Inside Software Renovation

Markus Pizka
ICSME 2018
About me & itestra

Research from 1994 – 2005
High Performance Operating Systems
Software Maintenance
e.g. Concise and Consistent Naming, IWPC 2005

Founded itestra in 2003

70 staff members at 8 locations
100+ customers in 11 countries
14 mio € p.a. revenue
Software Renovation is our USP!

- Understanding Software-Quality and Costs
- Redocumentation, migration, optimization, extension, redesign and complete reimplementation of large IT systems
- Increasing value, reducing costs

(Still) unique, challenging + a growing demand!

- Design and implementation of custom and mission critical business systems

Fun … important … but hard to differentiate!
Some say there is none

Fellow researches:
“Great ideas. But can’t be done. The next feature will always have higher priority.”

Professor:
“You are so young! Why do you want to deal with old COBOL and alike?”

Eric Schmidt:
“Don’t bother with the old stuff but with new technologies.”

… and a google search [Software Renovation] mainly delivers “home remodeling software”
Reality – enormous demand

Very large systems are in place virtually everywhere

50–100 MLoC of backend code within every average larger bank
In every domain and every industrial resp. emerging geographic region
In total: billions and trillions of LoC in various (older) languages, frameworks, …

Of course, most of this is “old stuff”, built in the 60s, 70s, 80s, 90s, 00s, …

The backbone of business and everyday processes

Suits the needs (mostly), stable, reliable
Contain otherwise unknown knowledge about processes and rules
Often with interesting solution ideas for logical or technical challenges.

They will not be replaced neartime but persist and continue to grow

50–100 MLoC means 750 – 1.500 Mio € costs
Means: 1,7 Mio PD ~ 1.000 highly productive people from day one over 8,5 years!
Knowledge Gaps

Ever programmed in COBOL?

... Spectra ASM, PL/I, FORTRAN, VaGen, CoolGen, RPG, SAS, Delphi, Gupta, Struts, XCOBOL, Delta or homegrown macro preprocessors.

COBOL disadvantages and advantages?

– Global vars, no functions, no parameters, etc.
– Limited abstraction, no reuse, no libs, arrays only, even quicksort difficult
+ Less possibilities to break it 😊

Former design patterns?

Stepwise processing with IPO (input/process/output)
Single program per logical transaction ... today: “micro service;:-)”
CICS and IMS transaction handling

Infrastructure costs of a large application?

Usually per-pay-use of CPU, up to millions of € p.a.
Renovation is needed

See
D.L. Parnas: Software Aging
M.M. Lehman: Laws of Software Evolution
S. Eick: Does Software Decay

Clearly
Requirements, technologies and expectations change
Large applications are not evolved at same pace
⇒ loss of value relative to costs
⇒ decreasing economic efficiency = justification
Not limited to “old” systems

Same in many “new” Java, C#, SAP systems
Unacceptable Performance
Unstable
Frameworks outdated during development
7 levels of indirection
Still null-pointer derefs
Awful data model
Experiments with all kinds of new techniques
Missing functionality
…

There are Java and SAP projects that cost 200 Mio € or even 1 Bil € with urgent need for renovation on day 1.

All of this needs to be renovated to (re)establish an acceptable cost / value ratio.

Software renovation will soon become bigger than development 😊
Need and use in practice
Commercial systems differ from OSS

Other (weird!) technologies

Not written by skilled “techies”
Excessive duplication (50% - 90%) and other defects
Sometimes very weak (and reinvented) algorithms
Data models far from normalized (e.g. address1 to address10)

Shaped by personal interests

Becoming indispensable (no doc, as complicated as possible on purpose)
Experimenting with new technologies (CASE tools, distribution, …)
Owned by a certain department

Influenced by biz, legal and finance
Ad hoc changes, compromises and tech. ideas by non-tech staff

When will ZNDFA-OK be initialized with 0?
Heavily impacted by the environment

1970s: **memory counted in kBytes**, hardware expensive, compilers etc. not very powerful

- computers used to store data, sequential files, basic processing, simple structures (GOTO), I/O

1980s: offices and shops (EU ;-) **open 8 AM – 5 PM, no open source software yet**

- work collected during the day in interactive mode and processed overnight (batch)
- all kind of proprietary “tools” and building blocks by some local / small companies

1990s: **first networks with sufficient speed**

- “applications” start exchanging data (files, DB replication) … but are still all-in-one monoliths

2000s: **no smartphones, yet** (iPhone started 2007)

- no mobile computing, not much omni-channel business or end-user centric process & UI design

Renovation is not just about unusual languages and quality defects! It affects all levels.
Options are limited

✖ Quality analysis tools (Sonar, …) → You will not get lighter by buying a scale

✖ Ostrich – ignore it and wait → Watch your problems grow! ;-)

✖ Translate to a different language → Tackles only small part of the problem, worsens others

✖ Wrapping → The pathway to high costs
  e.g. displaying full account status by firing 20 transactions

✖ Buy a packaged solution → There are no std. products for differentiating biz. processes

✖ Build new from scratch and cut over → Very large projects. High risk for disasters!
  In particular: double maintenance and moving target.

© itestra 2018 | Inside Software Renovation
Real customer pain

Yes: costs

Development: 2-5 times higher costs on average
Example: any CR is a multiple of 100 PDs
Expensive infrastructure (mass data x weak algorithms x monopolies)

But even stronger: TIME to system

Since mid of 2010s: “digitalization” – time to innovate is crucial!
You might lose your customer base without the new app and services

Lack of human resources

Hire “old” people or make young professionals work on old stuff?

Survival in mergers & acquisitions

E.g.: bank was sold and survived because of renewed IT landscape
Renovation is the key option

 Completely renew
 Towards a target with an attractive cost/value ratio

 In place
 There is always just one system
 No double maintenance or moving target

 Reuse what’s reasonable
 Artifacts, interfaces, knowledge, skills

 Possibly over a long period of time
 Stepwise
Overall Process

Strategic Design
• Assessment of the current state - Software HealthCheck
• Development target picture
• Design of a roadmap

Decision Making

Proof-of-Concept - “get things moving”
• Demonstrate feasibility
• Validate estimates

Implementation
• Iteratively with partial deployments and shutdowns
• Mixed Operations

Cleanup or final shutdown

- app. 20-50 PDs
- 0 PDs, 1 – 7 years!
- 100 PDs, ½ year
- 2 – 10 years
Access to “C”-level is a prerequisite

Usually CIO.
Must be proactive and have guts.

Assessment has to provide a link btw. software and economics

We use analysis tools but this is just a small part.
Investigation of current software costs and development performance as well as business processes

Several people, roles and departments must become convinced

Large scale renovation has a strong impact on the overall organization.
There will be very different opinions, fears and even actual loosers.
Hardly any organization is already used to think in projects > 1 to 3 years.
Usually the target picture is unclear!
Try to think at least 10 years ahead
- No sense in starting to build a solution solely for today
- What will the business and technology look like

There are virtually always possible ways.
Bring together
- Functional priorities and technical dependencies
- Available budget and organizational aspects
Implementation - key elements

Finding a way for mixed operations – old and new

Prerequisite to AVOID A BIG BANG
“Cutlines” must be found
Critical obstacles: performance (i.e. nsec vs. msec), technologies, data store
Usually along Use Cases and not technical layers (DB-first makes no sense!)

Reverse Engineering

One important key to success: because usually no-one could delivery or even specify the requirements
Strongly reduces costs and effort of the overall project – major difference compared to new development!
Sources of information: screens, reports, code, people … sometimes documents

B.t.w.: “agile” not so important because renovation ≠ learning problem
Particular tools

Estimation of the net size of a system (“intellectual” content)

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>44</td>
</tr>
<tr>
<td>non-functional</td>
<td>29 (65.4%)</td>
</tr>
<tr>
<td>technical</td>
<td>9 (20.9%)</td>
</tr>
<tr>
<td>MOVES</td>
<td>1 (2.27%)</td>
</tr>
<tr>
<td>functional</td>
<td>6 (13.6%)</td>
</tr>
<tr>
<td>SQL</td>
<td>3 (6.82%)</td>
</tr>
<tr>
<td>comments/whitespace</td>
<td>1 (2.27%)</td>
</tr>
</tbody>
</table>

Figure 6.3: Categorisation output for all files in the case study. The files are aligned on the x-axis and sorted by the amount of functional code, while the y-axis covers the relative category distribution.

number of statements: 16
number of MOVES: 5 (50.0%)
Support for testing new vs. old
Forward and backward tracing

Makes sure nothing is left out and allows to track progress.
Examples
# Bank: 10 years renovation project

<table>
<thead>
<tr>
<th>Year</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2017</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2021</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### New functionality

- Modus „opportunistic“
- Modus „active“ und „opportunistic“

### Renovation
50% size reduction though increased functionality

New solution

Old client (Powerbuilder)

Old backend (CBL)

Old interfaces (C++)
Insurance

Reverse Engineering

Use Cases

Business Rules

3 years mixed operation: old and new parts work on the same database.
Complex scenario: “Group Partner Data”

Situation

3.5 Mio Lines of Code  COBOL and z/OS
> 400 modules accessed by other applications
> 5,000 call locations in >2,000 foreign modules
50 Mio partner records
Millions of access per day

Goal: Java, non-host, services, … fast

Elements of the Strategy:

Vertical renovation according to business domains
2 synchronous data bases
„Write-Trough“ mechanism
Test automation
What’s next?
itestra Innovation Lab

More and better tools to support software renovation
- AI supported assessment of software quality
- Identification of generated and 3rd party components

Establishing software renovation as a discipline