SWEN 301 : Scalable Software Development

The Problem with Software

Jens Dietrich **(jens.dietrich@vuw.ac.nz)**

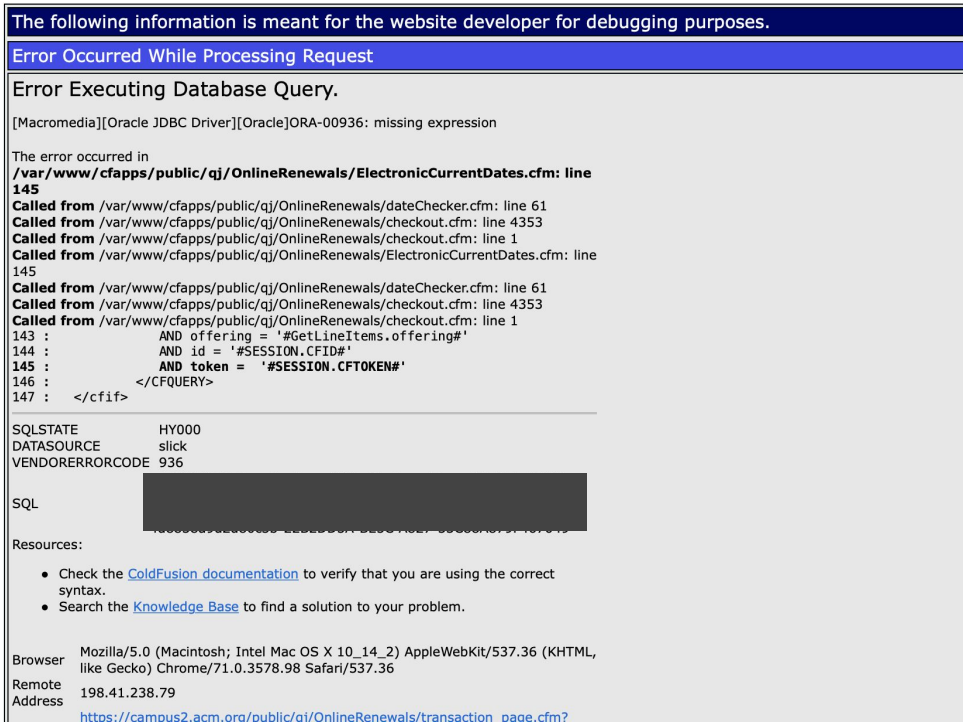
About Jens

- MSc / PhD Math / CS from University of Leipzig
- (East) German – thick accent you need to get used to (but Werner Herzog's is worse)
- consulting Work (Germany, Switzerland, UK, Namibia), then NZ Academia (first Massey, than VUW since 2018)
- research on Software Quality / Security using Automated Program Analysis, works closely with Oracle Labs (Oracle is the company behind Java)
- impact (follow links for examples):
  - vulnerabilities (CVEs and GHSA entries)
  - changes in Spring and Google Guava
  - bug fixes in mainstream static analysis algorithm
  - ...

Software Quality is Deplorable

ACM Membership Renewal, 14 Feb 19

The web site you are accessing has experienced an unexpected error.
Please contact the website administrator.



The following information is meant for the website developer for debugging purposes.

Error Occurred While Processing Request

Error Executing Database Query.

[Macromedia][Oracle JDBC Driver][Oracle]ORA-00936: missing expression

The error occurred in
/var/www/cfapps/public/qj/OnlineRenewals/ElectronicCurrentDates.cfm: line 145
Called from /var/www/cfapps/public/qj/OnlineRenewals/dateChecker.cfm: line 61
Called from /var/www/cfapps/public/qj/OnlineRenewals/checkout.cfm: line 4353
Called from /var/www/cfapps/public/qj/OnlineRenewals/checkout.cfm: line 1
Called from /var/www/cfapps/public/qj/OnlineRenewals/ElectronicCurrentDates.cfm: line 145
Called from /var/www/cfapps/public/qj/OnlineRenewals/dateChecker.cfm: line 61
Called from /var/www/cfapps/public/qj/OnlineRenewals/checkout.cfm: line 4353
Called from /var/www/cfapps/public/qj/OnlineRenewals/checkout.cfm: line 1

```
143 :               AND offering = '#GetLineItems.offering#'
144 :               AND id = '#SESSION.CFID#'
145 :               AND token =  '#SESSION.CFTOKEN#'
146 :             </CFQUERY>
147 :    </cfif>
```

SQLSTATE	HY000
DATASOURCE	slick
VENDORERRORCODE	936
SQL	[redacted]

Resources:

- Check the ColdFusion documentation to verify that you are using the correct syntax.
- Search the Knowledge Base to find a solution to your problem.

Browser	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36
Remote Address	198.41.238.79

https://campus2.acm.org/public/qj/OnlineRenewals/transaction_page.cfm?

Access to VUW Course Outline Editor 4 July 24

```
org.springframework.security.authentication.AuthenticationServiceException: Error validating SAML message
	at org.springframework.security.saml.SAMLAuthenticationProvider.authenticate(SAMLAuthenticationProvider.java:95)
	at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:156)
	at org.springframework.security.saml.SAMLProcessingFilter.attemptAuthentication(SAMLProcessingFilter.java:87)
	at
org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter.doFilter(AbstractAuthenticationProcessingFilt
er.java:195)
	at org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:342)
	at org.springframework.security.web.FilterChainProxy.doFilterInternal(FilterChainProxy.java:192)
	at org.springframework.security.web.FilterChainProxy.doFilter(FilterChainProxy.java:166)
	at org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:342)
	at
org.springframework.security.web.context.SecurityContextPersistenceFilter.doFilter(SecurityContextPersistenceFilter.java:87)
	at org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:342)
	at org.springframework.security.web.FilterChainProxy.doFilterInternal(FilterChainProxy.java:192)
	at org.springframework.security.web.FilterChainProxy.doFilter(FilterChainProxy.java:160)
	at org.springframework.web.filter.DelegatingFilterProxy.invokeDelegate(DelegatingFilterProxy.java:346)
	at org.springframework.web.filter.DelegatingFilterProxy.doFilter(DelegatingFilterProxy.java:259)
	at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:78)
	at weblogic.servlet.internal.RequestEventsFilter.doFilter(RequestEventsFilter.java:32)
	at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:78)
	'''
```

Make sure to hide error content from your production environments to minimize leakage of useful data to potential attackers.

Passports, licenses of 300 leaked in Ministry for Culture and Heritage data breach (2019)

Ministry chief executive Bernadette Cavanagh says the personal documents were compromised following a “coding error”.

“Good coding reviews and more complete acceptance testing will lead to the reduced probability of leaving a door wide open for malicious parties to exploit.”

https://securitybrief.co.nz/story/passports-licenses-of-300-leaked-in-ministry-for-culture-and-heritage-data-breach

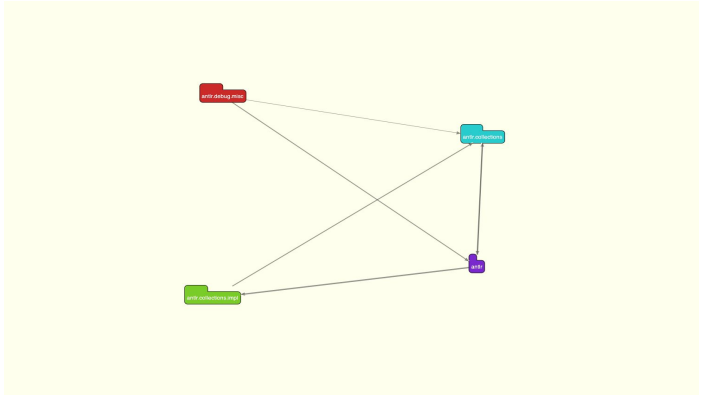
Bigger Issues

- Novapay (NZ, ca 50 mill NZD damage)
- glitches in airplanes, including flight control system of Boeing 737 MAX
  - social cost: crashes of Lion Air Flight 610 and Ethiopian Airlines Flight 302, with 346 fatalities
  - economic cost: Boeing settled to pay over $2.5 billion: a criminal monetary penalty of $243.6 million, $1.77 billion of damages to airline customers, and a $500 million crash-victim beneficiaries fund [source: https://en.wikipedia.org/wiki/Boeing_737_MAX]
  - .. and also: dreamliner (787), AirBus A400, and rockets (Ariane 5, ca 370 m EUR)

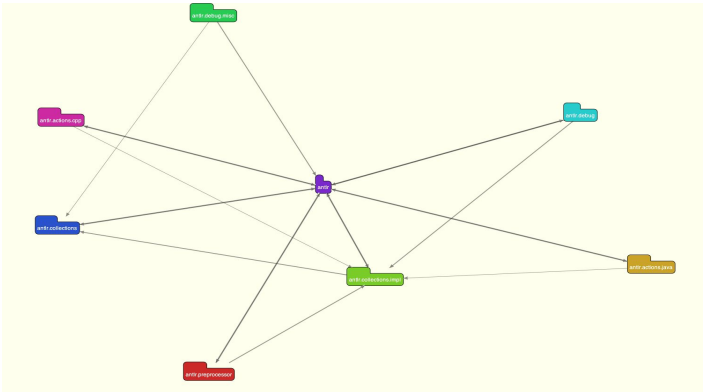
What is the Problem ?

- no simple answer
- a combination of technical and social aspects, often not completely understood
- we will focus mainly on the **technical reasons** that contribute to project failures and poor software quality

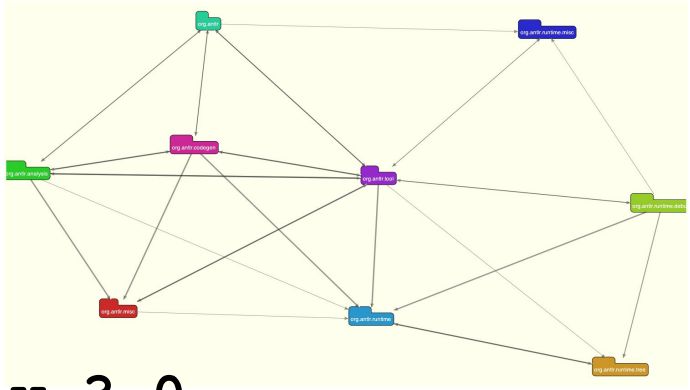
ANTLR Packages and their Dependencies



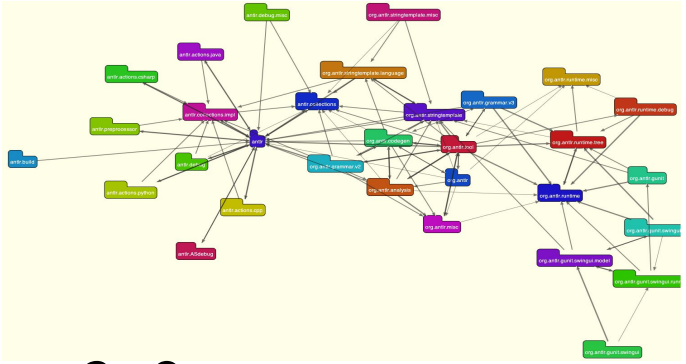
v 2.7.0



v 2.7.2

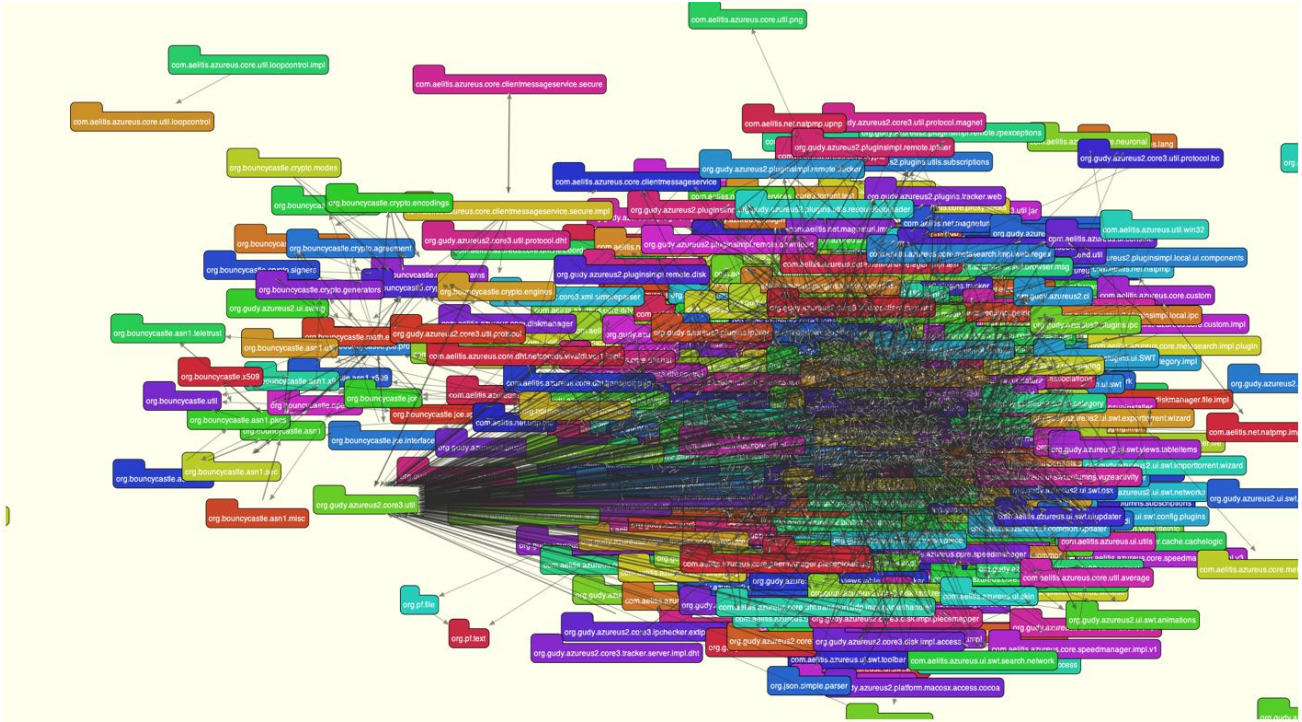


v 3.0



v 3.2

Azureus 4.5.0.4 - Packages and their Dependencies

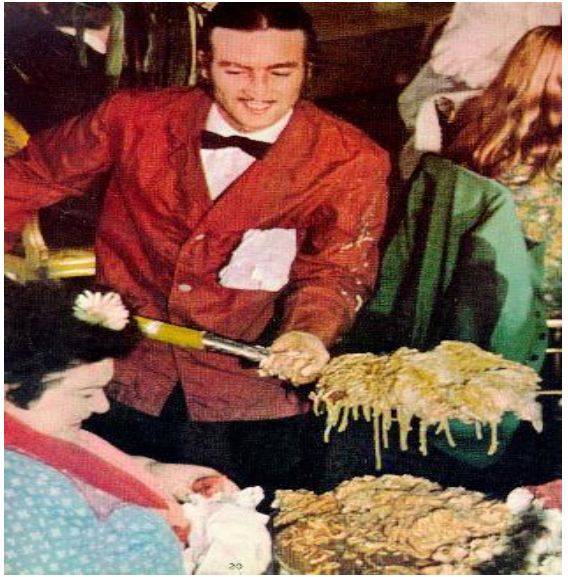


481 packages, 4391 relationships

Big Balls of Mud (B. Foote, J. Yoder 99)

A BIG BALL OF MUD is .. sprawling, sloppy, DuctTape and bailing wire, **SpaghettiCode jungle**. ..

The overall structure of the system may never have been well defined. If it was, it may have **eroded beyond recognition**.



The Winchester Mystery House

before the 1906 quake

today

The Winchester Mystery House

- Victorian mansion in California, built 1884 - 1922
- cost apprx 5.5 million USD
- seven stories high, four remaining (after 1906 quake)
- there are roughly 160 rooms, including 40 bedrooms, 2 ballrooms (one completed and one unfinished) as well as 47 fireplaces, 10,000 window panes, **17 chimneys (with evidence of two others)**, two basements and three elevators.
- **a metaphor for complex, evolved design and quality problems resulting from this**

source: http://en.wikipedia.org/wiki/Winchester_Mystery_House

Doors to Nowhere

The Quest for Simplicity

Constructed 2025

Taming Complexity with Divide-and-Conquer

Components !

(modules, parts, libraries, services ..)

Components to the Rescue

We undoubtedly **produce software by backward techniques**. We undoubtedly get the short end of the stick in confrontations with hardware people because **they are the industrialists and we are the crofters**. Software production today appears in the scale of industrialization somewhere below the more backward construction industries.

McIlroy, M. Douglas. "Mass produced software components." (1969): *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7–11 Oct.* ***1968***. Scientific Affairs Division, NATO.
https://www.cs.dartmouth.edu/~doug/components.txt

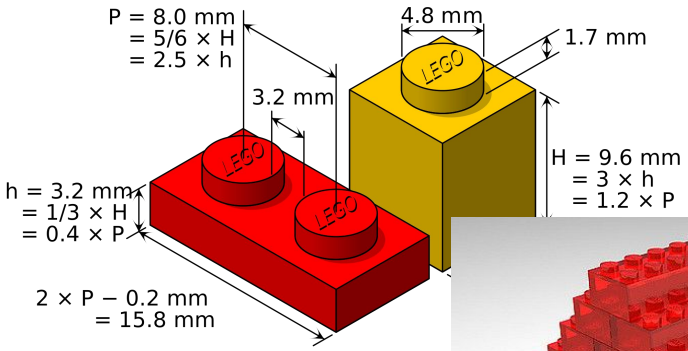
Components to the Rescue (ctd)

.. certain ideas from industrial technique I claim are relevant. The idea of **subassemblies** carries over directly and is well exploited. The idea of **interchangeable parts** corresponds roughly to our term `modularity,' and is fitfully respected.

My thesis is that the software industry is weakly founded, and that one aspect of this weakness is the absence of a software components subindustry.

McIlroy, M. Douglas. "Mass produced software components." (1969): *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7–11 Oct. 1968*. Scientific Affairs Division, NATO.
https://www.cs.dartmouth.edu/~doug/components.txt

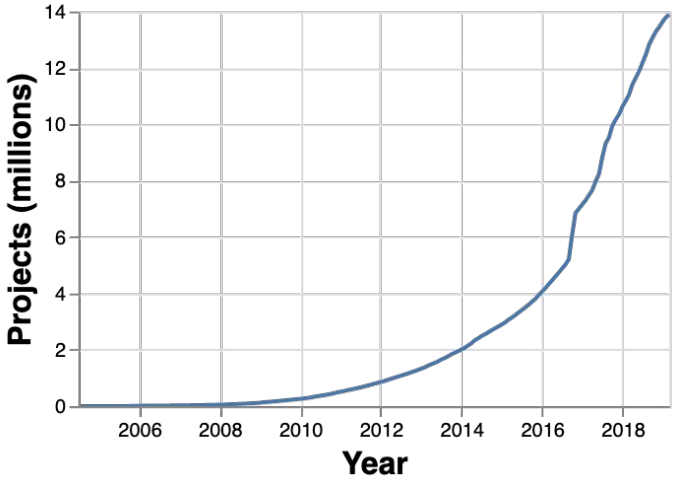
Components ⇒ Simplicity ?



Advances and Trends

- most programming languages have some kind of component / package model
  - NPM (JavaScript), Maven (Java), Gems (Ruby)
  - see https://libraries.io/ for a “meta repository”
- there are PL independent, network-centric models as well: CORBA, Protocol Buffers, and service-oriented approaches (SOAP, REST)
- some PLs promote **substitutability** of components implementing the same specification to develop **ecosystems** with competition promoting fitness for purpose
  - example: JDBC + SQL standards for pluggable database support
- some PLs support “hot swapping” of components to avoid recompilation/redeployment/restart
  - example: OSGi

The Maven Central Repository
https://mvnrepository.com/



NPM (package for JS): passed 2.1 million packages in 2022

https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager

Latest Maven Central Stats
https://search.maven.org/stats

Date index last refreshed: 2023-07-11 14:07:01 UTC

Total number of artifacts indexed (GAV): **11,598,360**

Total number of unique artifacts indexed (GA): **553,690**

Unsolved Problems: Substitutability

- replace one component (service, subsystem, module, package) by another
- most common scenario: upgrade a component - replace one version of a component by a new version of this component – aka **hot upgrade**
- necessary due to change pressure
- number of dependencies is large -- Maven projects have around 5 direct dependencies, and NPM projects around 8 direct dependencies in average *, with many more transitive dependencies

* Dietrich et al: Dependency Versioning in the Wild. MSR'19

Unsolved Problem: Scale and Granularity

- use components at the optimal level of granularity
- it is possible to get it wrong either way: over-engineering simple application vs under-engineering complex ones

New Problems: Spreading Bugs and Vulnerabilities

- very fine granularity of package re-use, leading to issues
  - leftpad 2016 incident (have we forgotten how to program?): package withdrawal lead to widespread build failure
- faults amplified by heavy reuse
- OWASP Top 10 (security threat list) now contains Using Components with Known Vulnerabilities
- software supply chain attacks: compromised builds, typo squatting, ..
- example:
  - log4j vulnerabilities in Dec 21
  - most recent (2024): xz, polyfill

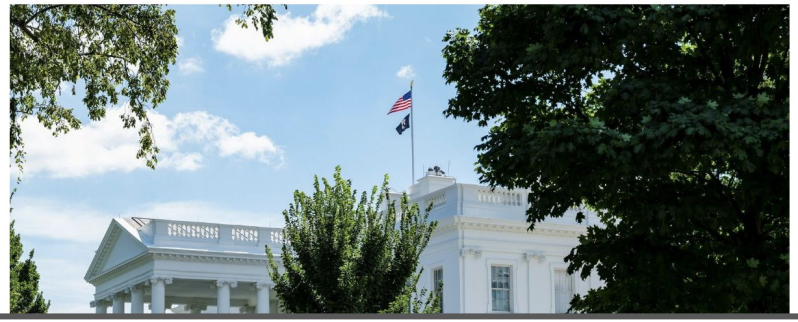
WSJ PRO

White House Convenes Open-Source Security Summit Amid Log4j Risks

Widespread use of open-source technologies and their maintenance by small groups creates national-security concerns, officials say

By James Rundle

Updated Jan. 13, 2022 8:09 pm ET | WSJ PRO



New Problems: DLL Hell & Co

- project A depends on B, B depends on C etc
- dependencies quickly form complex trees or networks
- how to deal with situations where A (indirectly) depends on multiple versions of C ?
- for instance, C might have added / removed methods from a class used by A
- classic problem known as DLL hell (jar/classpath hell), requires complex engineering to solve

Compatibility can be Subtle (Compiletime vs Runtime)

lib version 1.0

```
import java.util.*;
public class Foo {
      public static Collection getColl() {
            return new ArrayList();
      }
}
```

lib version 2.0

```
import java.util.*;
public class Foo {
      public static List getColl() {
            return new ArrayList();
      }
}
```

client program using lib

```
import java.util.*;
public class Main {
  public static void main(String[] args) {
    Collection coll = Foo.getColl();
    System.out.println(coll);
  }
}
```

```
java -cp lib-1.0.jar Main
java -cp lib-2.0.jar Main
```

Components are not Enough (to tackle Complexity)

What is Complexity ?

- lack of structure that decreases **comprehensibility**
- too much interconnectivity between parts that does not follow pattern
- facilitates **ripple effects**, co-change, make change difficult as it creates unexpected side-effects, results on **poor maintainability**
- some quantification is possible (complexity metrics)
- relationship with size?

Aspects of Complexity

multiplicity (size) -- the number of potentially interacting elements

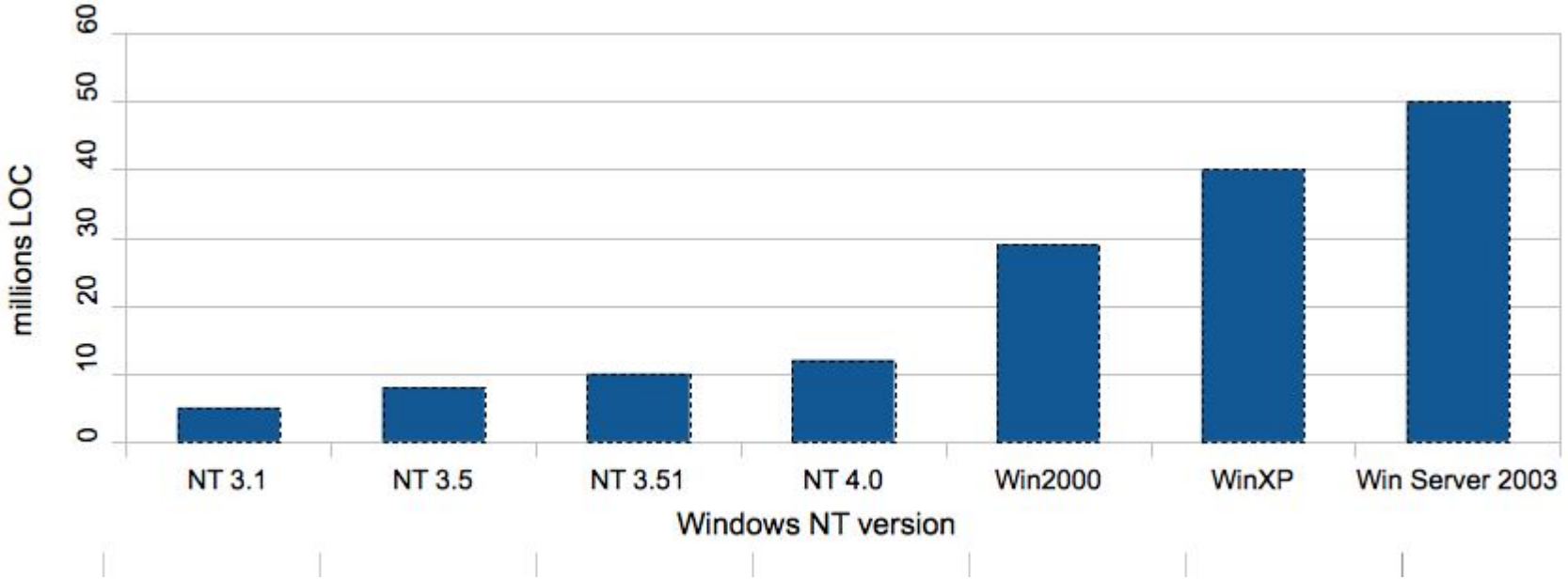
interdependence -- interconnectivity of those elements are

diversity -- has to do with the degree of their heterogeneity

Size

https://www.visualcapitalist.com/millions-lines-of-code/

Size



source: V. Maraia: The Build Master: Microsoft's Software Configuration Management Best Practices. Addison-Wesley 2005.

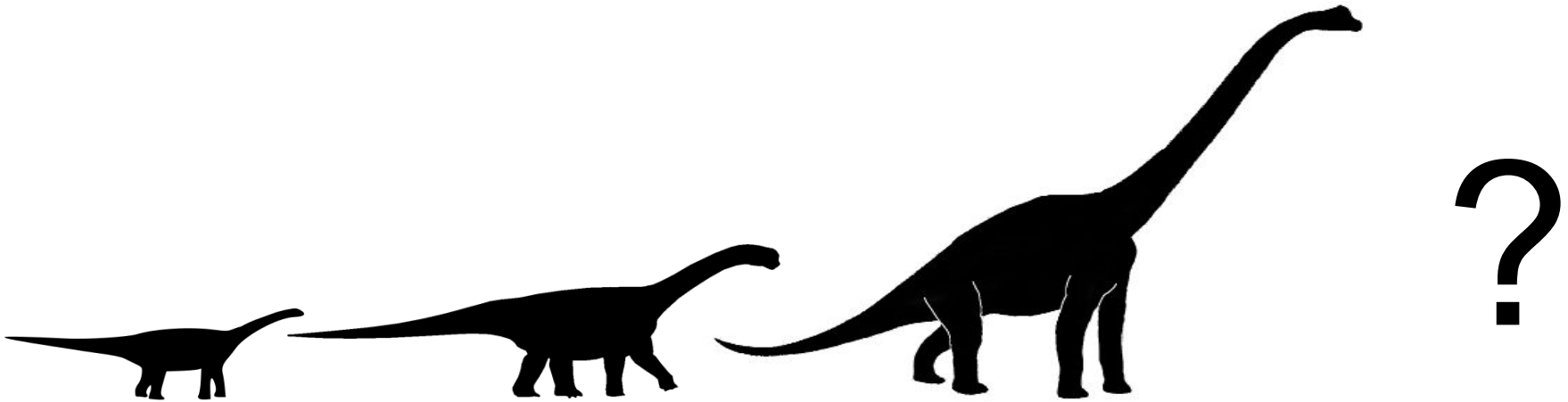
Code Size ≈ Team Size

Ringelmann Effect

The Ringelmann effect is the tendency for individual members of a group to become increasingly less productive as the size of their group increases.

https://en.wikipedia.org/wiki/Ringelmann_effect

Evolution and Size



Complexity vs Size

- in general: complexity != size
- but often both are correlated
- more structure is introduced to cope with larger systems

Complexity Emerges (Out of Nowhere)

- complex systems **emerge from simple elements** and their interactions
- Goldstein: Emergence is "the arising of novel and coherent structures, patterns and properties during the process of self-organization in complex systems" http://goo.gl/Qfr86
- emerging complexity can be observed in software systems, and new concepts are created and studied to describe this complexity
- for an example of emerging complexity, check out Conway's game of life

Lehmann's Laws

1. The law of **continuing change**. Any software system used in the real-world **must change** or become less and less useful in that environment.
2. The law of **increasing complexity**. As time flows forwards, **entropy increases**. That is, as a program evolves, its structure will **become more complex**.

source: Lehman, M.: Programs, life cycles and the laws of software evolution," *Proc. IEEE*, 15 (3), 1980.

Why Change, anyway ?

The World is Complex !

(and it sometimes takes time to understand this)

Encore: Why Denmark Is .07 Seconds Behind The World

The World (and our Understanding of it) is Changing

- **example: Y2K**
- caused by use of the Gregorian Calendar in the late 90ties
- problem faced by computers to roll over 2 digit dates to 4 digits dates
- old software had **incorrect built-in assumptions** over the lifetime of the system
- estimated cost to fix this: "Worldwide, organizations were estimated to have spent **$308 billion** before the millennium on remediation efforts." source: Computerword.
- good news: a Y2K movie was made

Feature Creep

- MS Word 2003 has **292** menu items that needed to be mapped to Word 2007 (source: http://goo.gl/abNC2)
- this is a sign of **feature creep** - the inclusion of non-essential feature (aka Creeping Featuritis)
- programmers **confuse what is needed and what is neat**
- this is sometimes also called "everything but the kitchen sink" syndrome
- reasons:
  - programmers with ego
  - marketing departments
  - (perceived) competitive advantage: based on the idea that value = number of features

Feature Creep

VS

Fighting Back: YAGNI & KISS

- **YAGNI** - you are not going to need it - always implement things when you **actually** need them, never when you just **foresee** that you need them
- **KISS** - keep it simple and straightforward
- agile principles aimed to retain simplicity
- consequences:
  - features should be traceable to (customer-sourced) requirements !
  - this **requires that requirements are documented and managed**, e.g. via use cases

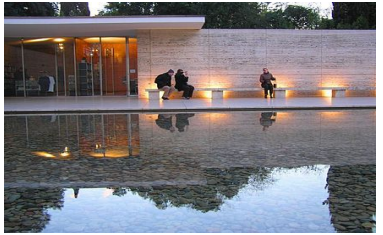
The Quest for Simplicity

- design driven by actual requirements
- the simplest design is the best
- use proven designs on all levels (reuse)
- simplify as part of regular maintenance (“refactoring Friday”)
  - simplicity as technical / organisational goal
  - simplicity often correlated to maintainability and therefore TCO
- careful modularise on all levels
- to control complexity, we need to measure it

The Quest for Simplicity Elsewhere

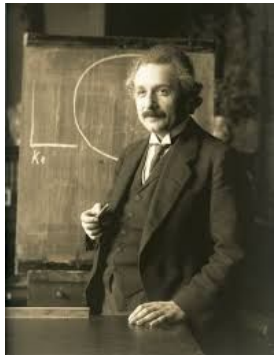
Philosophy: “Occam’ razor” - William of Ockham (ca 1300):
Entia non sunt multiplicanda praeter necessitatem:
More things should not be used than are necessary.
https://simple.wikipedia.org/wiki/Occam%27s_razor

Architecture: “Less is More” - Ludwig Mies van der Rohe (1886–1969) (Bauhaus Architecture)
https://en.wikipedia.org/wiki/Minimalism#Less_is_more_(architecture)



.. ctd

Science: “Everything should be made as **simple** as possible, but not simpler.” Albert Einstein



Netflix: Tidying Up with Marie Kondo, 2019.
https://www.netflix.com/title/80209379

The Agile Manifesto

Simplicity -- the art of maximizing the amount of work **not done** -- is essential.

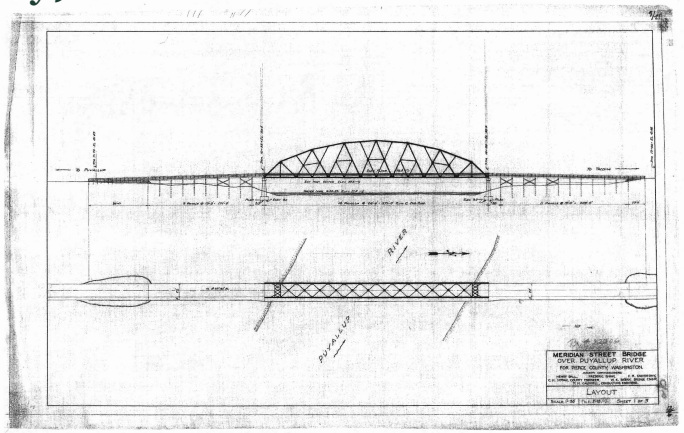
https://agilemanifesto.org/principles.html

Predictable vs Agile

- engineering projects have two conflicting goals:
- **predictability** is desirable for **planning and controlling**, in particular resource allocation
- **agility** is desirable to **facilitate change** when requirements, or the understanding of requirements, changes
- the right **balance** depends on the nature of project

Building Bridges

- traditional (civil) engineering is based on the **assumption** that **requirements can be understood early** with a high level of completeness and correctness
- this enables a “waterfall” style methodology: a structured **linear process** (analysis, design,construction, testing and delivery)
- backtracking (redesigning) is expensive



Backtracking is Expensive

https://edition.cnn.com/2019/10/01/asia/taiwan-bridge-collapse-intl-hnk-scli/index.html

Backtracking is Expensive

incorrect requirements

https://www.theguardian.com/business/2019/feb/14/a380-airbus-to-end-production-of-superjumbo

development costs USD 25bn https://en.wikipedia.org/wiki/Airbus_A380

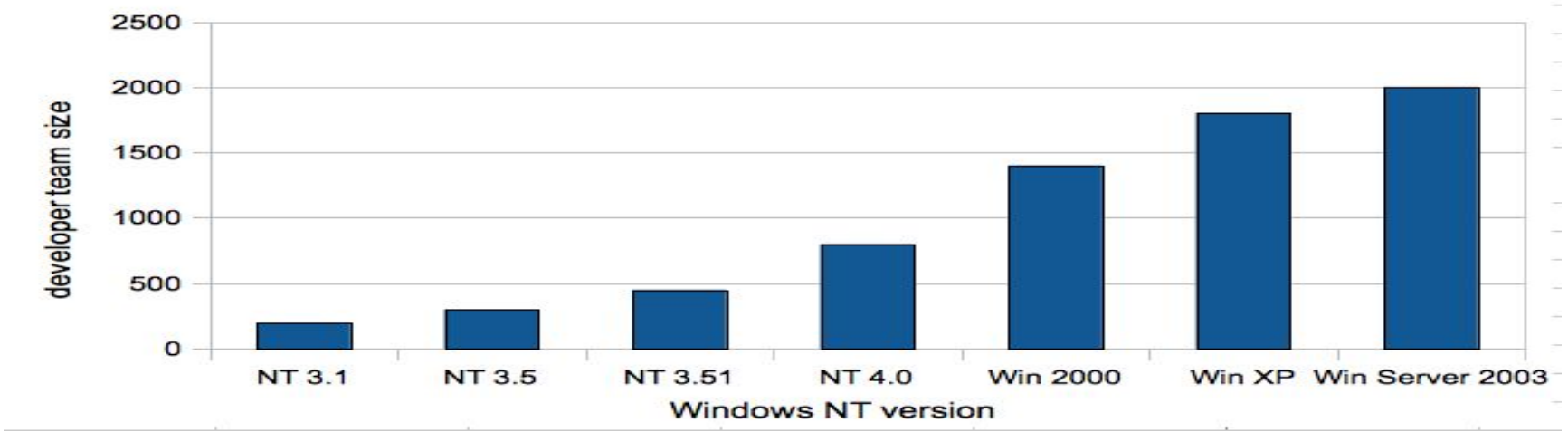
Software vs Civil Engineering

- assumption that requirements are stable and are understood early is not valid
- backtracking easier and can even be partially automated
- more iterative processes are possible: requirement changes do not necessarily derail projects
- but how much is lost by not taking advantages of “synergy effects” early ?

Agile Software Engineering: Background

- rapid changes starting in the late 80 ties: globalisation, liberalisation of industries (banking, telecom, utilities)
- changes for businesses:
  - more competition in deregulated industries (utilities, telecoms)
  - globalisation impact: exchange rates, i18n, time zones, outsourcing
  - regularly changes (Basel Accords from 1988, Sarbanes-Oxley US 2002, ..) to curb banking
- significant increases in the size and complexity of software
- large detached teams (exploding team sizes & outsourcing)
- software projects became a game with moving goalposts

Increasing Team Sizes



- the size of Windows NT developer teams from 1993-2003
- source: V. Maraia: The Build Master: Microsoft's Software Configuration Management Best Practices. Addison-Wesley 2005.

The Agile Manifesto

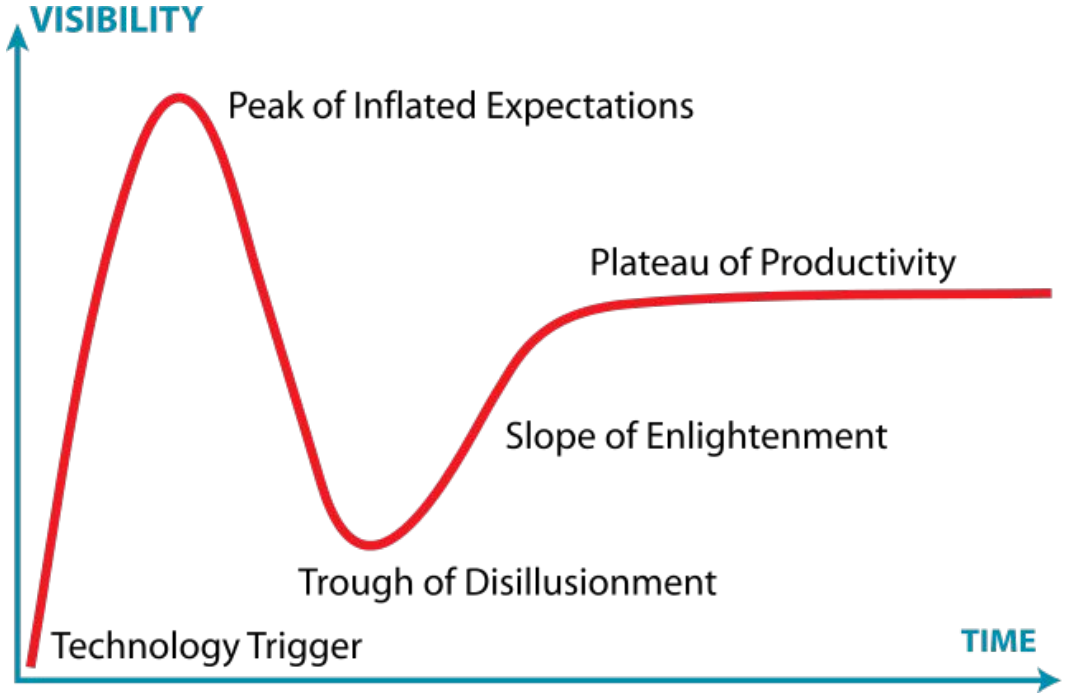
- principles for flexible methodologies to facilitate responding to change
- focus on:
  - short iterations with clear deliverables (code with features) at the end of each iteration, each focusing on a subset of requirements
  - strong communication to ensure requirements are understood
  - minimalistic design
- consequences: no big upfront design, instead, architecture is emerging from self-organising teams

https://agilemanifesto.org/principles.html

Adaptation of Agile

- hype in the early 2000s , with many methodologies: extreme programming, scrum , kanban , ..
- large consulting industry developing to make corporations agile
- followed by some disillusionment (“agile means fragile”)
- 2019 .. adaptation of selected practices, often blended with some predictive elements (e.g. high-level architecture)

The Gartner Hype Cycle



https://en.wikipedia.org/wiki/Hype_cycle

Amara’s Law

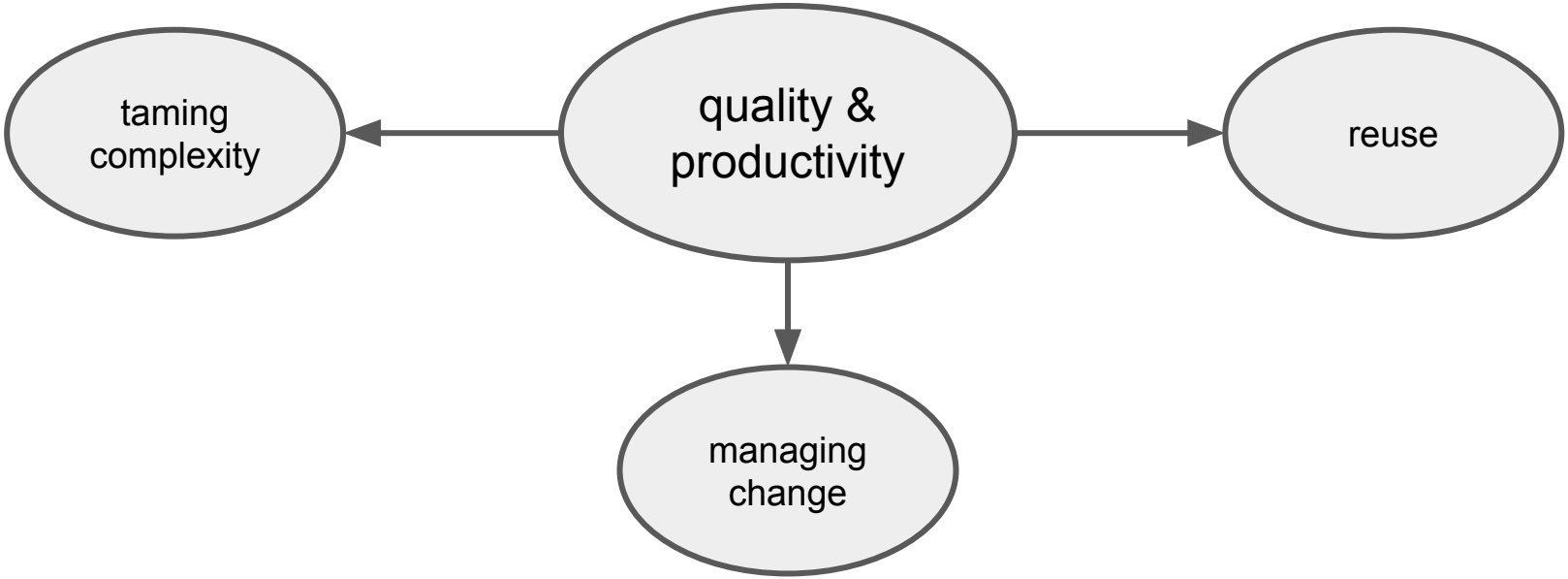
We tend to overestimate the effect of a technology in the short run and underestimate the effect in the long run.

Agile beyond the Hype

- since 2010: widespread adaptation of agile principle
- not always clear whether everything labelled agile is consistent with original intentions
- pure agile methodologies often more suitable for inhouse and longrunning projects
- less suitable for fixed-price projects
- some upfront design beneficial
- in practice, most projects benefit from **a combination of predictive and agile** approaches

Topic Overview

How to write and maintain good quality software of any size efficiently.



Topics: Managing Change

- build automation
- continuous integration and delivery
- refactoring
- reverse engineering design and architecture
- program analysis
- diagnostics (debugging, profiling, logging, monitoring)
- evolution and compatibility

Topics: Taming Complexity

- architecture
- modularity, components and services
- test-driven development
- refactoring
- reverse engineering design and architecture
- convention over configuration

Topics: Reuse

- modularity, components and services
- frameworks
- API evolution and versioning, compatibility
- convention over configuration

Software quality is deplorable.

bonus: https://www.youtube.com/watch?v=YRCzEqkCoiM

https://github.com/jensdietrich/se-teaching