Open Source Software Components (OSS)
Technical Debt and Scenario Analysis

Problem, Goals, Conceptual Model, Calculation Models, Evaluation, Future

Christian Bucholdt, UBS AG, IT Progam Manager

February 27, 2016

Agenda

1. **Problem**
2. **Goals**
3. **Conceptual Model**
4. **Calculation Model**
5. **Calculation Example**
6. **Challenges**
7. **Future Work**

Open Source Software (OSS) and Requirements Engineering

Non Functional Requirements	Open Source Software
Requirements Engineering in most cases takes place for existing applications	Open Source Software components are a major part of the application code base
Non functional requirements are the "poor cousin" of Requirements Engineering work	Open Source Software components have significant impact on productivity of software development (reuse of existing source code)
Improvement of existing non functional requirements beyond performance not in focus	Negative impact of Open Source Software components mostly not assessed
Project budget has focus on business value instead of maintainability and operations	Project and Product Risk

Situation in a global Bank

IT Facts	Challenges
Ca. 12’000 – 17'000 people out of 60’000 total bank employees working in IT	Usage of open source components to a significant extend in software development.
IT distributed in locations such as CH, PL, UK, US, SG, HK	Business applications can contain between just a few up to hundreds of open source software components
Developing and maintaining 5'000 – 7'000 applications and application components	Established process for introduction of new OSS components, but incomplete and reactive lifecycle management
Using about 10’000 or more different OSS components in about 30'000 component versions.	No consistent picture of existing and currently used OSS components and its versions.
	No pro-active lifecycle management for the entire portfolio of OSS components has been established

Goals

Goals		Known Concepts
Measure	Measure the problem exposure and provide transparency	**Technical Debt** for measurement on OSS component level
Advise	Provide advise for corrective actions	**Scenario Analysis** for corrective actions on Business Application level

Note: Multiple specific category scores will be used instead of one "simplified" score.

Technical debt (TD) introduction

Definition

Technical debt is a metaphor [Cunnigham1992] [Kruchten2012]

Describes the fact that during software development tactical solutions will be applied in order to reach short-term goals

This is on expense of a sustainable long-term solution.

Technical debt in software addresses topics such as code smells, code duplication, unwanted code structures, anti-patterns, etc.

Technical debt metaphor has also been applied to architectural debt in the research literature.

Standard Measures

There are two technical debt measures, e.g. [Curtis2012]:

- Principal: What is the effort/cost to fix the debt problem once for all
- Interest: What is the additional effort/cost (per release, per change) for ongoing development if the identified debt continues to stay in the code.

Based on expert effort estimations or historic source code repository data

TD approach to OSS

Decline of Standard Measurement Approach

OSS component lifecycle management is about the usage of components, not the actual source code of business application or OSS component

Measurement of principal and interest might be possible within a development project with a stable project team and knowledge.

It will not work if the "debt" has accumulated over many years and multiple generations of developers and changing offshore resources

Different aspects of OSS components have to be considered such as community, vulnerabilities, functionality and license model.

New Measurement Approach

Make technical debt measurement independent from particular developer knowledge and estimations

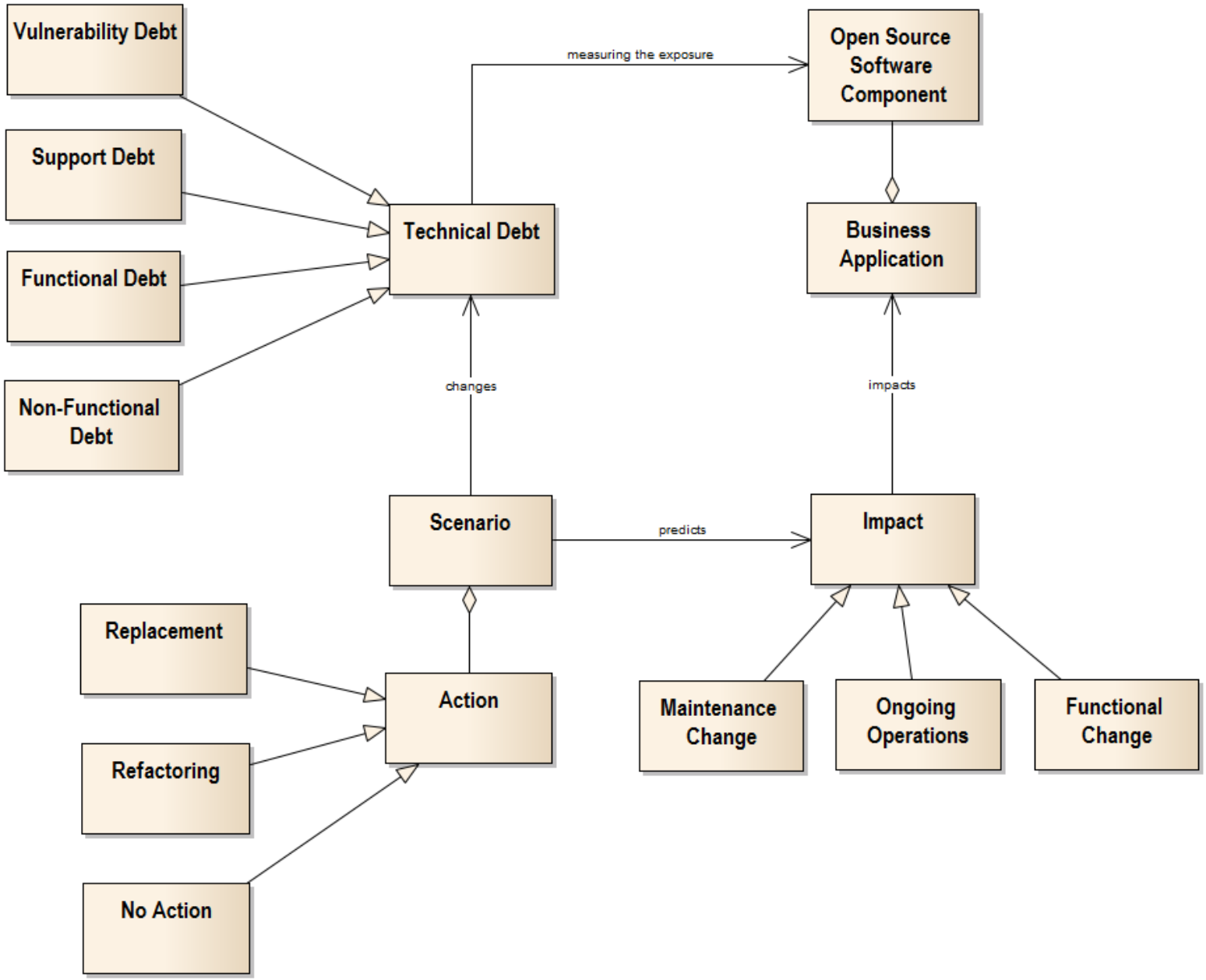
Provide different categories of technical debt: Support Debt, Vulnerability Debt, Functional Debt, Non-Functional Debt

Identify human understandable parameters for basic measurement of OSS technical debt

Calculate a synthetic, not effort based, measurement for each of the debt categories per OSS component in a business application

Provide a business application perspective over all used OSS components based on future scenarios

Conceptual Model 1/2



Conceptual Model 2/2

Business Application: A software system that provides a particular business value by performing a functionality. Such an application contains OSS components.

Open Source Software Component: A reused software component with available source code that was developed outside of the company and is also used by other users outside the company

Technical Debt: The exposure of an business application by usage of specific open source software components

Vulnerability Debt (VD): Technical debt due to security vulnerabilities in OSS components, currently and over the lifecycle

Support Debt (SD): Technical debt due to the exposure to the contributing and using community of an OSS component

Functional Debt (FD): Technical debt from a lack of functionality due to a legacy OSS component compared to the newest version or component

Non-functional Debt (NFD): Technical debt from a lack of performance or stability of a legacy OSS component compared to the newest version or component

Scenario: This represents the overall concepts of how a business application is operated in regards to OSS usage.

Impact: This represents the outcome of a particular scenario action. The impact is directly related to the business application in regards to OSS usage.

Functional Change (FCI): This indicator is for the fact where further functional changes will be applied to a system with a legacy component over the entire application lifecycle.

Ongoing Operations (OOI): This indicator is about running a system with a legacy component. It can be affected by occurrences of downtime, security breaches or legal dispute.

Maintenance Change (MCI): This indicator addresses maintaining technical or configuration changes on an existing business application without functional

Action: This represents the possible actions to be addressed in the scenario analysis.

Refactoring (REF): This indicator represents modernization using refactoring of an existing OSS component such as removal of a deprecated method or a version upgrade.

Replacement (REP): This indicator represents the case where refactoring is not possible or suitable, e.g. complete replacement of a component by a new component

No Action: Doing nothing leads to further decline of the business application

Excurse: Risks vs. Scenarios and Impacts

Original Idea

Calculate commercial impact of operational risk of business applications caused by OSS components Operational Risk according to Basel II)

Challenges

No historic data are available to validate a cause and impact relation of OSS components and operational losses. → The problem persists even if near-loss events are considered

The **proof of correlation** is difficult if and how much of a historic loss might be caused by legacy OSS component(s)

It is also difficult to calculate the technical debt interest of future product work that might be caused by OSS components. There is a similar

Scenario Analysis

Scenario Analysis

Definition

Consideration of future events and their impact to alternate outcomes

Multiple ways and steps to get to this future outcomes

No extrapolation from historic data, but observable or imaginable steps

OSS Scenarios

Based on current status

Application of actions:

- Replacement
- Refactoring
- No Action

Alternate outcomes:

- Ideal OSS component management → all best current OSS component versions
- Worst case scenario of no OSS improvements → no action leads to further decline, not just status quo
- Improvement by limited changes of just one or a few OSS component changes

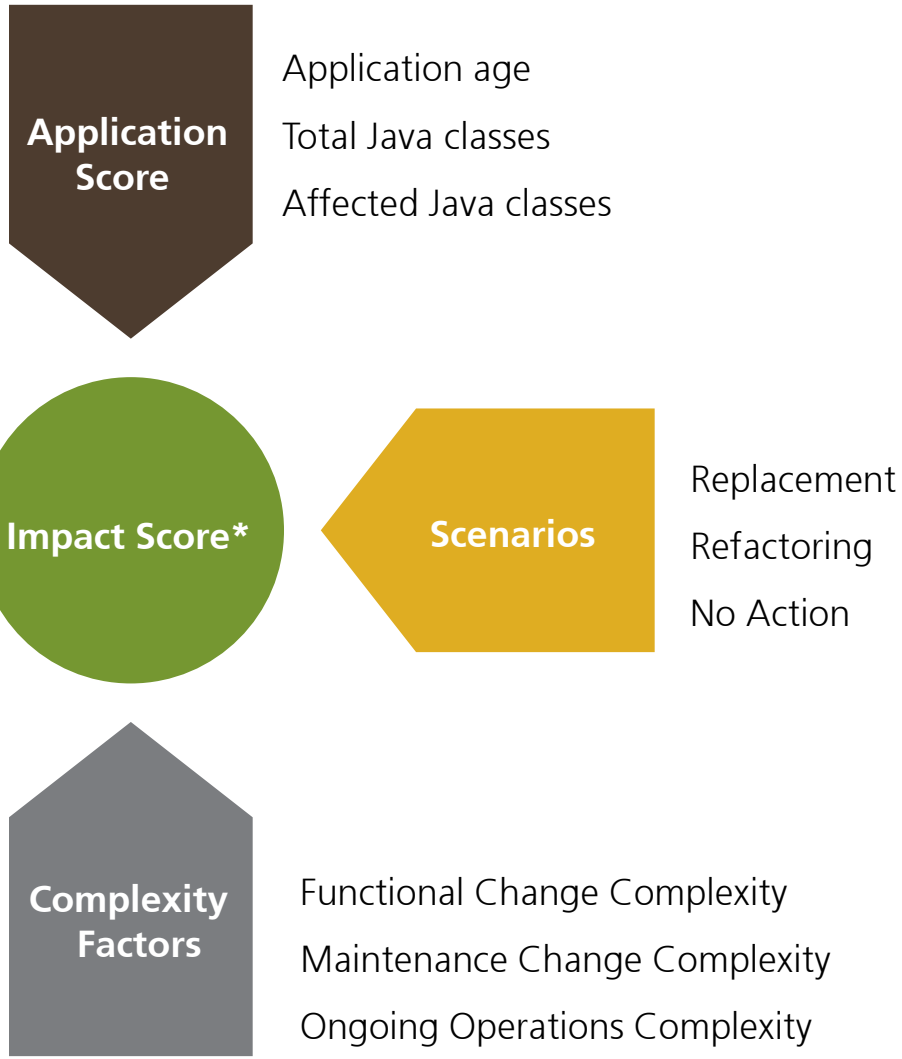
Calculation Model Overview

Impact Score* = Application Score * Consolidated Technical Debt Score * Complexity Factor

*** Functional Change Impact**

Maintenance Change Impact

Ongoing Operations Impact



Application Score

Formula:

Application Score =

Application age * (Affected Java classes / Total number of Java classes)

Parameters	Description
Application Age	Years of an entire application lifecycle since creation or last major refactoring
Affected Java classes	Java classes affected by OSS components, either by one component or by all components
Total Java classes	Total number of Java classes in the business application (OSS classes not counted)

Consolidated Technical Debt Score

- Calculate average of scores for each technical debt category (SD, VD, FD, NFD)
- Define weight of each of the scores based on priorities (0 – 1 → total weights = 1)
  - Current tests with higher weight for SD score based on problem assessment in the bank
- Calculate weighted score and summarize

Example

	Average TD score	Weight	Weighted score
SD	0.8	0.4	0.32
VD	0.1	0.2	0.02
FD	0.2	0.2	0.04
NFD	0.2	0.2	0.04
			0.42

Use weighted score for further calculation as ***Consolidated Technical Debt Score***

Support Debt (SD) Calculation

Formula:

Aggregated SD score = (NUV*Weight1 + NDS*Weight2 + NCS*Weight3 + NVD*Weight4) / 4

Parameters	Description
NUV	**Normalized component Usage per Version** Downloads or usages of component versions compared to previous version downloads or usages
NDS	**Normalized Delivery Sustainability** Component versions per year compared to total number of versions
NCS	**Normalized Contributors Sustainability** Contributors of a component version compared to previous version
NVD	**Normalized Version Distance** "Distance" in count of versions from current version to newest version in comparison to distance in years

OSS component Support Debt parameters

Parameters of support debt: downloads, dependencies, contributors, versions, years

Parameter	Description
Number of downloads current / last trend change version	Number of downloads of a OSS component version (e.g. from Maven Central repository)
Number of usages current / last trend change version	Number of usages of this OSS component version in another OSS component
Number of downloads over complete lifecycle	Number of downloads of an OSS component over complete lifecycle counting all versions
Number of versions per year	Number of releases of an OSS component per year (major, minor)
Number of versions over complete lifecycle	Number of OSS component releases over complete lifecycle of the OSS component
Numbers of contributors current / last trend version	Contributors in OSS component development of the OSS component
Distance count of versions	Number of releases of OSS component between currently used version and newest version (counting all major and minor releases)
Distance count of years	Years between release of currently used version and release of recommended version)
Total years of lifecycle	Number of all years from first release of OSS component until newest version

Details OSS component Support Debt Calculation

- Normalized OSS component usages per version (NUV): Specifies the support of the usage community of the OSS component. Decline in usage numbers of a component version are an indicator for declining community support
  - Calculation: = 1 – (Number downloads current version / Number of downloads at last trend change)

Normalized delivery sustainability (NDS): Specifies the release of versions of the OSS component over the complete lifecycle of a component. Longer times between releases (compared to the past of the lifecycle) or no releases indicate a declining contributor community

- Calculation: = 1 – (Number of versions per year / Number of versions over complete lifecycle)

- Normalized contributors sustainability (NCS): Specifies the number of contributors of the OSS component over the complete lifecycle of an OSS component. A small number of contributors can be a risk.
  - Calculation: = 1 – (Number contributors current version / Number contributors at last trend change)

Normalized version distance (NVD): Specifies the "distance" between the latest available version of an OSS component and the currently used version in a business application.

- Calculation: = (Distance count of versions/Distance count of years) / (Total count of versions over complete lifecycle/Total years of lifecycle)

→ **Aggregated SD score** =

(NUV*Weight1 + NDS*Weight2 + NCS*Weight3 + NVD*Weight4) / 4 (value range: 0–1)

Vulnerability Debt (VD) Calculation

Formula:

Aggregated VD score = (CVS*Weight1 + NCV*Weight2+NVVD*Weight3) / 3

Parameters	Description
CVS	**Currently used version Vulnerability Status** Version vulnerability status and its threat level
NCV	**Normalized Component Vulnerability** Vulnerabilities over complete component lifecycle and per version
NVVD	**Normalized Vulnerability Version Distance** "Distance" in count of versions from current version to recommended version from a vulnerability perspective in comparison to distance in years

OSS component Vulnerability Debt parameters

Parameters of functional debt: vulnerabilities from NVD, CVSS score, versions, years

Parameter	Description
Vulnerable	Currently used OSS component version has vulnerability (0 ¦ 1)
Vulnerability Threat Level Score	Overall CVSS Score calculated from CVSS v. 2
Average vulnerabilities per version	Average of vulnerabilities over all versions of an OSS component
Total number of vulnerabilities	Number of all vulnerabilities over the lifecycle of an OSS component
Number of versions over complete lifecycle	Number of OSS component releases over complete lifecycle of the OSS component
Total count of versions until recommended version	Number of all OSS component releases from beginning until recommended version
Distance count of versions	Number of releases of OSS component between currently used version and newest version (counting all major and minor releases)
Distance count of years	Years between release of currently used version and release of recommended version)
Total years until recommended version	Number of all years from first release of OSS component until newest version

Functional Debt (FD) Calculation

Formula:

Aggregated FD score = (NFS*Weight1 + NFV*Weight2 + NDF*Weight3) / 3

Parameters	Description
NFS	**Normalized Functionality Score** Potentially missing features compared to currently used features (classes) of an OSS component
NFV	**Normalized Functionality Volatility** Changes of the interfaces and functionality of an OSS component during its lifecycle
NDF	**Normalized Distance in Functionality** "Distance" in count of functionality (classes) between current version and recommended version from a functionality perspective in comparison to distance in years

OSS component Functional Debt parameters

Parameters of functional debt: OSS component "features" = API classes

Parameter	Description
Currently missing features	Features that a development team would like to use, but can not because of legacy OSS component version
Number of used features	Features of an OSS component that a development team is currently using in its current OSS component version
Feature volatility per version	Average number of API (classes) changes per OSS component version
Total feature volatility	Total number of API (classes) changes of an OSS component over complete lifecycle
Distance count of features	Number of API (classes) changes between currently used OSS component version and newest OSS component version
Total number of features	Total number of features over complete lifecycle of OSS component

Non Functional Debt (NFD) Calculation

Formula:

Aggregated NFD score = (NSS*Weight1 + NBV*Weight2 + NSD*Weight3) / 3

Parameters	Description
NSS	**Normalized Stability Score** This score addresses the currently open bugs of an OSS component in currently used version and over complete lifecycle
NBV	**Normalized Bug Volatility** Score of bugs per individual versions compared to bugs over OSS component lifecycle
NSD	**Normalized Distance in Stability** "Distance" between current version and recommended version from a bug perspective in comparison to distance in years

OSS component Non Functional Debt parameters

Parameters of functional debt: Bugs, version

Parameter	Description
Open bugs in current version	All bugs of currently used OSS component version that are not fixed/closed yet
Total number of open bugs over lifecycle	All bugs over all OSS component versions that are not fixed/closed yet
Bugs per version	Average of all bugs (open and closed/fixed) per OSS component version
Total number of bugs over lifecycle	All bugs over all OSS component versions regardless if open or closed/fixed
Total open bugs until recommended version	All bugs of all OSS component versions that are not fixed/closed yet from first version until the recommended version for installation
Open bugs in recommended version	All bugs of recommended OSS component version that are not fixed/closed yet

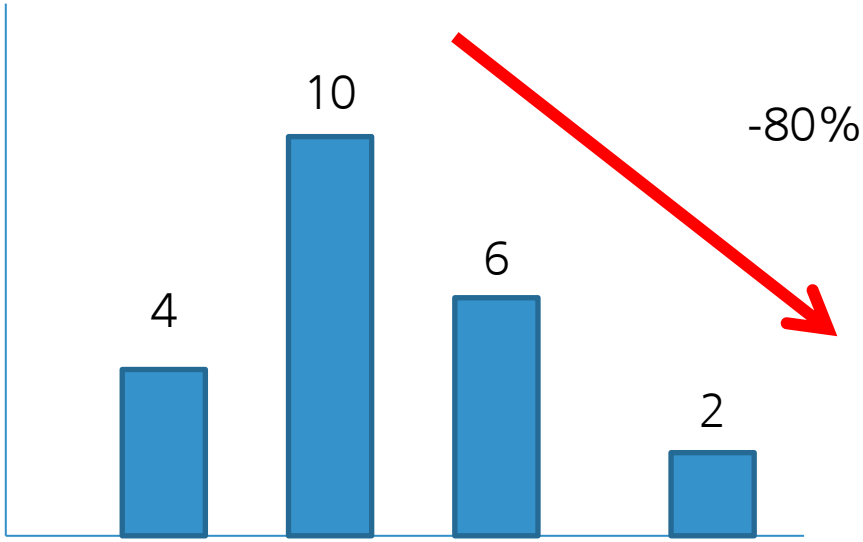
Complexity Factors

Different complexity factors for different impacts:

- Functional Change
- Maintenance Change
- Ongoing Operations

Complexity Factor	Formula
Functional Change Complexity	last trend x (business requirements last release / total number of business requirements)
Maintenance Change Complexity	last trend x (technical changes per year / total number of technical changes)
Ongoing Operations Complexity	last trend x (production incidents per year / total number of production incidents

Excurse: Last trend explanation



We check for usages or downloads or contributors for each version or year.

We look for the last trend change (peak of 10 in the picture).

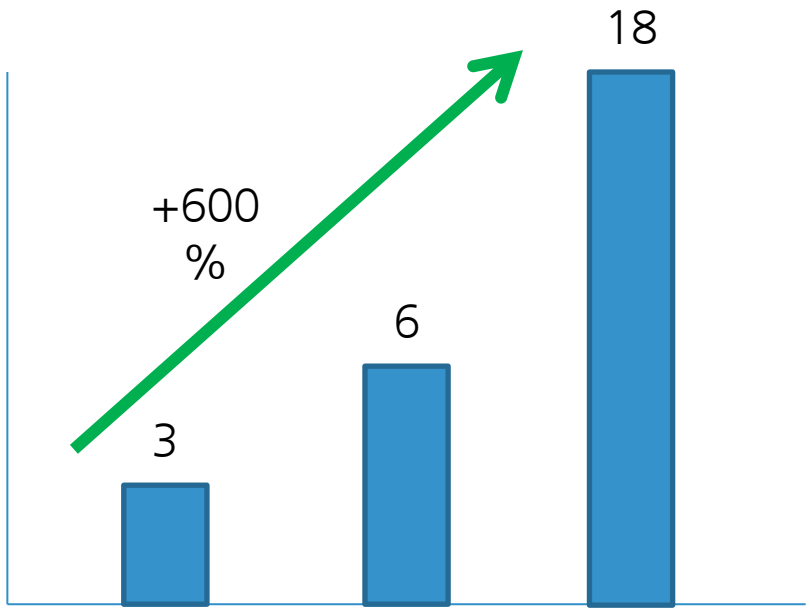
We check the trend until current version (e.g. -80% decline).

We check for usages or downloads or contributors for each version or year.

We look for the last trend change (peak of 10 in the picture).

We check the trend until current version (e.g. +600% increase).

If increase >=100% set result to 0.



Note: last trend = Trend of input parameters (decline = 1; same = 2; increase = 3

Example

Example

The example shows the following:

- Anonymized Excel simulation with 25/50 OSS components
- Calculation of technical debt for each component
- Calculation of application score
- Calculation complexity factors
- Comparison of current, worst and ideal world of application between scenarios
- Impact of individual OSS component change

Link to Excel

Challenges

Challenges	Description
Availability of parameters	Have they been captured in the past? Are they available going forward?
Automation of parameter capture	Are there technical interfaces that can be accessed? Which permissions are required? Implementation effort?
Broad validation	Validate multiple hundred applications within the bank Validation beyond the bank
Comparison	What can be considered "good" or "bad" between applications?

Further work

Open Items	Description
Validation	Evaluation of suitable parameters and formulas Comparison of results between business applications
Simulation	Artificial model validation based on artificial input scores and application settings (support validation beyond one company context and over many applications)
Automation	Interfaces to bank internal and external systems for data capture (internal, external) • Demand management systems inhouse • Source code repositories inhouse • OSS scanning systems (Blackduck) • Maven Central repository • OSS component source code repositories (e.g. GitHub)