

# Analyzing the Evolution of Software by Change Analysis



Egon Wuchner  
[egon.wuchner@siemens.com](mailto:egon.wuchner@siemens.com)

Jürgen Salecker  
[juergen.salecker@siemens.com](mailto:juergen.salecker@siemens.com)

Siemens AG, CT T DE IT1, Munich, Germany

## Software Maintenance Nightmare – Why does it happen?

### Technical Debt

“Other debt accumulates from taking hundreds or *thousands of small shortcuts*—generic variable names, *sparse comments*, *creating one class in a case where you should create two*, *not following coding conventions*, and so on.

This kind of debt is *like credit card debt*. It's easy to *incur unintentionally*, it *adds up faster than you think*, and it's harder to track and manage after it has been incurred.”

Steve McConnell

### Reasons

- “... debt are commonly incurred in response to the directive to *“Get it out the door as quickly as possible.”* “
- *Pressure due to time-to-market*, lack of tests
- These costs due not show up at the beginning, *defects show up later*
- Because you need highly motivated and qualified engineers to put emphasis on quality

## Software Maintenance Nightmare – Why does it happen?

SIEMENS

### Technical Debt

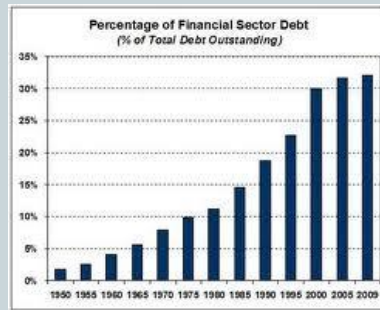
“All successful software gets changed. So if we think we're working on code that will be successful ... we need to keep it easy to change. *Anything that makes code difficult to change is technical debt.*

*Just like any other debt, the cost of paying off technical debt gets more and more expensive over time. ... Technical debt drives the total cost of software ownership relentlessly higher ... eventually we will have to pay it off or the system will go bankrupt.”*

Mary Poppendieck

### Financial Debt

- Incurs [interest payments](#)



## Software Maintenance Nightmare – Possible actions to take

SIEMENS

### Static tests

- [Coding guidelines, programming standards, code inspections](#)  
=> manual checks
- [Code analysis tools](#)  
aim at high code quality and less defects  
  
=> automated checks  
=> reduce manual inspection effort
- [Architecture analysis tools](#)  
aim at a maintainable and testable dependency structure

### Dynamic tests

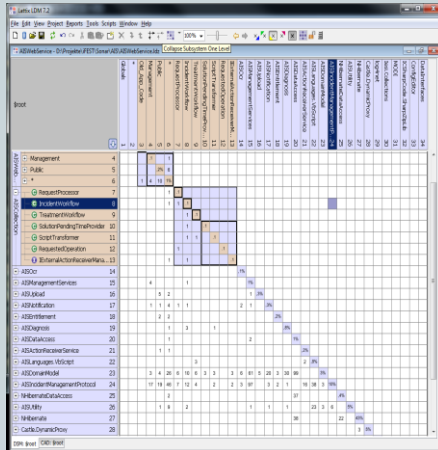
- Test, test and test – unit test, integration tests, system tests, acceptance tests
- Test-driven development

### Agile methodologies, e.g. Scrum

- efficient and necessary roles; Customer involvement
- small iterations, commitment – we fail or succeed together
- fast feedback: sprint reviews, retrospectives

## Software Maintenance Nightmare – Architecture analysis tools

SIEMENS



Page 5

### Usage scenarios

- Search dependencies, get overviews
- Monitor and [evaluate dependencies for a hierarchical and testable design](#)
- Identify [cyclic dependencies](#)
- Simulate refactorings by [what-if analysis](#)
- Take analysis results to [generate re-structured artifacts](#)

### Tools (all commercial)

- Understand
- Sonargraph/Sotoarc
- Latix
- NDepend
- (Re)Structure101

## Software Maintenance Nightmare – Emergency case scenarios

SIEMENS

No high test coverage

No code analysis, architecture  
analysis tool has been used

You have done this kind of  
quality assurance, but the size of  
your systems implies some  
technical debt to take anyway.

Results of analysis tools give  
you hints about a lot of pain  
points. Even starting with the  
main ones implies some initial  
effort.



Nevertheless,

You would like to [understand](#)  
[how the code has evolved](#).

You would like to know [why the](#)  
[code has evolved the way it](#)  
[did](#).

You would like to know [where](#)  
[to look at else when you](#)  
[change code](#).

Page 6

## Software Maintenance – Change Analysis addresses Technical Debt Interests

SIEMENS

**Goal: Increase the level of comprehension by analyzing changes.**

- Stakeholders need to understand **changes beyond text diffs**
- Each person is focussed on a fraction of work, it is hard to perceive **relevant changes beyond ones perception**.  
  
It is especially hard in case of **multiple teams or multiple locations**.
- Stakeholders need to understand changes better: **why has this part of the code changed**



- Stakeholders need guidance: **which other code is relevant when changing code**
- The solution has to be **applicable to any level of testing maturity or quality analysis tool usage**

Page 7



85% of costs are devoted to system maintenance & evolution

50% of the time is spent in the process of *understanding the code*

Koskinen, University of Jyväskylä, Finland  
<http://users.jyu.fi/~koskinen/smcosts.htm>

Page 8

**Change Analysis –  
Why are changes hard to recognize?**

**SIEMENS**



Page 9

**Change Analysis –  
Why are changes hard to recognize?**

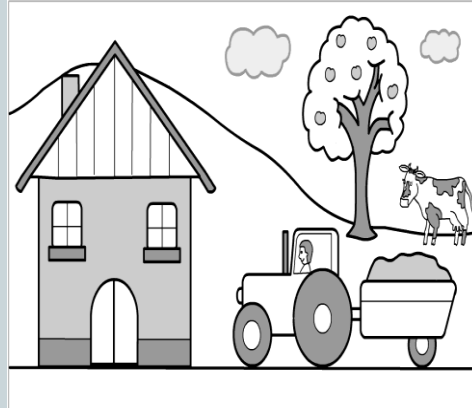
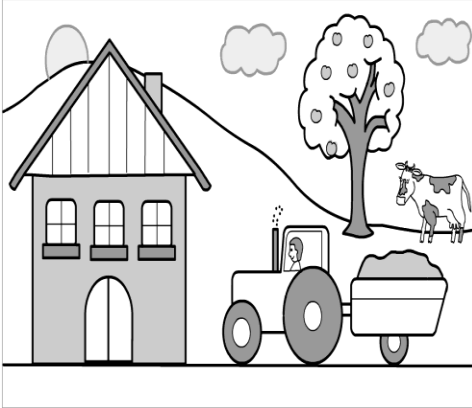
**SIEMENS**



Page 10

**Change Analysis –  
What about these changes?**

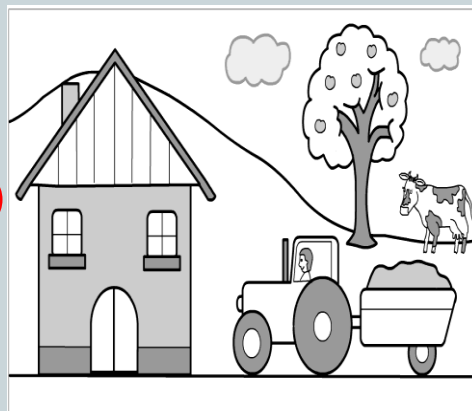
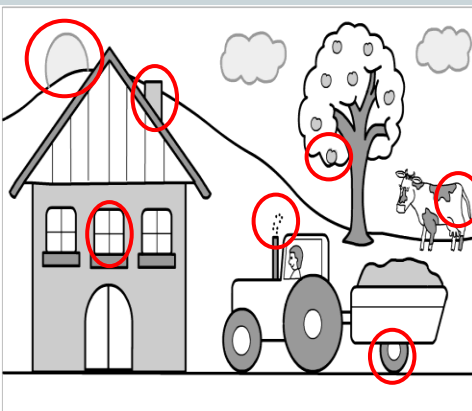
**SIEMENS**



Page 11

**Change Analysis –  
What about these changes?**

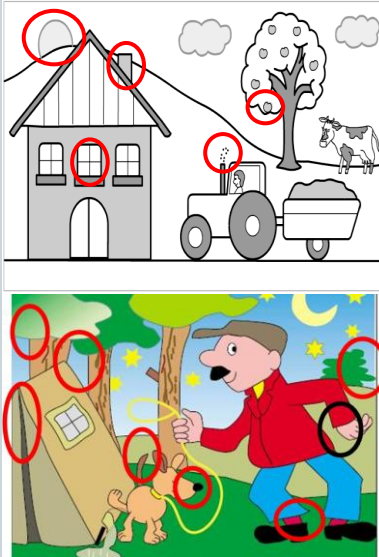
**SIEMENS**



Page 12

## Change Analysis – Why is it hard to recognize certain changes?

SIEMENS



Page 13

Because we lack abstraction w.r.t changes

But, we can start with

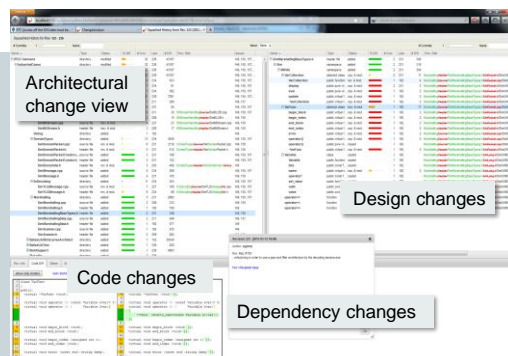
- **well-known design categories:**  
components, required/provided interfaces, dependencies
- **change categories:**  
added/changed/removed
- **parameterized annotations:**  
for design principles
  - e.g. annotation for error handling with parameters like *PropagateUp*, *TranslateTo*, *NotifyUser*, *FailGracefullyByReturningDefaultValue*, *LogError*

## Change Analysis – Levels of change abstraction

SIEMENS

### Structural levels of changes

- Changes on the level of the **architecture, design, code, tests, requirements**
  - Architecture components (artifacts, directories)
  - Design constructs (functions, class methods)
- **Relation of changes** on different levels of abstraction
- Dependency changes
  - **Calls, called-by** for design constructs
  - **Provided and required interfaces** for architecture components (artifacts, directories)



### Change categories

- **added, removed** components
- **changed** provided component interfaces
- changed required component interfaces
- changed component implementations
- added, removed, **moved, renamed, similar to**, changed design constructs

Page 14



## Change Analysis – Hot spots

SIEMENS

What has changed the most over a period of time?

### Hot spots granularity

- Architecture components, design constructs
- Traversing between component and design hot spots
- Aggregating the numbers up the hierarchy

### Hot Spot properties

- Last commit
- Number of commits
- Percentage of changes
- Number of lines of total code

Status	% Diff	# Com.	Last	# Diff	Prev. Path
modified		32	236	43187	
modified		32	236	43187	
modified		32	236	43187	
modified		32	236	43187	
modified		8	234	453	
modified		7	234	91	
modified		7	234	362	
modified		31	236	7291	
added		7	211	290	
added		8	227	80	
mov. & mod.		4	211	30	BtStreamHandlingSource/SimBLOB.cpp
mov. & mod.		2	209	10	BtStreamHandlingHeader/SimBLOB.h
mov. & mod.		4	227	23	BtStreamHandlingSource/SimBtStream.cpp
mov. & mod.		3	226	17	BtStreamHandlingHeader/SimBtStream.h
added		1	182		
added		13	234	3640	
ren. & mod.		6	231	2118	DomainTypesSource/SimDomainPacket.cpp
ren. & mod.		5	231	412	DomainTypesHeader/SimDomainPacket.h
added		3	231	198	
added		3	231	102	
ren. & mod.		3	203	406	DomainTypesHeader/SimDomainVarhSimi.a
added		5	234	299	
added		4	231	105	
added		11	234	254	
ren. & mod.		7	227	166	EnDecodingSource/SimTLGMessageFile.cpp
ren. & mod.		6	234	88	EnDecodingHeader/SimTLGMessageFile.h
added		5	231	2993	
added		2	209	333	
added		1	192	190	

Page 15

## Change Analysis – Rationale of changes

SIEMENS

Relate requirements, defects to design and code to improve impact analysis of new changes

- which parts of an artifact, design construct contribute to a requirement (e.g. hazard key) or bug-fix?
- which set of architecture components, design constructs and code lines contribute to a requirement or bug-fix?
- whom to ask about the rationale of a specific change?

Blame View for WP6/Demonstrators/TraceabilityAnalysis/ChangeAnalysis/chi/CppUndHistProcessor/\_walkRefASTree@8597

Blame Info				Content
Rev.	Author	Issue	Line	Source
8135	nko.wilbert		0	@classmethod
			1	def _walkRefASTree(cls, rootRef, par
			2	'''Recursively create the AST nod
8578	nko.wilbert	158	3	# Warning: Do not use "unique=Tru
			4	# this will break the cc
8135	nko.wilbert		5	rootEnt = rootRef.ent()
8578	nko.wilbert	158	6	rootRefKind = rootRef.kindname()
8135	nko.wilbert		7	# avoid cyclic recursion by limit
			8	undFileState.validEntUnames.dicsa
8578	nko.wilbert	158	9	if (rootRefKind == 'Name' or
			10	(rootRefKind == 'Define' and
8233	nko.wilbert	144, 132	11	rootRef.file() != undFile
			12	astNode = histcore.Container2
8259	nko.wilbert	152	13	
			14	
8578	nko.wilbert	158	15	astNode.analytics['und.unique
			16	astNode.analytics['ref.kind']
8135	nko.wilbert		17	else:
			18	startLine = rootRef.line()
8259	nko.wilbert	152	19	defRef = None
8135	nko.wilbert		20	endRef = None
8259	nko.wilbert	152	21	for ref in rootEnt.refs(refki
			22	entki
8578	nko.wilbert	158	23	if ref.file() == undFileS
8259	nko.wilbert	152	24	if not endRef:
			25	endRef = ref
			26	endLine = endRef.
			27	else:
			28	msg = (('Found mu
			29	' block f
			30	format(roc
			31	logging.warning(n
			32	if not endRef:
			33	endLine = startLine
8135	nko.wilbert		34	if rootRefKind == 'Define
8259	nko.wilbert	152	35	msg = (('Found no end
8311	nko.wilbert	154, 152	36	' for node "(
8259	nko.wilbert	152	37	format(rootEnt

Page 16



## Future Change Analysis – Reverse engineer the rationale of changes

SIEMENS

Select a subset of changes  
(e.g. over the last weeks) and  
relate them manually to an issue

- This feature could be part of the Change Analysis functionality

Reverse engineer the design and  
architecture

- Analyse **design constructs and code used in common** by many requirements



Relate requirements/use  
cases/scenarios/tests to the  
current code base automatically



- Take **coverage tools and their output** as input to a change analysis
- Take **tracing tool output** as run-time information input to a change analysis

Page 17

## Change Analysis – Maintenance Trend Analysis over the application lifecycle

SIEMENS

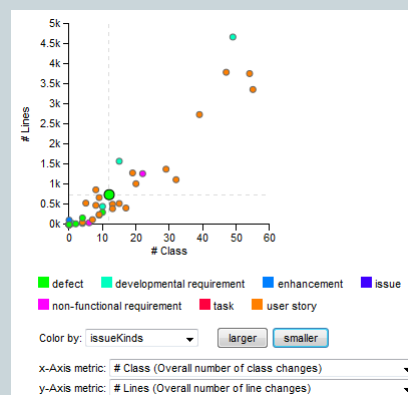
How invasive has a requirement been implemented ?  
How local has a bug-fix been? Trends over time?

### Feature invasiveness

- how many design construct have been changed for a feature implementation?
- invasiveness should not increase over time, otherwise there is certainly the need for refactoring.

### Bug-fix locality

- how many design construct changes a bug-fix implies. How 'local' (focused instead of widespread) are these changes?
- the locality should become greater over time.



Page 18

## Future Change Analysis

### Detect change (co-)relations over the application lifecycle

#### Change prognosis

- Which design constructs have often been changed in common?
- Which requirement implementations are (co-)related? On which level?
- Which changes can be clustered?

#### Interaction of issues

- Which requirements are affected by which bug-fixes? Bug-to-feature relation.
- Which manual tests are affected by a set of changes? Thus, which manual tests are recommended to do first?
- Which client software has to be build due to changed dependencies?

## Application Lifecycle Features of Change Analysis

### Several change categories. One change category does not fit all.

*Developers and architects understand changes on the respective level.*

### Understand the code in relation to issues, (recover the rationale)

*Developers and architects get the relevant information for their impact analysis.*

### Change prognosis, (co-)relate changes, (co-)relate features

*Stakeholders get guidance about possible side-impacts of changes.*

### Applicable to many scenarios

- Development within a team, multiple teams, multiple development locations
- Developing up to the first release, Maintaining a system, new features, removing defects, adapting to new technologies
- Applicable to any level of testing or quality analysis tool usage or to any development process

## Application Lifecycle Features of Change Analysis

### Determine the feature Invasiveness

*Developers/testers can assess the extent of verification&validation effort.*

### Relate bugs to features, hot spots

*Project managers can reliably spot pain points of development.*

### Relate implementation changes to manual tests

*Integration/System testers prioritize the large set of manual tests.*

### Change prognosis of latest changes, Determine feature invasiveness

*Project managers have quantifiable data to decide upon release inclusion.*

### Detect dependency changes

*Build managers get hints which client systems to build only.*

### Determine maintenance trends

*Project managers/R&D managers have quantifiable data to know the amount of technical debt*

## Future Change Analysis

### Change Analysis on Architecture and Design Models (e.g. UML, SDL)

- Using the [Atego Workbench](#) for Application Lifecycle Management
- apply change analysis of the architecture and design on modeling level
- relate changes over different modeling types
- integrate model changes and manual code changes per time period

### Change Analysis over artifact types of the application lifecycle

- determine the volatility of requirements, models etc.
- relate changes over different artifact types and phases
- e.g. relate all changes of requirements, design and code for a change request

## Current status of the Change Analysis platform

**Software pilot project at two business units**

**Possibly going open-source after successful software pilots**

**Adaptions to customer specific tool landscape**

**Supposed to present experience report in 2013**

### **Multi-language support**

- C/C++, C# (using Understand)
- Python

### **Configuration Management support**

- Subversion, Git

### **Client/Server architecture, Browser-based client**

- ongoing work on server scalability and server performance

### **Platform independence**

- implemented in Python
- Interoperable with C# (IronPython.NET)

Page 23

