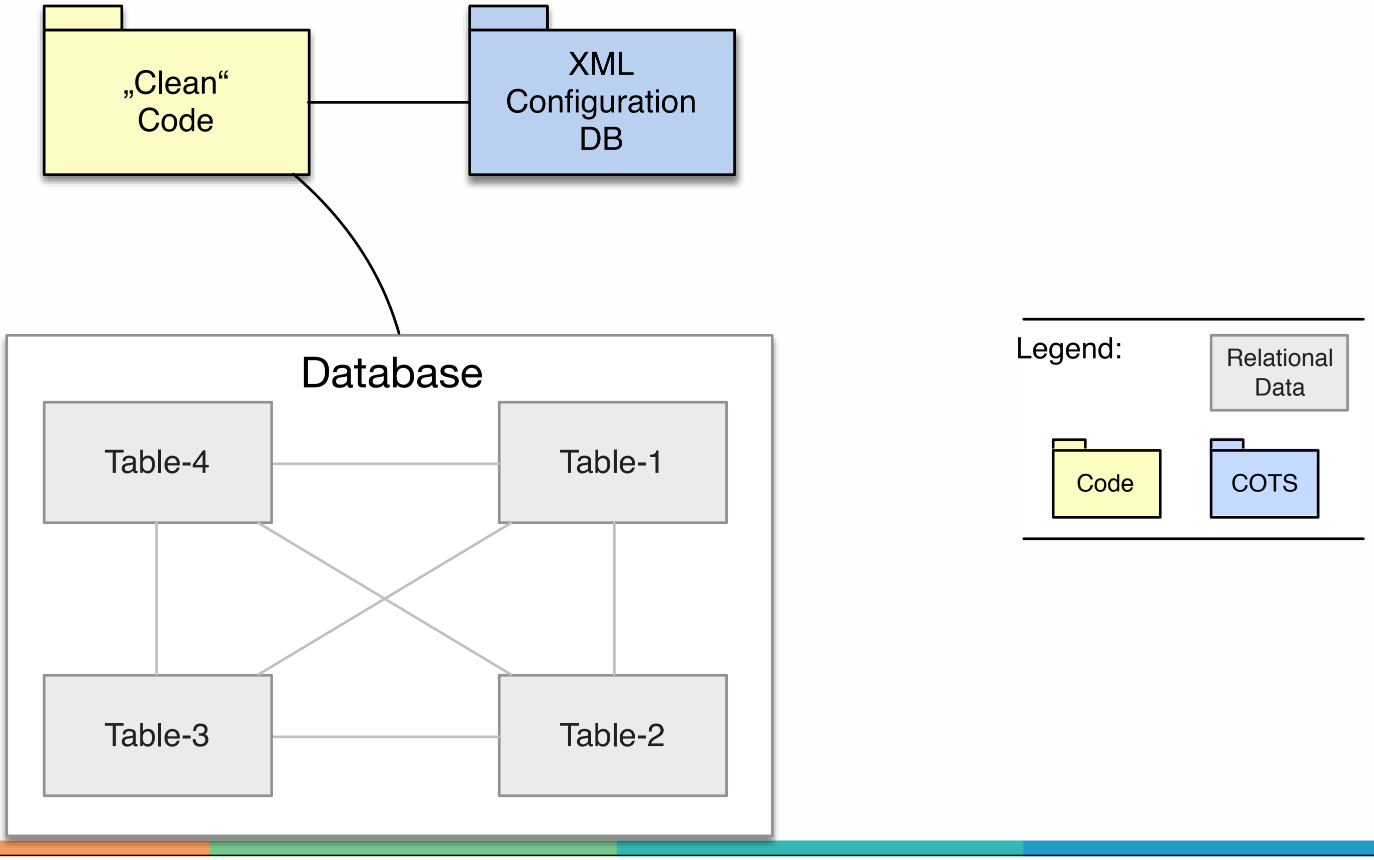
› Horrible persistent data structures for both runtime and configuration data

› Some configuration stored in various XML formats

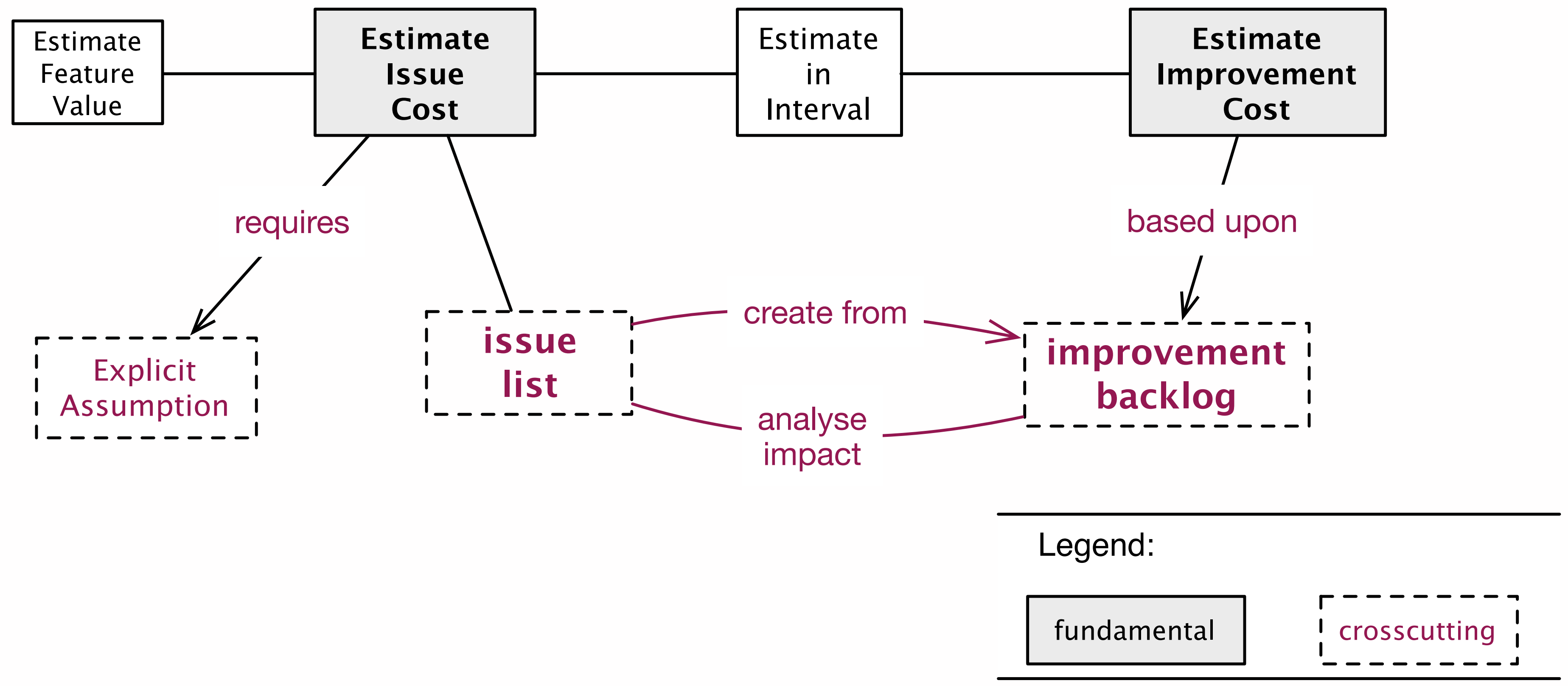
Croc: Analysis (3)



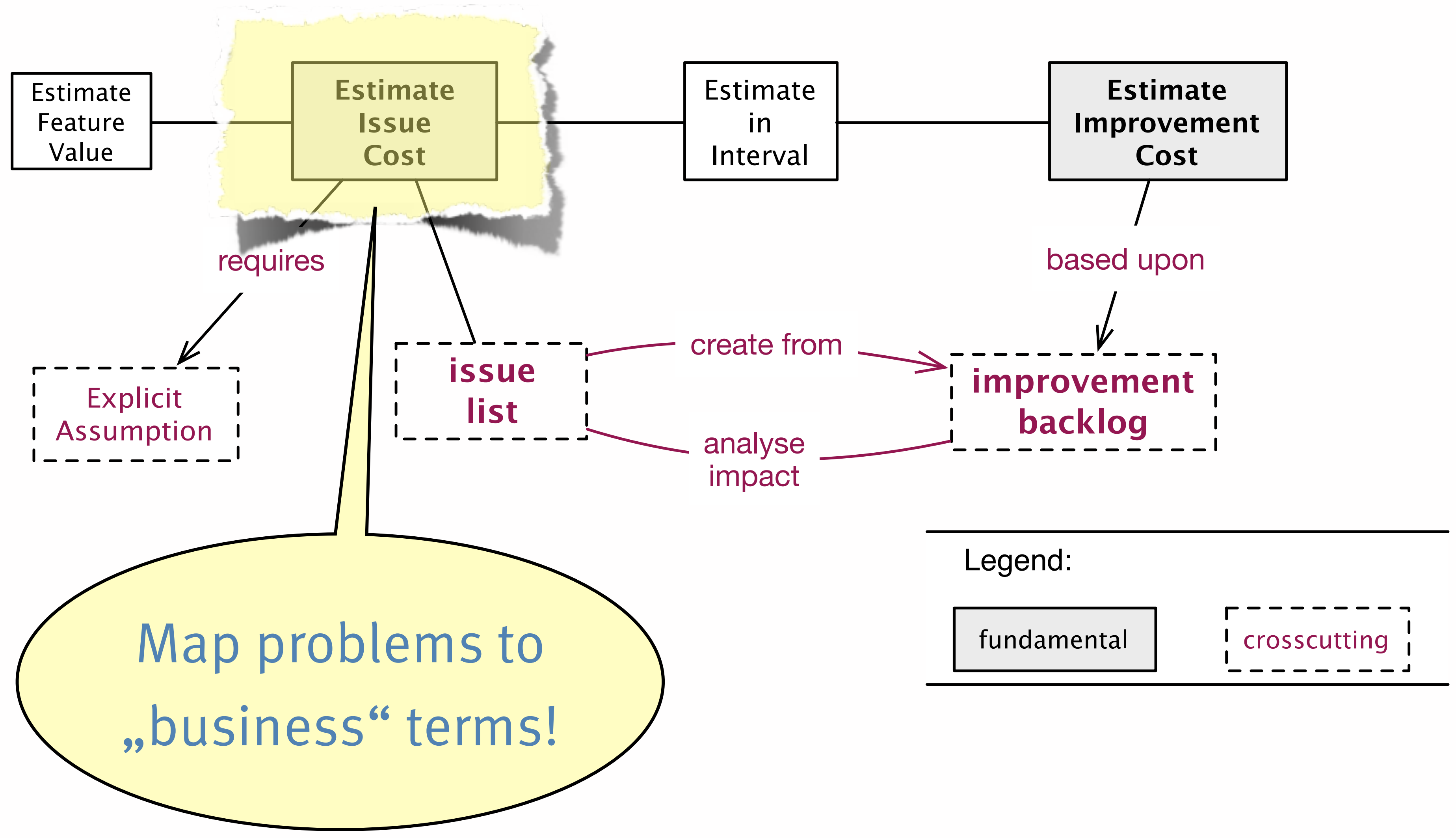
- Few key tables with 500-700 columns (!!) each.
- Stores complete application state - including cursor position.



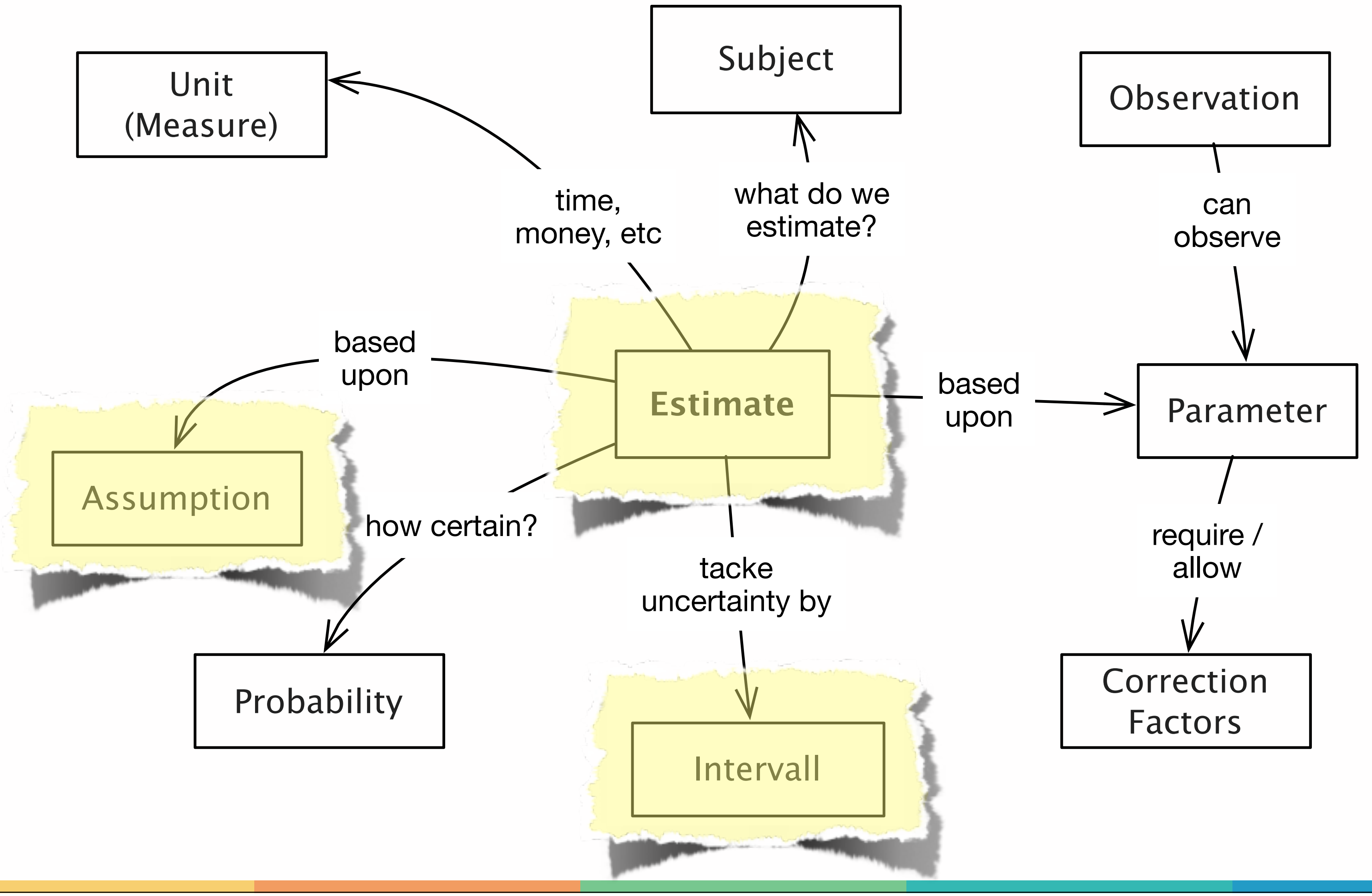
„Evaluate“ Overview



„Evaluate“ Overview



„Evaluate“ Concepts



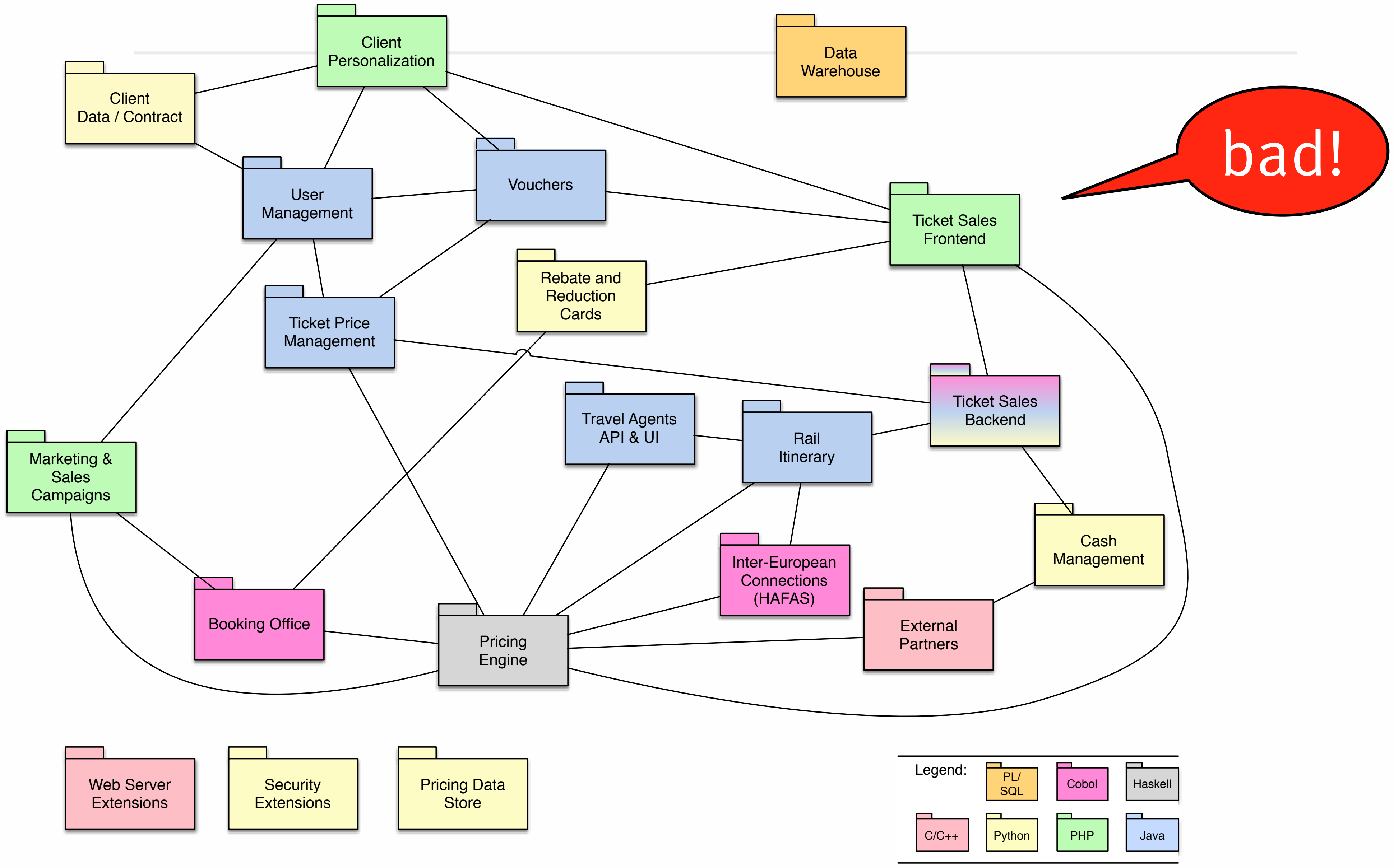
Rail Transport Provider

- Heterogeneous IT landscape
- Problem areas:
  - 6-12 month from initial business requirement to production („time-to-market“)
  - Stability, reliability
  - Performance

Rail - aim42 Analysis

- Stakeholder Analysis + -Interviews
  - yielded several problems + problem-areas
  - Issue Tracker Analysis + Software Archeology
- Qualitative (ATAM-like) Analysis
- Static Code Analysis
- Development Process Analysis

Rail (1): Overview



Rail (2): Challenges

- Embrace new sales channels (mobile)
  - requires (much) higher availability
- Marketing demands rapid price adjustments

Rail (4): Analysis (excerpt)

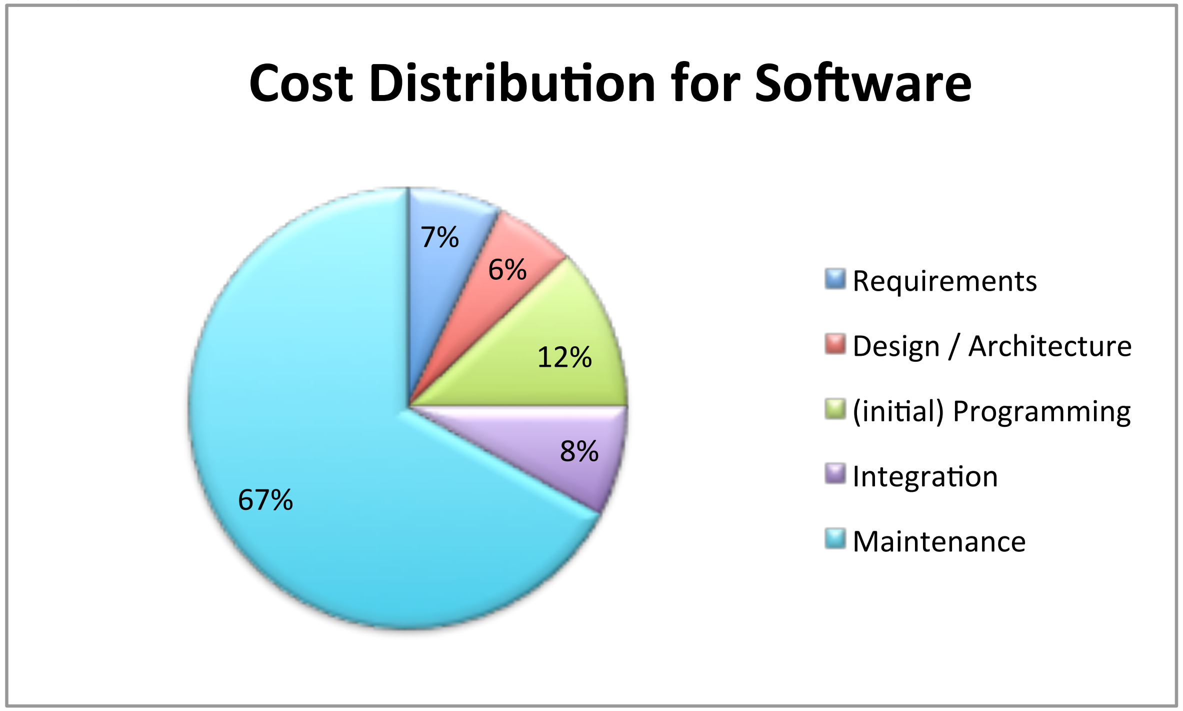
issue (problem)	description	problem-cost
time-to-market	6-12 month (!) from business requirement to production	
configuration of certain ticket types crashes backend	when either end-users or sales-clerks configure specific ticket-types (groups > 5 persons, more than one rebate reason, border crossing or >2 train changes), several backend processes crash	
know-how drain in development	many dissatisfied developers and business experts leave (development) organization, migration from internal to external development, fix-price projects	

Rail (5): Evaluation (excerpt)

What‘s the (additional) cost of „heterogenity“?

1. Explicit assumptions

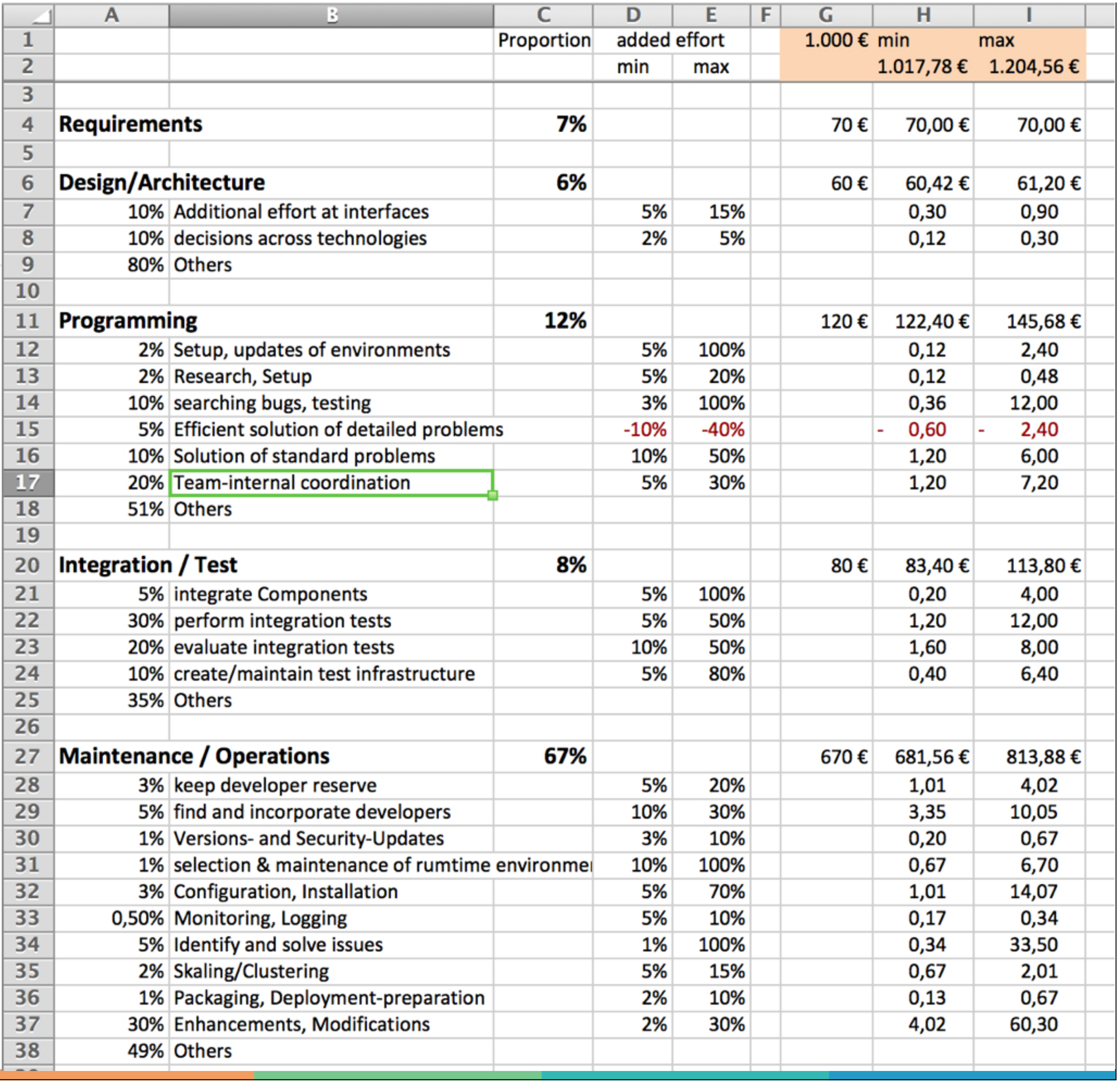
- Heterogenity „costs“ in all phases
- Phase effort is known



http://courses.cs.vt.edu/~csonline/SE/Lessons/LifeCycle/

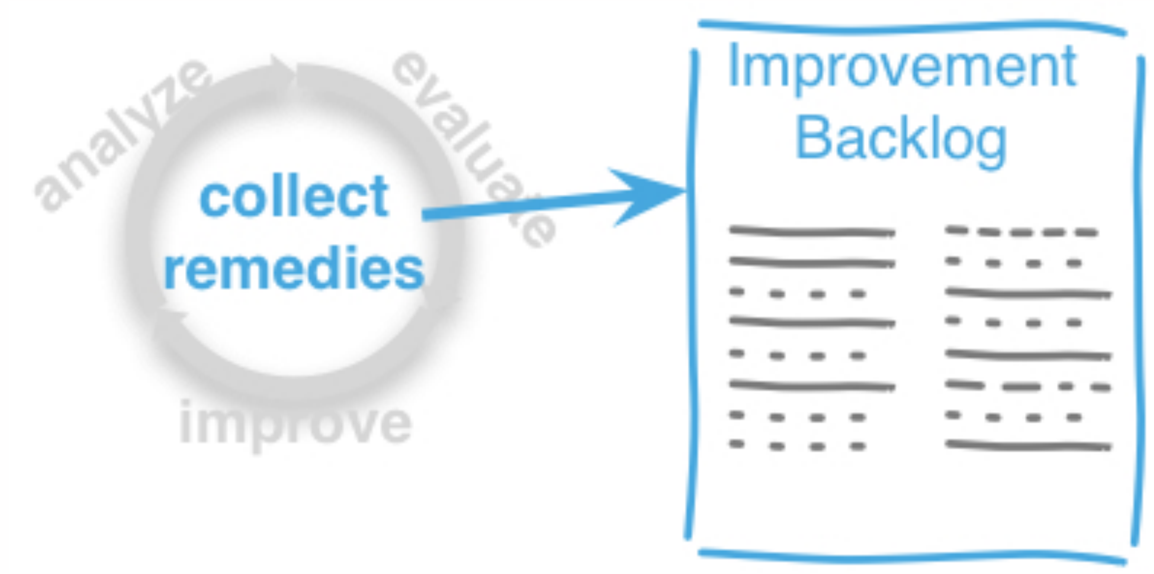
Rail (6)...

Collected tasks in which additional effort might occur..



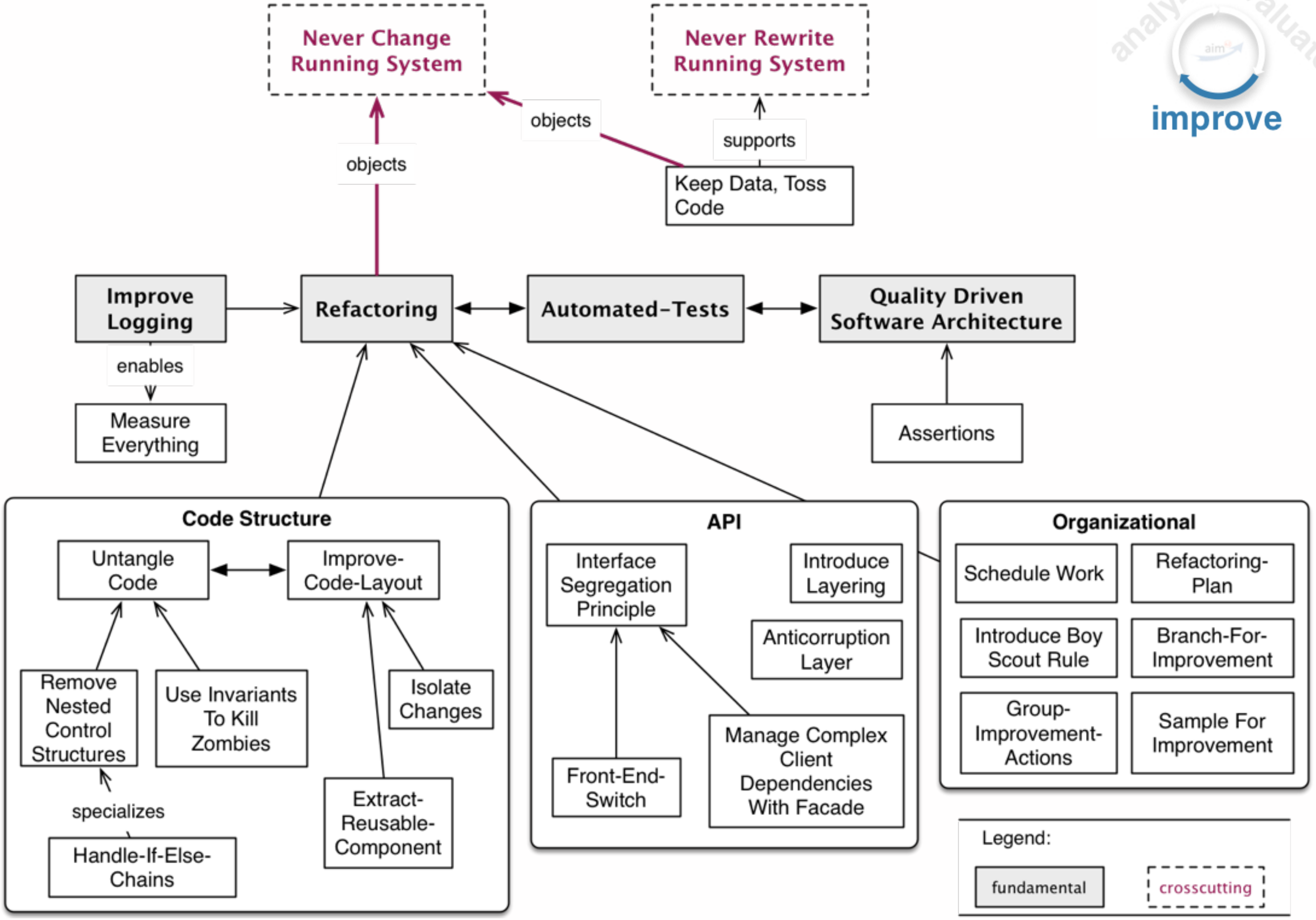
	A	B	C	D	E	F	G	H	I
1			Proportion	added effort			1.000 €	min	max
2				min	max			1.017,78 €	1.204,56 €
3									
4	**Requirements**		**7%**				70 €	70,00 €	70,00 €
5									
6	**Design/Architecture**		**6%**				60 €	60,42 €	61,20 €
7	10%	Additional effort at interfaces		5%	15%			0,30	0,90
8	10%	decisions across technologies		2%	5%			0,12	0,30
9	80%	Others							
10									
11	**Programming**		**12%**				120 €	122,40 €	145,68 €
12	2%	Setup, updates of environments		5%	100%			0,12	2,40
13	2%	Research, Setup		5%	20%			0,12	0,48
14	10%	searching bugs, testing		3%	100%			0,36	12,00
15	5%	Efficient solution of detailed problems		-10%	-40%			- 0,60	- 2,40
16	10%	Solution of standard problems		10%	50%			1,20	6,00
17	20%	Team-internal coordination		5%	30%			1,20	7,20
18	51%	Others							
19									
20	**Integration / Test**		**8%**				80 €	83,40 €	113,80 €
21	5%	integrate Components		5%	100%			0,20	4,00
22	30%	perform integration tests		5%	50%			1,20	12,00
23	20%	evaluate integration tests		10%	50%			1,60	8,00
24	10%	create/maintain test infrastructure		5%	80%			0,40	6,40
25	35%	Others							
26									
27	**Maintenance / Operations**		**67%**				670 €	681,56 €	813,88 €
28	3%	keep developer reserve		5%	20%			1,01	4,02
29	5%	find and incorporate developers		10%	30%			3,35	10,05
30	1%	Versions- and Security-Updates		3%	10%			0,20	0,67
31	1%	selection & maintenance of rumtime environme		10%	100%			0,67	6,70
32	3%	Configuration, Installation		5%	70%			1,01	14,07
33	0,50%	Monitoring, Logging		5%	10%			0,17	0,34
34	5%	Identify and solve issues		1%	100%			0,34	33,50
35	2%	Skaling/Clustering		5%	15%			0,67	2,01
36	1%	Packaging, Deployment-preparation		2%	10%			0,13	0,67
37	30%	Enhancements, Modifications		2%	30%			4,02	60,30
38	49%	Others							

„Improve“ Overview

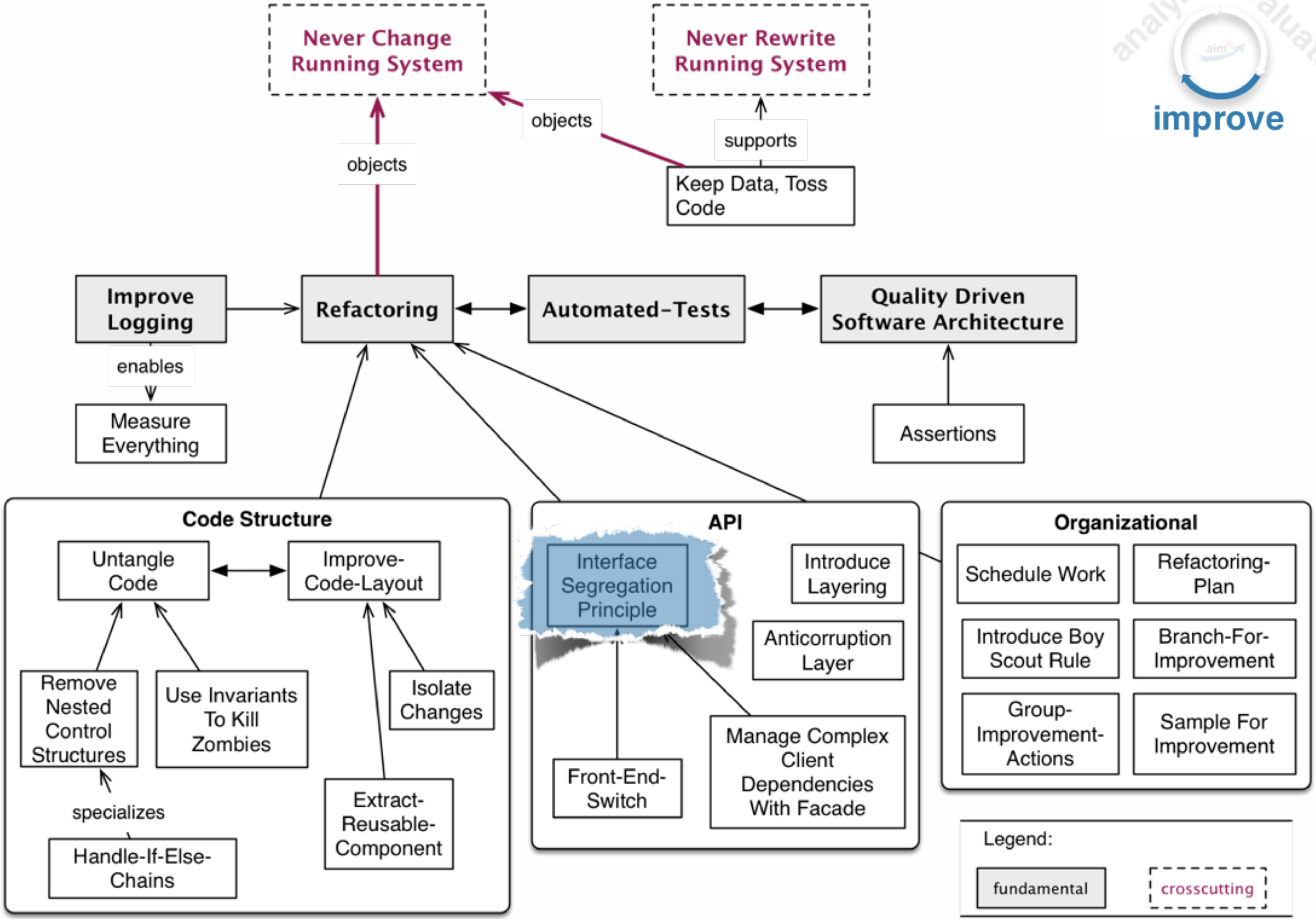


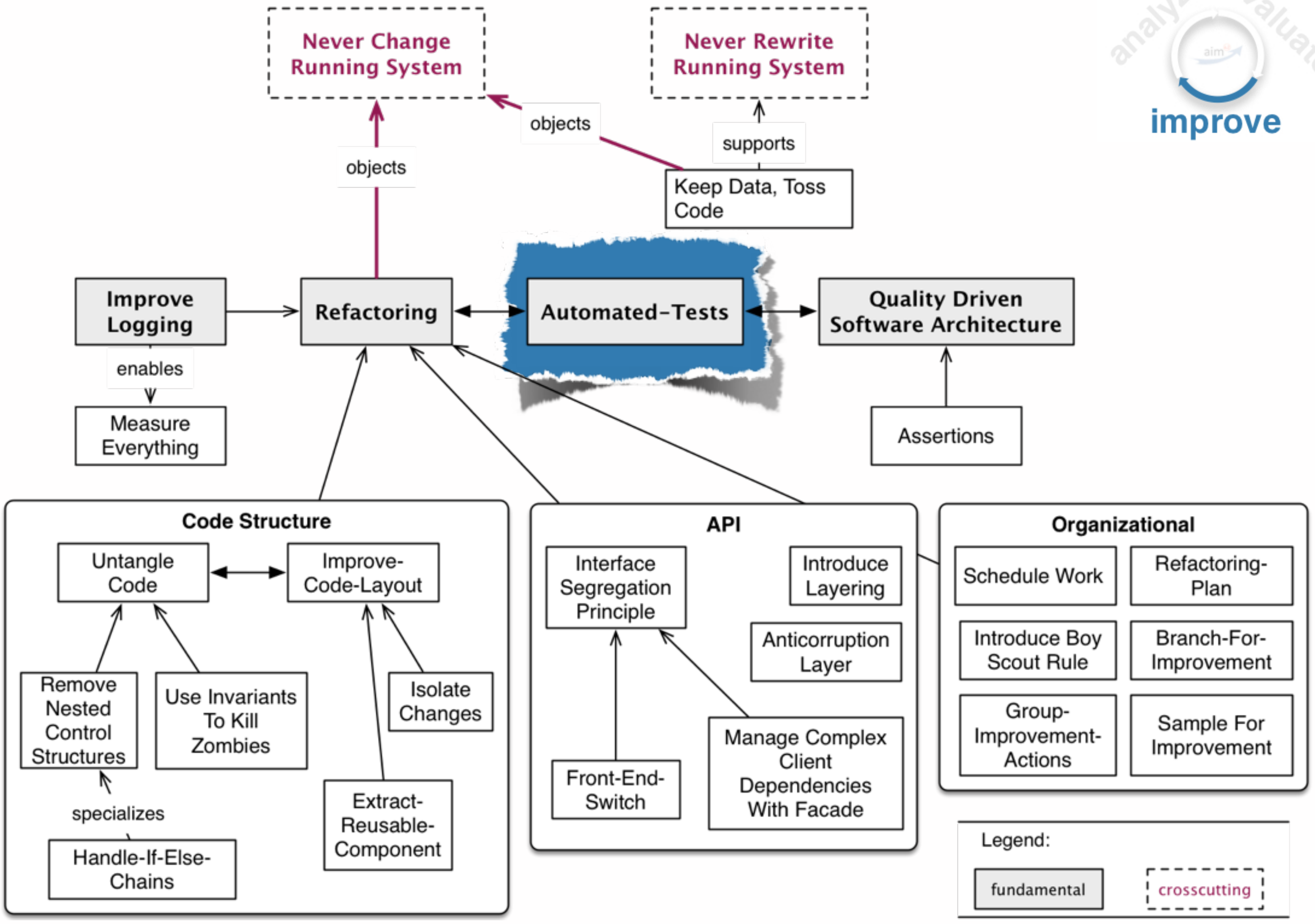
„Improve“ Practices

- Anticorruption Layer
- Assertions
- Automated-Tests
- Branch-For-Improvement
- Extract-Reusable-Component
- Front-End-Switch
- Group-Improvement-Actions
- Handle-If-Else-Chains
- Improve-Code-Layout
- Improve Logging
- Interface Segregation Principle
- Introduce Boy Scout Rule
- Introduce-Layering
- Isolate-Changes
- Keep-Data-Toss-Code
- Manage Complex Client Dependencies With Facade
- Measure-Everything
- Never-Change-Running-System
- Never-Rewrite-Running-System
- Quality-Driven-Software-Architecture
- Refactoring
- Refactoring-Plan
- Remove-Nested-Control-Structures
- Sample-For-Improvement
- Schedule-Work
- Untangle-Code
- Use Invariants To Kill Zombies



Never Change Running System
Never Rewrite Running System
objects
objects
supports
Keep Data, Toss Code
Improve Logging
Refactoring
Automated-Tests
Quality Driven Software Architecture
enables
Measure Everything
Assertions
Code Structure
Untangle Code
Improve-Code-Layout
Remove Nested Control Structures
Use Invariants To Kill Zombies
Isolate Changes
specializes
Handle-If-Else-Chains
Extract-Reusable-Component
API
Interface Segregation Principle
Introduce Layering
Anticorruption Layer
Front-End-Switch
Manage Complex Client Dependencies With Facade
Organizational
Schedule Work
Refactoring-Plan
Introduce Boy Scout Rule
Branch-For-Improvement
Group-Improvement-Actions
Sample For Improvement
Legend:
fundamental
crosscutting


Never Change Running System
Never Rewrite Running System
objects
objects
supports
Keep Data, Toss Code
Improve Logging
Refactoring
Automated-Tests
Quality Driven Software Architecture
enables
Measure Everything
Assertions
Code Structure
Untangle Code
Improve-Code-Layout
Remove Nested Control Structures
Use Invariants To Kill Zombies
Isolate Changes
specializes
Handle-If-Else-Chains
Extract-Reusable-Component
API
Interface Segregation Principle
Introduce Layering
Anticorruption Layer
Front-End-Switch
Manage Complex Client Dependencies With Facade
Organizational
Schedule Work
Refactoring-Plan
Introduce Boy Scout Rule
Branch-For-Improvement
Group-Improvement-Actions
Sample For Improvement
Legend:
fundamental
crosscutting



Never Change Running System
Never Rewrite Running System
objects
objects
supports
Keep Data, Toss Code
Improve Logging
Refactoring
Automated-Tests
Quality Driven Software Architecture
enables
Measure Everything
Assertions
Code Structure
Untangle Code
Improve-Code-Layout
Remove Nested Control Structures
Use Invariants To Kill Zombies
Isolate Changes
specializes
Handle-If-Else-Chains
Extract-Reusable-Component
API
Interface Segregation Principle
Introduce Layering
Anticorruption Layer
Front-End-Switch
Manage Complex Client Dependencies With Facade
Organizational
Schedule Work
Refactoring-Plan
Introduce Boy Scout Rule
Branch-For-Improvement
Group-Improvement-Actions
Sample For Improvement
Legend:
fundamental
crosscutting

Automated Tests

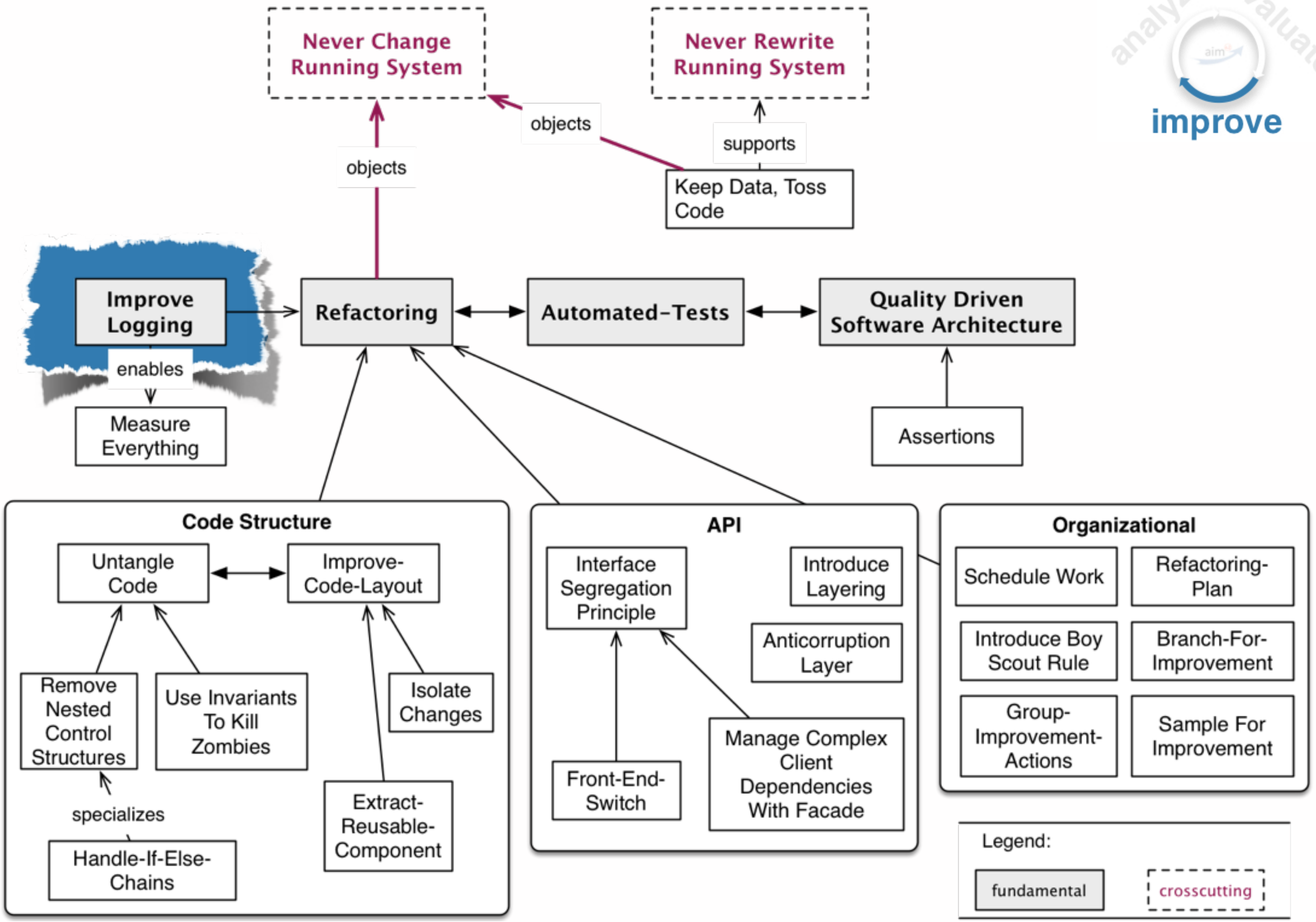
- **Risk:** Changes fail existing processes in prod
- Put this into numbers:
  - Which processes are impacted by the new feature's code changes?
  - Estimate the hourly cost of those processes failing in production
  - Estimate the probability of each process's failure

Unit Tests

- If you don’t have any, start with new features
- Reproduce each bug as a unit test
- Write small tests
- Use self-explaining test case names

Integration Tests

- Priority: **Test your API!**
  - cheaper than UI testing
  - usually not acceptance tested
- **Don’t use mocks** if you’re not forced to!
  - only if 3rd-party regularly blocks you



Never Change Running System
Never Rewrite Running System
objects
objects
supports
Keep Data, Toss Code
Improve Logging
enables
Measure Everything
Refactoring
Automated-Tests
Quality Driven Software Architecture
Assertions
Code Structure
Untangle Code
Improve-Code-Layout
Remove Nested Control Structures
Use Invariants To Kill Zombies
Isolate Changes
specializes
Handle-If-Else-Chains
Extract-Reusable-Component
API
Interface Segregation Principle
Introduce Layering
Anticorruption Layer
Front-End-Switch
Manage Complex Client Dependencies With Facade
Organizational
Schedule Work
Refactoring-Plan
Introduce Boy Scout Rule
Branch-For-Improvement
Group-Improvement-Actions
Sample For Improvement
Legend:
fundamental
crosscutting

Thesis:

Logging is the most underestimated task in IT

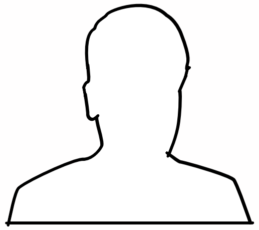
State of Logging

- Growing number of user transactions
- much larger log files
- log files distributed across multiple systems
- Increasing demand for real-time analysis

Information Types

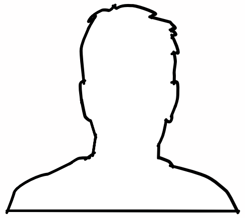
- Operational data
- Actions or state of the application
- User interaction

Stakeholders



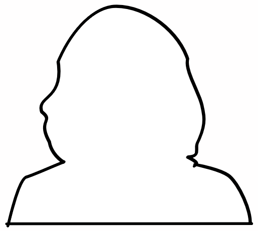
Operations

real-time health information



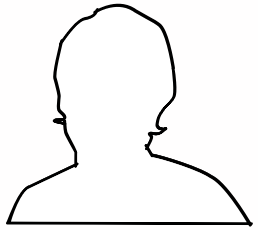
Developer

failure analysis after weeks



Product Owner

weekly usage reports



???

some complex daily report?

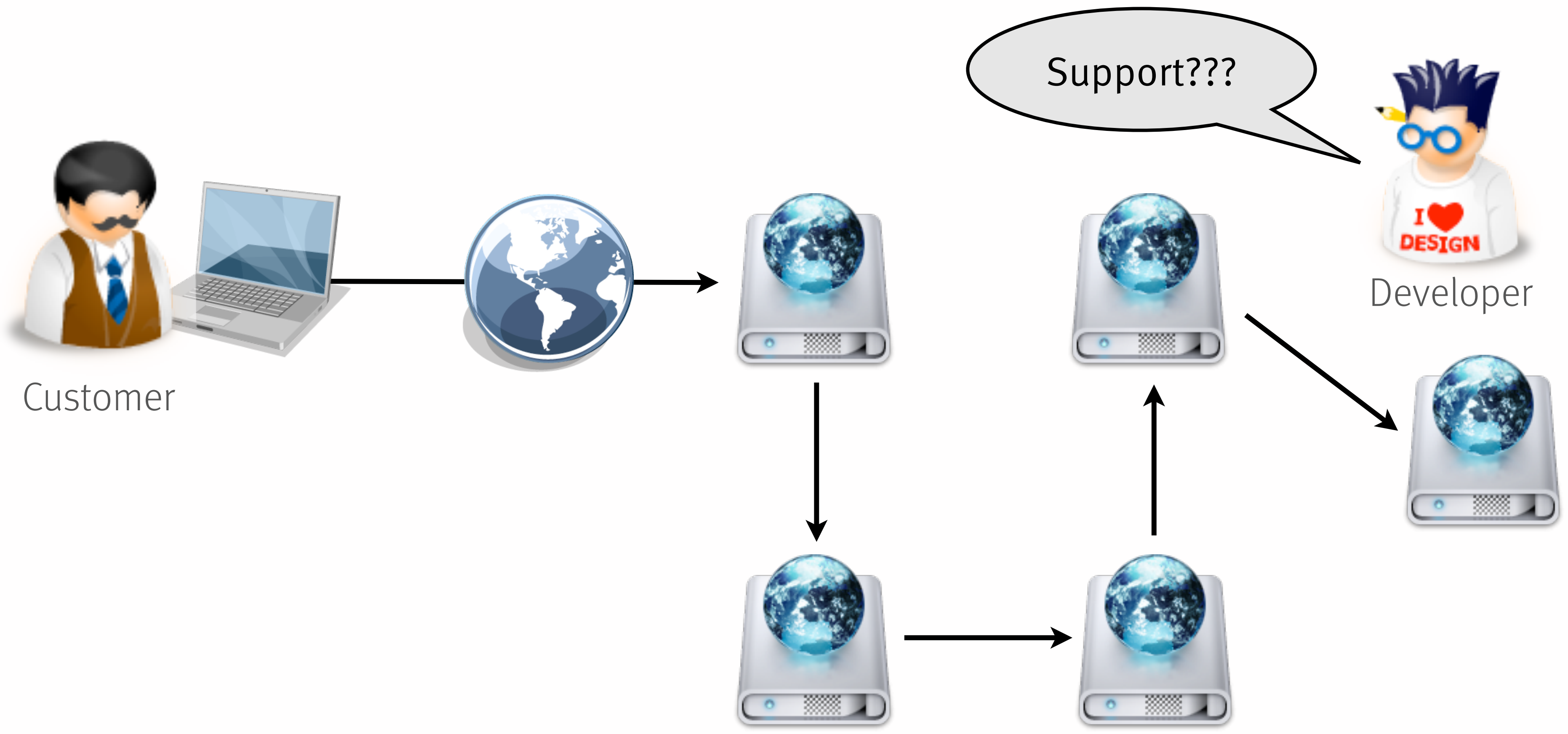
Improve Logging

- Diagnostic contexts
- Filters
- Defined log format
- Log aggregation
- CorrelationID

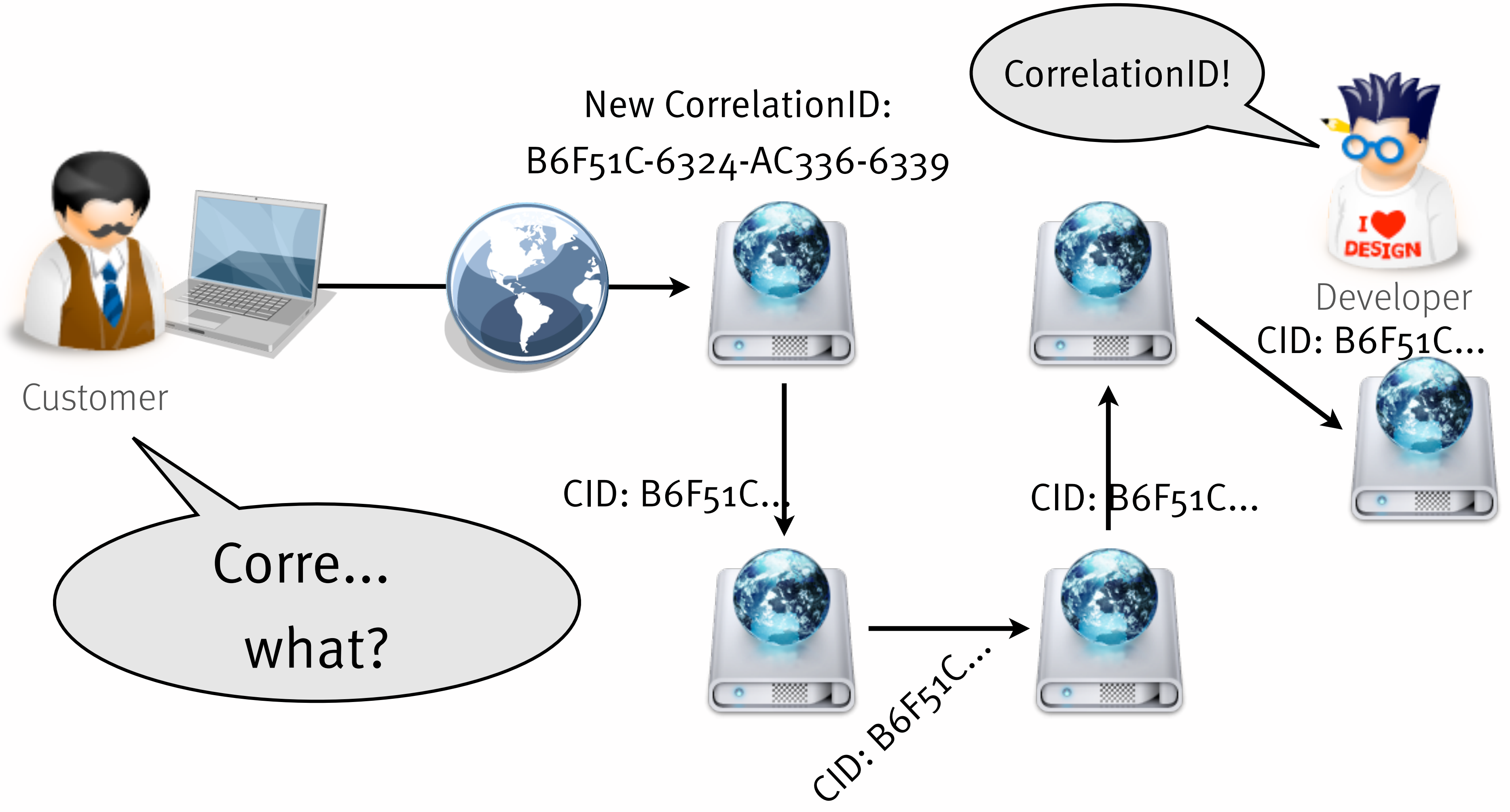
Customer Story

- A well-known German bank
- Web application for customer self-service
- Customers call support hotline for failures
- Hotline shall track state of transactions

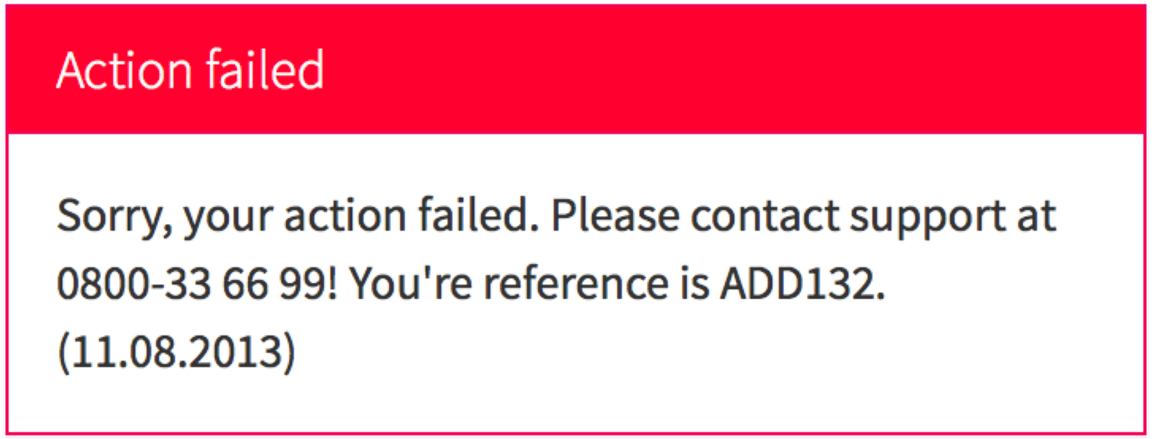
Customer Story



Customer Story



Customer Story


Action failed

Sorry, your action failed. Please contact support at 0800-33 66 99! You're reference is ADD132. (11.08.2013)

Thesis:

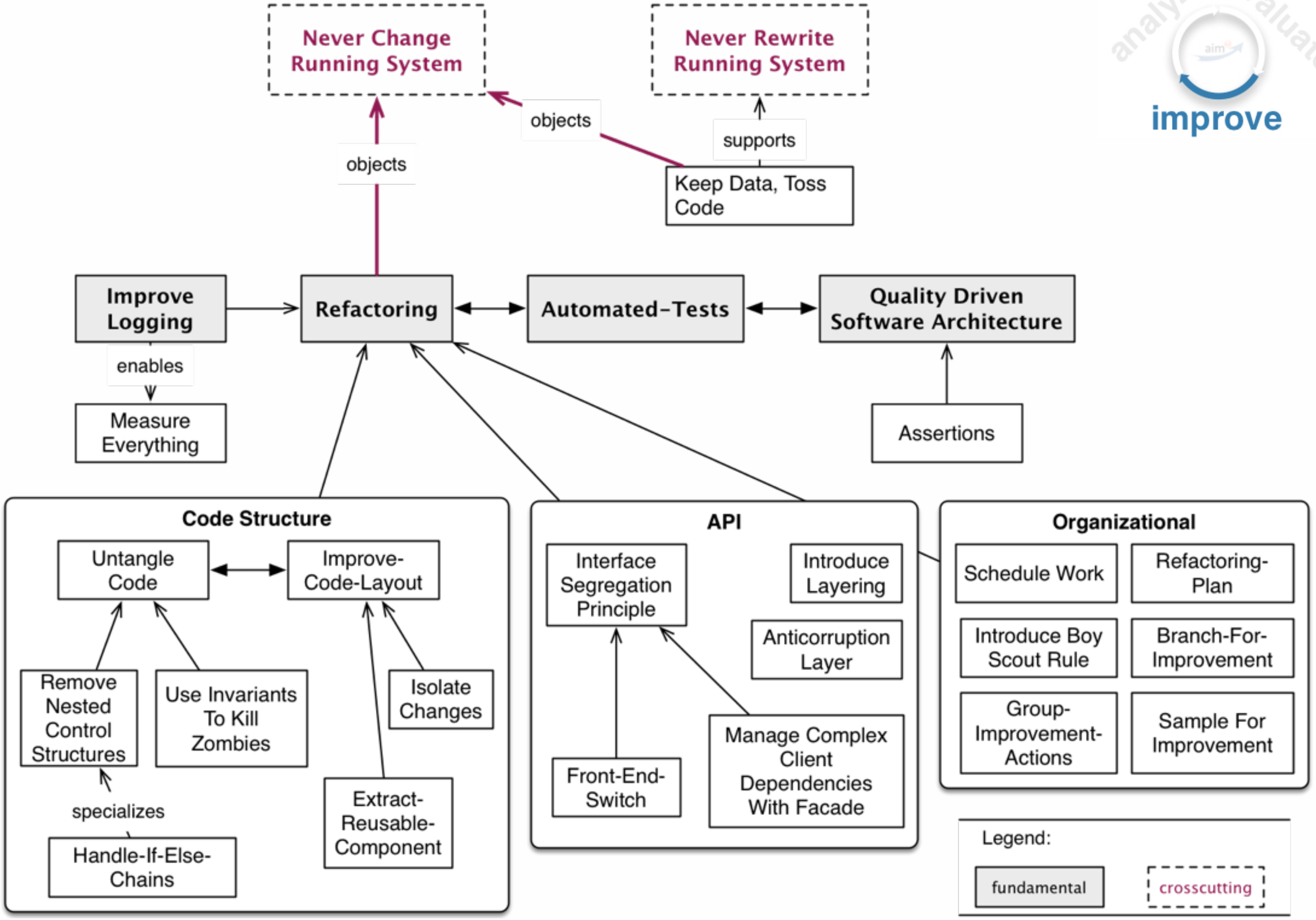
Each piece of relevant information is actually an event

“An event is anything that we can observe occurring at a particular point in time.”

— Alexander Dean, Unified Log Processing, Manning

Event Streams

Taking the next step to continuous reporting



Never Change Running System
Never Rewrite Running System
objects
objects
supports
Keep Data, Toss Code
Improve Logging
Refactoring
Automated-Tests
Quality Driven Software Architecture
enables
Measure Everything
Assertions
Code Structure
Untangle Code
Improve-Code-Layout
Remove Nested Control Structures
Use Invariants To Kill Zombies
Isolate Changes
specializes
Handle-If-Else-Chains
Extract-Reusable-Component
API
Interface Segregation Principle
Introduce Layering
Anticorruption Layer
Front-End-Switch
Manage Complex Client Dependencies With Facade
Organizational
Schedule Work
Refactoring-Plan
Introduce Boy Scout Rule
Branch-For-Improvement
Group-Improvement-Actions
Sample For Improvement
Legend:
fundamental
crosscutting

Questions?

Comments?

Dr. Gernot Starke, @gernotstarke
gernot.starke@innoq.com
http://gernotstarke.de

Alexander Heusingfeld, @goldstift
alexander.heusingfeld@innoq.com
https://www.innoq.com/en/staff/alex

innoQ

www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

Offices in:
Berlin
Darmstadt
München

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116

info@innoq.com