Obsolescence
The part of IT you don’t care for

June 2015

Darius Blasband – darius@raincode.com
The company
Based in Brussels, Belgium
- Founded in 1998, 33 people
- Privately owned,
- Profitable from day one
- Strong relationship with Microsoft
- RainCode USA incorporated in April 2012
- To better serve our biggest market (> 55%)

We do compilers and languages
- Compilation, migrations and quality control
- PACBASE, CA-DATACOM and COOL:Gen as major service activities
- But also code generation for DSLs and model-based development (Mastercard...)

An evolving business model...
- From indirect sales through partnerships...
- .. to an increasing share of direct sales

End user references:
- DVV Insurance (Brussels), PWGSCA, Sun Life Financial, GE Aviation (USA), Mastercard (Belgium), Wells Fargo (USA), UBS Warburg (Zurich), CRI Trade (USA), SAIC (USA), ALD Automotive (France), Société Générale (France), San Mateo County (USA), NRI (Japan), Alstom (World wide), Sopra (World wide), Volvo-IT (France), Microsoft (World wide), MAAF (France), Sage group (France), MBank (Poland), PriceWaterhouse Coopers (Netherlands), SDC (Denmark), Elektron (Faroe Islands), mBank (Poland), Pole Emploi (Nantes, France), Natixis (France)...
Looking for bright people for our technical staff in Brussels... and elsewhere
Worth remembering...

Owning a software system costs money
- Maintenance
- Know-how
- Liability

This cost increases with magnitude
- Goes without saying

...and with technological dependencies
- Correlated with time
- Not felt before timescales hit the first decade
- Fragility

In comparison....
- Development costs are negligible
- Optimizing development productivity is often a misguided strategy
- As it usually implies more numerous and dubious dependencies
An inconvenient negative correlation

![Graph showing a negative correlation between Lifetime and Abstraction]
The cost of an IT system, along two axis

Magnitude

Avoiding technical debt as sole motivation

Common evolution

Expressivity as sole motivation

Dependencies
If you were to develop a system which must be available for 30 years to come, which language(s) would you use?

(You’re not gonna like this)
Debunking the Java myth

- Will java still be around in 20 years?
  - Probably

- Can a Java application survive 20 years easily?
  - Definitely not

Because it lives in an ecosystems of components
- Struts, Spring, Ajax, Eclipse (Swt), Jboss, XML parser, Hibernate, etc...
- Which all evolve independently
- Kind of a super legacy with distributed responsibilities

A Java application goes through a series of technical migrations, to update components, deal with conflicts,
- Fragile

.NET, for all its flaws, does not suffer from this
- Or not to the same degree
Debunking myths: how old is old?

- Before worrying about obsolescence...
  - What kind of timeframe are we talking about?
- How long does a typical application live?
  - Before falling into oblivion?

- Depending on the domain?
  - Industrial?
  - Banking?
  - Insurance?
- We essentially don’t know
  - For banking and insurance the median/average lifetime may very well turn around 100 or 150 years
  - But we’ll have to wait before we’re sure
  - At least 100 or 150 years 😊
- For all practical purpose:
  - Undetermined in the future
There is such a thing as too much, too old software
- When cost exceeds value

Shows in multiple areas
- Balance sheet
- Head count
- Flexibility
Even less of a platitude

Rewriting a software system...
- Fails more often than not
- Counter intuitive
- But experimentally sound

Replacing it by a package is no picnic either
- More on this later...

System decommissioning is a nightmare
- Untapped business opportunity
Debunking myths: how about redevelopment?

- Why make such a drama of it?
  - Let’s just redevelop everything every 10-15 years
  - Time is no longer an issue!
  - Intuitively reasonable

- But…

- Factor 1: volume
  - These systems took 20+ years to develop
  - Redeveloping them in 3 to 5 years is insanely optimistic
  - And already way too long
  - Because productivity has not improved significantly

- Factor 2: price
  - Guess what: money is always an issue

- Factor 3: continuity
  - The application must keep evolving during redevelopment
  - Moving target
  - Data
  - Integration with external systems
  - Evolving at their own pace

- All in all:
  - Redevelopment projects seem to fare even worse than plain developments from scratch
- **Groupe SNCB:**
  - 400 million euros on a SAP implementation
  - 6,000 users
  - One Mercedes Sedan per user (Leather seats, onboard computer, bar, etc.)
  - 40 euros for every Belgian
    - Babies, infants and crippled
  - And this does not even start contributing to running trains
    - Just organizational overhead
  - And it is not over yet
    - Maintenance costs
    - Evolving needs
    - A SAP-fueled legacy

- Still, presented as a success!
  - By SAP themselves!
  - Admittely, could have been even worse
  - But the how rational is it?
  - Cost vs. value
Application-level dependencies

- Poorly mastered issue
  - Sold by salespeople with quarterly quotas
  - And managed for 20+ years by those naïve enough to listen
- An extreme form of dependence
  - Where the organization bends rather than let the software adjust
  - Negating competitive advantage
- More violence than one may think
  - No one is too big to fail
  - Nor SAP, IBM, Oracle, Google
  - On a 20 years horizon, anyone can die (Compaq, Wang, Sun, Informix, Digital, Lotus)

- Not limited to enterprise-level systems
  - SaaS applications only speed up the process
  - Google’s graveyard (Google wave?)
  - Mobile apps turbocharging it
- This is only the beginning!
  - Twitter, Facebook, MySpace, Salesforce, Gmail, are uncontrolled liabilities
Organizations can die under the burden of their software
• Soaring costs and diminishing value

No such thing as too big to fail
• Extremistan and mediocristan
(Nassim Nicholas Taleb)
A gloomy picture...

- It is bad
- It is about to get worse
- Replacing by packages fail often
- Do does application re-development
- Is there any hope at all?
Legacy modernization

- Technical migrations
  - In the sense of dealing with technical dependencies
- Better than nothing
- Ungluing applications from ageing infrastructures
  - Programming languages
  - Databases
  - Hardware
  - TP monitor
- Divided in a large number of niche markets
- Two enemies
  - Entropy
  - And dependencies...
  - ...whether one replace a dependency by another...
  - ...or whether one addresses a product dependency but ignores a skill dependency
Raincode’s business, along two axis

- **Transformational**
  - COBOL to COBOL
    - Behaviour-preserving refactorings
    - Abstract interpretation (but not on an idealized language!)
    - Clone detection
    - Used as a vector for 4GL migration
  - CA-DATACOM
    - Call-level to embedded SQL
    - Also based on abstract interpretation
  - Misc
    - Progress to Java
    - Pascal to C
    - Flat files to relational
    - Etc.

- **Emulation**
  - The exact opposite of transformational
    - Keep the system as it is
    - Idiosyncrasies and all
  - Compilers
    - COBOL and PL/I
    - The “legacy compiler” concept
    - Soon: ASM370 compiler (!!!)
  - Emulators
    - CICS
    - IMS
    - CA-DATACOM
  - Reproduce complex behaviours
    - Concurrency and locking
    - Bugs!
  - A balancing act
Technical migration

- Magnitude
  - Entropic case
  - Best case
  - Worst case

- Dependencies
Java as a migration target

- The idiom issue must be addressed
- Short of which the migration will be pointless

```java
public void f90Ao00W_M640 () {
    wAo00.fill(" ");
    wAo00.setValue(aoFile.getAo00());
    udFile.setUd00Undkey(aoFile.getAo00Undkey());
    udFile.setUd00Recnum(1);
    if (! (aoFile.getAo00Rechdr() > 450)) {
        udFile.setUd00Rechdr(aoFile.getAo00Rechdr());
    } else {
        udFile.setUd00Rechdr(450);
    }
    udFile.insert();
    udFile.setUd00Rechdr(udFile.getUd00Rechdr() + 15);
    udFile.setUd00Half(wAo00.getWAo00Half1());
    wssBegin.setIk("0");
    BaseService fbBS = ServicesFactory.create("com.whoever.fb", this);
    fbBS.addCallParameter(17, false);
    fbBS.addCallParameter(fbActions.getFieldfbProgram(), false);
    fbBS.addCallParameter(fbActions.getFieldfbWrite(), false);
    fbBS.addCallParameter(fbActions.getFieldNullInfosStatus(), false);
    fbBS.execute();
    if (statusArea.getFieldOneUd00Status().isGreaterThan("02")
        && !statusArea.getFieldOneUd00Status().equals("22")
        && !statusArea.getFieldOneUd00Status().equals("23")
        && w_perform("secud_M720")
        && statusArea.getFieldOneUd00Status().equals("22")
        || statusArea.getFieldOneUd00Status().equals("23")
        || wssBegin.setIk("1")
        || udFile.insert();
    if (!wssBegin.getFieldIk().isFilled('0')
        && aoFile.getAo00Rechdr() > 450)
        udFile.setUd00Undkey(aoFile.getAo00Undkey());
```
Things can get so much worse: VistA and MUMPS

- **VistA** is health-care management system
  - Developed by the Veteran Affairs in the States
  - Open source
  - A monster:
    - 160 functional modules
    - 963 hospitals and clinics
  - An entire ecosystem
  - Potentially impacts up to 25% of the US population

- **Technical, public and political success**

- But this success now turns to a huge liability

- VistA is developed using MUMPS
  - Non-relational data store
  - Transparent persistence abstraction
  - No other abstraction
  - A disaster in the making

- For several years now:
  - The VA invests 10-15 million dollars a year in studies alone
  - A total figure of 500 million is often cited
  - To no avail
Obsolescence is like caring for the environment

It is not always fun
But it is here to stay
Get used to it!

Our future will be about dealing with the past
Thank You!