

goto; conference

Magazine vol.2, issue 2, 2012

www.gotocon.com



Analysis for Continuous Delivery: Five Core Practices

Memory Access Patterns Are Important

The Systemico Model

Databases & Psychology

A Requested Red Flag For The Internet



con

goto magazine 2012

Dear GOTO Community,

GOTO Aarhus 2012 is only a few weeks away and we would like to prepare you for the conference by providing you with the third version of the GOTO Conference Magazine. We are happy to once again provide a variety of articles and online interviews from some of the speakers from the GOTO community.

In this edition, we have articles from Jez Humble who dives into an analysis for Continuous Delivery, Rick Falkvinge who will be keynoting at GOTO Aarhus this year, provides his views on the status of internet, Martin Thompson goes into detail around Memory Access Patterns and why they are important, Stefan Edlich dives into the psychology of databases, and lastly Barry O'Reilly discusses use cases with the Systemico Model.

Also make sure to take a look at the video interviews we have online as they provide candid discussions where speakers and practitioners talk about some of the challenges and ideas behind the projects that they are working on.

We hope you enjoy this edition of the GOTO Magazine and looking forward to seeing you at GOTO Aarhus 2012.

tents;

& contributors

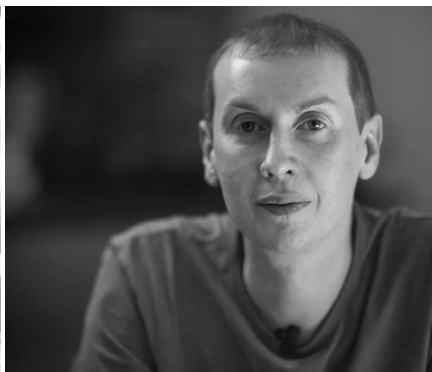
4. Analysis for Continuous Delivery: Five Core Practices

Jez Humble



10. Memory Access Patterns Are Important

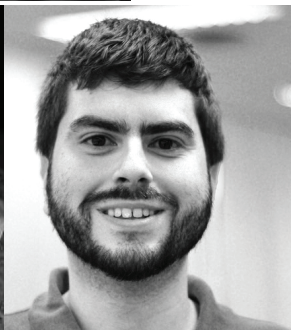
Martin Thompson



20. Conversations with GOTO speakers

22. The Systemico Model

Barry O'Reilly



Lourenco Soares

28. Databases & Psychology

Stefan Edlich



32. A Requested Red Flag For The Internet

Rick Falkvinge

Analysis for Continuous Delivery:

Five Core Practices



Jez Humble
Principal Consultant, ThoughtWorks



Jez Humble, coauthor of [Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation](#), urges teams to move away from the all-or-nothing design of traditional software delivery approaches. Following the practices outlined here, you can deliver single-feature or small-story batches that dramatically decrease the time needed to build a new product or new release, testing and moving forward on successful features and redesigning or dropping features that fail (or that users show they don't really want)

Continuous delivery is a software development strategy that optimizes your delivery process to get high-quality, valuable software delivered as quickly as possible. This approach allows you to validate your business hypothesis fast and then rapidly iterate by testing new ideas on users. Although *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation* focuses on engineering practices, the concept of continuous delivery has implications for the whole product-delivery process, including the “fuzzy front end” and the design and analysis of features.

Here's the general principle: Rather than coming up with a bunch of features and planning a multi-month release, come up with new ideas continually and try them out individually on users. With enough thought, even big features or large-scale changes can be implemented as a series of smaller steps to get faster feedback, with the ability to change course or stop at any point if your idea is found wanting. With a cross-functional team working to deliver these small increments in hours or days, you can be more innovative than your competition and maximize your return on investment.

In this article, I'll discuss five practices of continuous delivery that can help you to create the most efficient path from hypothesis to continuous feedback:

- Start with a minimum viable product (MVP).
- Measure the value of your features.
- Perform *just enough* analysis up front.
- Do less.
- Include feature toggles in your stories.

Start with a Minimum Viable Product (MVP)

“If you are not embarrassed by [the first version of your product], you’ve launched too late!” Reid Hoffman, co-founder and chairman of LinkedIn (see “Ten Entrepreneurship Rules for Building Massive Companies”).

If the start of your project is marked by a big requirements document landing on the project manager’s desk, you’ve already failed. One of the key ideas that the Lean Startup movement has popularized is the minimum viable product (MVP), defined as the least possible amount of work you can do to test your business hypothesis.

Of course, people in manufacturing have been producing minimum viable products for decades—they’re called *prototypes*. As with prototypes, you don’t need to show the whole world your minimum viable product—you could expose it to a select group of beta users. It might not even be working software—you could create a pretotype instead, to gather data without writing a line of code.

The final version you reveal to the wider world may be a much more polished product—if that’s important to your target audience. One company releases its MVP iPhone apps under a different brand. The point is simply to get statistically significant feedback on whether your business plan is likely to succeed; then, as a secondary goal, prove out the delivery process.

Crucially, working out how the MVP looks requires a cross-functional team consisting of representatives from both business and technology. Roles you want on that team include User Experience Designer (UX), analysis, testing, development, operations, and infrastructure. (Of course, one person could potentially play multiple roles, so you don’t necessarily need an enormous committee to get something done.)

Since your minimum viable product is going to take a small team just a few weeks—and under no circumstances more than a few months—to build, you don’t need a lot of ceremony at this point, because you’re not betting the company or spending big wads of cash.

Measure the Value of Your Features

“Metrics are part of your product.” John Allspaw, VP of Technical Operations, Etsy (see “Building Resilience in Web Development and Operations”).

Another key lean concept is validated learning, in which you gather actionable metrics based on real usage of your product, without asking people. As TV’s Dr. Gregory House likes to say, people lie—although, more charitably, you might say that they don’t know what they want. Treat your users as experimental subjects rather than intelligent agents.

You need to be able to answer questions like these:

- Did the changes we made to the product improve signup, retention, revenue? Or is it time to pivot?
- Which version of our new feature came out better when we did A/B testing?
- All our system metrics look okay, but a user reports that our site isn’t working. Are we down?
- Which features of our product are generating the most revenue?

You should be able to answer these questions without trawling through Apache logs, trying to instrument features retroactively, or running custom queries. These questions should be answerable by looking at a dashboard, and the information should be completely auditable.

In his book *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses* (Crown Business, 2011), Eric Ries tells the story of Grockit:

Following the lean manufacturing principle of kanban, [...]Grockit changed the product prioritization process. Under the new system, user stories were not considered complete until they led to validated learning. Thus, stories could be cataloged as being in one of four states of development: in the product backlog, actively being built, done (feature complete from a technical point of view), or in the process of being validated. Validated was defined as “knowing whether the story was a good idea to have been done in the first place.” This validation usually would come in the form of a split test showing a change in customer behavior but also might include customer interviews or surveys.

This kind of learning is only possible if metrics are built into the stories that are being played.

This principle might appear to be web-specific, but it is also true of embedded systems and user-installed products. All types of systems need to gather these kinds of metrics for remote debugging and fault-reporting purposes, as well as for understanding usage patterns.

Perform Just Enough Analysis Up Front

“You know when you are not doing iterative development when [the] team tries

to complete specifications before programming.” Bas Vodde, “History of Nokia Test.”

Once you have an idea for a minimum viable product, you need to start delivering software. The first step is analysis. But having a backlog of fully analyzed stories is wasteful. To analyze stories fully, you need input from customers, developers, testers, UX, and users. If your team is spending time gathering this information, they're not working on delivering valuable functionality and getting real feedback from users based on working software.

How much analysis needs to be done up front? Before development starts on a story, we only care about three things:

- What is the marginal value of delivering the story?
- What is the marginal cost of delivering the story?
- Do we have enough information to *begin* delivering the story?

The first two questions are important so that we can decide, at the point that delivery capacity becomes available, which story the team should play next. In order to do this, we need to work out which story will maximize economic outcome. The final two questions are closely related, because the amount of information required to estimate the marginal cost of delivering a story is usually more than you need to begin delivering it. But as my colleague Peter Gillard-Moss once pointed out to me, you need at least one acceptance criterion.

This discipline of doing just enough analysis needs to continue throughout the lifecycle of the project, with the emphasis on creating very small, incremental stories. This brings us to the next practice, doing less.

Do Less

“[If] you find yourself running out of room on the cards, use smaller cards.” Philp (see “Re: [XP] Re: Token definition in User Stories”).

Perhaps the most popular acronym in Agile analysis is Bill Wake's INVEST principles. Wake says that good stories are *independent, negotiable, valuable, estimable, small, and testable*. I want to focus in particular on “small.”

People often think that features and stories are interchangeable. Sometimes people think of a feature as something that might take weeks to complete. I remember on one project being presented with “stories” that came in the form of Word documents that were many pages long.

There's a reason why XP told people to write the summary of a story on 3 [ts] 5 index cards. Stories shouldn't take more than a couple of days to complete. Anything bigger than a week is way too long, and should be broken up into smaller bits. Why?

- To make sure that we're getting constant feedback on our work from users, so we can find out whether what we're doing is actually valuable
- To validate that we're actually getting work done—not just “dev complete,” but releasable—so we can demonstrate that we're making real progress
- To prevent us from creating big batches of work that will be painful to integrate, test, and deploy
- To ensure that we're constantly testing and improving our delivery process

The usual objection is that you can't do anything valuable in a couple of days. I think this statement demonstrates both a lack of imagination and a misinterpretation of what constitutes value. The

value of a story can only be measured by showing it to a user, as I mentioned previously. Sure, you're not going to complete a whole feature in a couple of days. But you can complete and get feedback on the kernel of the feature. For example, say you're working on a hotel booking site, and you want to add a feature to allow people to choose whether they want breakfast. Don't create this feature for all hotels or for all partner sites. Instead, start with a story that allows you to add that feature for a single hotel, with no configuration options, and get feedback for that possibility before you proceed further.

Whatever you do, don't decompose features into "stories" focused on one tier of the solution, such as a "story" to implement the persistence layer, another to do the business logic, and a third to implement the UI. Stories should *always* be small vertical slices. If you're going to have to do a bunch of integration work, focus on making the vertical slice as narrow as possible. For example, if you have to pass a series of messages to another system as part of a piece of functionality, your first story should be to pass the simplest possible message in order to drive out the end-to-end integration.

Forcing yourself to work out the true value of a feature by constantly stripping away functionality until you get to the smallest possible bit of functionality you can show to users (and thus learn from) is a difficult but tremendously valuable discipline. You can use that feedback to determine what small, incremental step—another couple of days' work—you should do next, or whether you should just stop working on the feature at all in its current form, because it's not as valuable as you thought.

Always keep in the front of your mind that the biggest source of waste in soft-

ware delivery is unused functionality—more than half of all functionality developed is never or rarely used, according to one study.

Instead of asking, "What more do we need to put into this feature to make sure people are going to love it?" or "What extra features do we need to put into this release to create a really great product?" ask this: "Can we deliver what we have *now*? If not, why not?" Do less, so that you can focus your efforts instead on learning from your users.

You may not want to show your feature to the world until a number of stories have been played. There's always a tradeoff between getting something out and making sure that everything released is of sufficient quality (as determined by the user). That's when you need feature toggles, as described in the next section.

Include Feature Toggles in Your Stories

"Right now, out on Facebook.com, is all the code for every major thing we're gonna do in the next six months and beyond." Chuck Rossi (see TechCrunch discussion about Rossi's May 26, 2011 "Facebook Tech Talk").

If you want to increase your release frequency, it's essential to decouple your deployments from the act of taking a feature live. Feature toggles (also known as *bits or switches*) is a pattern whereby you can control who can see which features. This setup allows you to release new versions of your software that include partially complete features, containing perhaps a story or two of work, but nothing you'd want the general public using.

Facebook's *Gatekeeper* software allows Facebook to control dynamically who

can see which features; for example, exposing a given feature to only 10% of Facebook users, or to people who work at Facebook, or to female users under the age of 25, or to UK residents, and so on. (The Gatekeeper even has a toggle that makes a feature visible to everyone except TechCrunch employees.) This capability allows the Facebook programmers to try a feature on only a certain demographic, and to roll it out incrementally over time.

Feature toggles represent a crucial constraint on the way you break down features into stories. One common objection to the use of feature toggles goes like this: "Some of my stories spray controls all over the user interface. It's going to be a lot of work to add a toggle to make this story invisible to users." The answer? Decompose your features into stories in such a way that it's easy to add the feature toggles.

Feature toggles should be a first-class part of your stories. One team at Orbitz builds feature bits into their stories such that the first task they perform when they play a story is to add the feature bit for that story. The feature bit forms part of the value of the story, and of course the work to add the feature bit gets estimated along with the story. If the estimate for the task of adding the feature bit is high, it's a sign that you've decomposed your feature poorly.

In addition to enabling incremental delivery of functionality, feature toggles have other important applications. They make it possible to degrade your service gracefully (for example, by turning off some resource-intensive feature such as a recommendation engine) in the case of unexpected load. They also let you turn off a problematic feature when a release goes wrong, as an alternative to rolling back your deployment.

Conclusions

A common failure mode of software projects is what Don Reinertsen, in his book *The Principles of Product Development Flow: Second Generation Lean Product Development* (Celeritas, 2009), calls the *large batch death-spiral*, whereby product owners, in an attempt to ensure the success of their product, add more and more scope as the project progresses, in a vicious circle that leads to exponential growth of cost and schedule.

Continuous delivery allows teams to reduce dramatically the transaction cost of releasing high-quality software, so you can do it much more frequently, providing a much richer and faster feedback cycle from users back to product teams. But, in turn, you need to change the way you think about managing the flow of work through the software delivery process. In particular, if you're doing continuous delivery correctly, the technology

people are no longer the constraint in terms of testing new ideas on users. With traditional delivery processes, we have to wait weeks or months to see our ideas turned into working software. By delivering small increments of functionality and getting feedback, we can constantly be thinking, "What should we try next?" No team that has achieved this transformation wants to go back to the old way of working.

Using traditional delivery methods, we had to be careful to select which ideas we would actually attempt to deliver, because the software delivery process was so expensive. Of course, that sifting process wasn't based on real data. With continuous delivery, we have what my colleague Darius Kumana calls an "airbag for innovation failure"; we can try crazily innovative ideas cheaply and safely at any stage in the evolution of the system, mitigating the risk if they don't work out by (for example) exposing them to only a

small group of users. Continuous delivery liberates us by massively reducing the cost and risk of releasing software, putting analysts back where they belong—powering innovation.

Thanks to Chad Wathington, Elena Yatzeck, Dan McClure, and Darius Kumana for feedback on an earlier draft of this article.

The GOTO Conference Magazine wants to thank InformIT, <http://www.informit.com> for granting the permission to use this article.

Jez Humble will run the "Continuous Delivery" Training Course at GOTO Aarhus 2012. He takes the unique approach of moving from release back through testing to development practices, analyzing at each stage how to improve collaboration and increase feedback so as to make the delivery process as fast and efficient as possible. Furthermore Jez will present in both the "Agile Perspectives" and the "Continuous Delivery" track at the conference.



Skab det stærkeste grundlag for IT-udvikling

- på tværs af afdelinger og interesser med Application Lifecycle Management



Fra et projekt går i gang til det bliver afsluttet, ligger et massivt arbejde for en stor gruppe mennesker – fra arkitekter, udviklere og testere til kunder, projektledere og direktion. En vellykket proces stiller krav om adgang og rapportering fra start til slut. Derfor er Application Lifecycle Management vigtig. Denne proces kan understøttes med løsninger fra IBM Rational, hvor der netop bliver lagt vægt på at håndtereflowet mellem mennesker, opgaver og informationer.

Gør en svær proces mere enkel – og rentabel

Application Lifecycle Management, også kaldet ALM, koordinerer projekter på en så effektiv og logisk måde, at succesraten bliver væsentligt større, mens omkostningerne og fejlmargen holdes helt i bund. I stedet for at arbejde i siloer, som fører til misforståelser, arbejder du i en åben proces med adgang for alle, der arbejder på projektet. "Rational Solution for Collaborative Lifecycle Management" er navnet på IBM's ALM løsning, der indeholder:

- **Rational Requirements Composer**, der anvendes til at definere og styre krav
- **Rational Team Concert** der sikrer sammenhæng mellem work items, levende planer og kodeændringer
- **Rational Quality Manager** sikrer planlægning, eksekvering og rapportering af test

Sporbarhed og rapportering limer de 3 produkter sammen til "Collaborative Lifecycle Management"



Bliv inspireret af mulighederne med ALM

Mød os på GOTO 2012

10gen

the
MongoDB
company

10gen.com | MongoDB.org

CouchConf BERLIN

CouchConf Berlin
30 October 2012

Use the code **GOTO_DISCOUNT**
to get **10% off your ticket!**

Register now at

www.couchbase.com/couchconf-berlin

Why attend?

CouchConf Berlin is a one-day event focused on Couchbase NoSQL technology. Meet the experts, hear real world use cases and learn more about NoSQL at this dual-track full day conference.

Your ticket will get you the full day of sessions plus access to the after party bash!

Presented by

CouchBase

riak

[ree-ahk] **-noun**

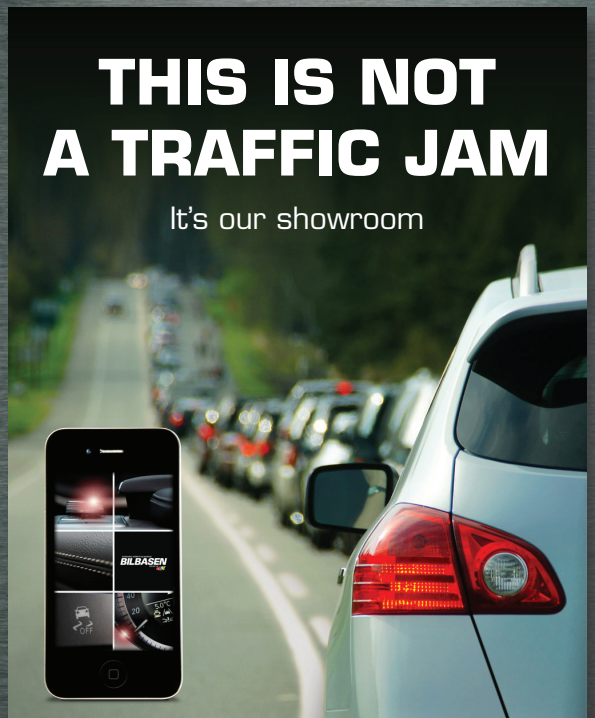
1. The most powerful open-source, distributed database you'll ever put into production.
2. The feeling you get when disaster strikes and you realize you haven't lost any data.

(See also: 'paradise,' 'utopia')



THIS IS NOT A TRAFFIC JAM

It's our showroom



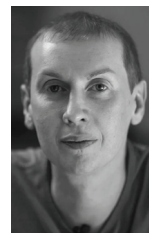
THE DANISH ROADS ARE OUR SHOWROOM
BilBasen has Denmark's largest selection of cars. If you see a car you like in the traffic jam, you can find one just like it with our new, free **BilBasen app** – here and now. Try it for yourself!

BILBASEN
PART OF **etay**

Memory Access Patterns Are Important



Martin Thompson
High-Performance Computing Specialist



In high-performance computing it is often said that the cost of a cache-miss is the largest performance penalty for an algorithm. For many years the increase in speed of our processors has greatly outstripped latency gains to main-memory. Bandwidth to main-memory has greatly increased via wider, and multi-channel, buses however the latency has not significantly reduced. To hide this latency our processors employ evermore complex cache sub-systems that have many layers.

The 1994 paper “Hitting the memory wall: implications of the obvious” describes the problem and goes on to argue that caches do not ultimately help because of compulsory cache-misses. I aim to show that by using access patterns which display consideration for the cache hierarchy, this conclusion is not inevitable.

Let’s start putting the problem in context with some examples. Our hardware tries to hide the main-memory latency via a number of techniques. Basically three major bets are taken on memory access patterns:

1. **Temporal:** Memory accessed recently will likely be required again soon.
2. **Spatial:** Adjacent memory is likely to be required soon.
3. **Striding:** Memory access is likely to follow a predictable pattern.

To illustrate these three bets in action let’s write some code and measure the results.

1. Walk through memory in a linear fashion being completely predictable.
2. Pseudo randomly walk round memory within a restricted area then move on. This restricted area is what is commonly known as an operating system page of memory.
3. Pseudo randomly walk around a large area of the heap.



Code

The following code should be run with the -Xmx4g JVM option.

```
public class TestMemoryAccessPatterns
{
    private static final int LONG_SIZE = 8;
    private static final int PAGE_SIZE = 2 * 1024 * 1024;
    private static final int ONE_GIG = 1024 * 1024 * 1024;
    private static final long TWO_GIG = 2L * ONE_GIG;

    private static final int ARRAY_SIZE = (int)(TWO_GIG / LONG_SIZE);
    private static final int WORDS_PER_PAGE = PAGE_SIZE / LONG_SIZE;

    private static final int ARRAY_MASK = ARRAY_SIZE - 1;
    private static final int PAGE_MASK = WORDS_PER_PAGE - 1;

    private static final int PRIME_INC = 514229;

    private static final long[] memory = new long[ARRAY_SIZE];

    static
    {
        for (int i = 0; i < ARRAY_SIZE; i++)
        {
            memory[i] = 777;
        }
    }

    public enum StrideType
    {
        LINEAR_WALK
        {
            public int next(final int pageOffset, final int wordOffset, final int pos)
            {
                return (pos + 1) & ARRAY_MASK;
            }
        },
        RANDOM_PAGE_WALK
        {
            public int next(final int pageOffset, final int wordOffset, final int pos)
            {
                return pageOffset + ((pos + PRIME_INC) & PAGE_MASK);
            }
        },
        RANDOM_HEAP_WALK
        {
            public int next(final int pageOffset, final int wordOffset, final int pos)
            {
                return (pos + PRIME_INC) & ARRAY_MASK;
            }
        };

        public abstract int next(int pageOffset, int wordOffset, int pos);
    }

    public static void main(final String[] args)
    {
        final StrideType strideType;
        switch (Integer.parseInt(args[0]))
        {
            case 1:
                strideType = StrideType.LINEAR_WALK;
                break;

            case 2:
                strideType = StrideType.RANDOM_PAGE_WALK;
                break;

            case 3:
                strideType = StrideType.RANDOM_HEAP_WALK;
                break;

            default:
                throw new IllegalArgumentException("Unknown StrideType");
        }

        for (int i = 0; i < 5; i++)
        {
            perfTest(i, strideType);
        }
    }
}
```



```

private static void perfTest(final int runNumber, final StrideType strideType)
{
    final long start = System.nanoTime();

    int pos = -1;
    long result = 0;
    for (int pageOffset = 0; pageOffset < ARRAY_SIZE; pageOffset += WORDS_PER_PAGE)
    {
        for (int wordOffset = pageOffset, limit = pageOffset + WORDS_PER_PAGE;
            wordOffset < limit;
            wordOffset++)
        {
            pos = strideType.next(pageOffset, wordOffset, pos);
            result += memory[pos];
        }
    }

    final long duration = System.nanoTime() - start;
    final double nsOp = duration / (double)ARRAY_SIZE;

    if (208574349312L != result)
    {
        throw new IllegalStateException();
    }

    System.out.format("%d - %.2fns %s\n",
        Integer.valueOf(runNumber),
        Double.valueOf(nsOp),
        strideType);
}
}

```

Results

Intel U4100 @ 1.3GHz, 4GB RAM DDR2 800MHz,
Windows 7 64-bit, Java 1.7.0_05

```

=====
0 - 2.38ns LINEAR_WALK
1 - 2.41ns LINEAR_WALK
2 - 2.35ns LINEAR_WALK
3 - 2.36ns LINEAR_WALK
4 - 2.39ns LINEAR_WALK

0 - 12.45ns RANDOM_PAGE_WALK
1 - 12.27ns RANDOM_PAGE_WALK
2 - 12.17ns RANDOM_PAGE_WALK
3 - 12.22ns RANDOM_PAGE_WALK
4 - 12.18ns RANDOM_PAGE_WALK

0 - 152.86ns RANDOM_HEAP_WALK
1 - 151.80ns RANDOM_HEAP_WALK
2 - 151.72ns RANDOM_HEAP_WALK
3 - 151.91ns RANDOM_HEAP_WALK
4 - 151.36ns RANDOM_HEAP_WALK

```

Intel i7-860 @ 2.8GHz, 8GB RAM DDR3 1333MHz,
Windows 7 64-bit, Java 1.7.0_05

```

=====
0 - 1.06ns LINEAR_WALK
1 - 1.05ns LINEAR_WALK
2 - 0.98ns LINEAR_WALK
3 - 1.00ns LINEAR_WALK
4 - 1.00ns LINEAR_WALK

0 - 3.80ns RANDOM_PAGE_WALK
1 - 3.85ns RANDOM_PAGE_WALK
2 - 3.79ns RANDOM_PAGE_WALK
3 - 3.65ns RANDOM_PAGE_WALK
4 - 3.64ns RANDOM_PAGE_WALK

0 - 30.04ns RANDOM_HEAP_WALK
1 - 29.05ns RANDOM_HEAP_WALK
2 - 29.14ns RANDOM_HEAP_WALK
3 - 28.88ns RANDOM_HEAP_WALK
4 - 29.57ns RANDOM_HEAP_WALK

```

Intel i7-2760QM @ 2.40GHz, 8GB RAM DDR3 1600MHz,

```
Linux 3.4.6 kernel 64-bit, Java 1.7.0_05
=====
0 - 0.91ns LINEAR_WALK
1 - 0.92ns LINEAR_WALK
2 - 0.88ns LINEAR_WALK
3 - 0.89ns LINEAR_WALK
4 - 0.89ns LINEAR_WALK

0 - 3.29ns RANDOM_PAGE_WALK
1 - 3.35ns RANDOM_PAGE_WALK
2 - 3.33ns RANDOM_PAGE_WALK
3 - 3.31ns RANDOM_PAGE_WALK
4 - 3.30ns RANDOM_PAGE_WALK

0 - 9.58ns RANDOM_HEAP_WALK
1 - 9.20ns RANDOM_HEAP_WALK
2 - 9.44ns RANDOM_HEAP_WALK
3 - 9.46ns RANDOM_HEAP_WALK
4 - 9.47ns RANDOM_HEAP_WALK
```

Analysis

I ran the code on 3 different CPU architectures illustrating generational steps forward for Intel. It is clear from the results that each generation has become progressively better at hiding the latency to main-memory based on the 3 bets described above for a relatively small heap. This is because the size and sophistication of various caches keep improving. However as memory size increases they become less effective. For example, if the array is doubled to be 4GB in size, then the average latency increases from ~30ns to ~55ns for the i7-860 doing the random heap walk.

It seems that for the linear walk case, memory latency does not exist. However as we walk around memory in an evermore random pattern then the latency starts to become very apparent.

The random heap walk produced an interesting result. This is a our worst case scenario, and given the hardware specifications of these systems, we could be looking at 150ns, 65ns, and 75ns for the above tests respectively based on memory controller and memory module latencies. For the Nehalem (i7-860) I can further subvert the cache sub-system by using a 4GB array resulting in ~55ns on average per iteration. The i7-2760QM has larger load buffers, TLB caches, and Linux is running with transparent huge pages which are all working to further hide the latency. By playing with different prime numbers for the stride, results can vary wildly depending on processor type, e.g. try `PRIME_INC = 39916801` for Nehalem. I'd like to test this on a much larger heap with Sandy Bridge.

The main take away is the more predictable the pattern of access to memory, then the better the cache sub-systems are at hiding main-memory latency. Let's look at these cache sub-systems in a little detail to try and understand the observed results.

Hardware Components

We have many layers of cache plus the pre-fetchers to consider for how latency gets hidden. In this section I'll try and cover the major components used to hide latency that our hardware and systems software friends have put in place. We will investigate these latency hiding components and use the Linux perf and Google Lightweight Performance Counters utilities to retrieve the performance counters from our CPUs which tell how effective these components are when we execute our programs. Performance counters are CPU specific and what I've used here are specific to Sandy Bridge.

Data Caches

Processors typically have 2 or 3 layers of data cache. Each layer as we move out is progressively larger with increasing latency. The latest Intel processors have 3 layers (L1D, L2, and L3); with sizes 32KB, 256KB, and 4-30MB; and ~1ns, ~4ns, and ~15ns latency respectively for a 3.0GHz CPU.

Data caches are effectively hardware hash tables with a fixed number of slots for each hash value. These slots are known as “ways”. An 8-way associative cache will have 8 slots to hold values for addresses that hash to the same cache location. Within these slots the data caches do not store words, they store cache-lines of multiple words. For an Intel processor these cache-lines are typically 64-bytes, that is 8 words on a 64-bit machine. This plays to the spatial bet that adjacent memory is likely to be required soon, which is typically the case if we think of arrays or fields of an object.

Data caches are typically evicted in a LRU manner. Caches work by using a write-back algorithm where stores need only be propagated to main-memory when a modified cache-line is evicted. This gives rise to the interesting phenomenon that a load can cause a write-back to the outer cache layers and eventually to main-memory.

```
perf stat -e L1-dcache-loads,L1-dcache-load-misses java -Xmx4g
TestMemoryAccessPatterns $
```

```
Performance counter stats for 'java -Xmx4g TestMemoryAccessPat-
terns 1':
```

```
1,496,626,053 L1-dcache-loads
274,255,164 L1-dcache-misses
# 18.32% of all L1-dcache hits
```

```
Performance counter stats for 'java -Xmx4g TestMemoryAccessPat-
terns 2':
```

```
1,537,057,965 L1-dcache-loads
1,570,105,933 L1-dcache-misses
# 102.15% of all L1-dcache hits
```

```
Performance counter stats for 'java -Xmx4g TestMemoryAccessPat-
terns 3':
```

```
4,321,888,497 L1-dcache-loads
1,780,223,433 L1-dcache-misses
# 41.19% of all L1-dcache hits
```

```
likwid-perfctr -C 2 -g L2CACHE java -Xmx4g TestMemoryAccessPat-
terns $
```

```
java -Xmx4g TestMemoryAccessPatterns 1
```

Event	core 2
INSTR_RETIRED_ANY	5.94918e+09
CPU_CLK_UNHALTED_CORE	5.15969e+09
L2_TRANS_ALL_REQUESTS	1.07252e+09
L2_RQSTS_MISS	3.25413e+08

Metric	core 2
Runtime [s]	2.15481
CPI	0.867293
L2 request rate	0.18028
L2 miss rate	0.0546988
L2 miss ratio	0.303409

Event	core 2
L3_LAT_CACHE_REFERENCE	1.26545e+08
L3_LAT_CACHE_MISS	2.59059e+07

```
java -Xmx4g TestMemoryAccessPatterns 2
```



```

+-----+-----+
|          Event          |   core 2   |
+-----+-----+
| INSTR_RETIRED_ANY      | 1.48772e+10 |
| CPU_CLK_UNHALTED_CORE  | 1.64712e+10 |
| L2_TRANS_ALL_REQUESTS  | 3.41061e+09 |
| L2_RQSTS_MISS          | 1.5547e+09  |
+-----+-----+
|          Metric        |   core 2   |
+-----+-----+
| Runtime [s]            | 6.87876     |
| CPI                    | 1.10714     |
| L2 request rate        | 0.22925     |
| L2 miss rate           | 0.104502    |
| L2 miss ratio          | 0.455843    |
+-----+-----+
|          Event          |   core 2   |
+-----+-----+
| L3_LAT_CACHE_REFERENCE | 1.52088e+09 |
| L3_LAT_CACHE_MISS      | 1.72918e+08 |
+-----+-----+

java -Xmx4g TestMemoryAccessPatterns 3
+-----+-----+
|          Event          |   core 2   |
+-----+-----+
| INSTR_RETIRED_ANY      | 6.49533e+09 |
| CPU_CLK_UNHALTED_CORE  | 4.18416e+10 |
| L2_TRANS_ALL_REQUESTS  | 4.67488e+09 |
| L2_RQSTS_MISS          | 1.43442e+09 |
+-----+-----+
|          Metric        |   core 2   |
+-----+-----+
| Runtime [s]            | 17.474      |
| CPI                    | 6.4418      |
| L2 request rate        | 0.71973     |
| L2 miss rate           | 0.220838    |
| L2 miss ratio          | 0.306835    |
+-----+-----+
|          Event          |   core 2   |
+-----+-----+
| L3_LAT_CACHE_REFERENCE | 1.40079e+09 |
| L3_LAT_CACHE_MISS      | 1.34832e+09 |
+-----+-----+

```

Note: The cache-miss rate of the combined L1D, L2 and L3 increases significantly as the pattern of access becomes more random.

Translation Lookaside Buffers (TLBs)

Our programs deal with virtual memory addresses that need to be translated to physical memory addresses. Virtual memory systems do this by mapping pages. We need to know the offset for a given page and its size for any memory operation. Typically page sizes are 4KB and are gradually moving to 2MB and greater. Linux introduced Transparent Huge Pages in the 2.6.38 kernel giving us 2MB pages. The translation of virtual memory pages to physical pages is maintained by the page table. This translation can require multiple accesses to the page table which is a huge performance penalty. To accelerate this lookup, processors have a small hardware cache at each cache level called the TLB cache. A miss on the TLB cache can be hugely expensive because the page table may not be in a nearby data cache. By moving to larger pages, a TLB cache can cover a larger address range for the same number of entries.

```
perf stat -e dTLB-loads,dTLB-load-misses java -Xmx4g TestMemoryAccessPatterns $
```

```
Performance counter stats for 'java -Xmx4g TestMemoryAccessPatterns 1':
```

```
1,496,128,634 dTLB-loads
310,901 dTLB-misses
```

```

#      0.02% of all dTLB cache hits

Performance counter stats for 'java -Xmx4g TestMemoryAccessPat-
terns 2':
    1,551,585,263 dTLB-loads
    340,230 dTLB-misses
#      0.02% of all dTLB cache hits

Performance counter stats for 'java -Xmx4g TestMemoryAccessPat-
terns 3':
    4,031,344,537 dTLB-loads
    1,345,807,418 dTLB-misses
#      33.38% of all dTLB cache hits

```

Note: We only incur significant TLB misses when randomly walking the whole heap when huge pages are employed.

Hardware Pre-Fetchers

Hardware will try and predict the next memory access our programs will make and speculatively load that memory into fill buffers. This is done at it simplest level by pre-loading adjacent cache-lines for the spatial bet, or by recognising regular stride based access patterns, typically less than 2KB in stride length. The tests below we are measuring the number of loads that hit a fill buffer from a hardware pre-fetch.

```
likwid-perfctr -C 2 -g LOAD_HIT_PRE_HW_PF:PMC0 java -Xmx4g Test-
MemoryAccessPatterns $
```

```

java -Xmx4g TestMemoryAccessPatterns 1
+-----+-----+
|          Event          |   core 2   |
+-----+-----+
| LOAD_HIT_PRE_HW_PF | 1.31613e+09 |
+-----+-----+

```

```

java -Xmx4g TestMemoryAccessPatterns 2
+-----+-----+
|          Event          |   core 2   |
+-----+-----+
| LOAD_HIT_PRE_HW_PF | 368930 |
+-----+-----+

```

```

java -Xmx4g TestMemoryAccessPatterns 3
+-----+-----+
|          Event          |   core 2   |
+-----+-----+
| LOAD_HIT_PRE_HW_PF | 324373 |
+-----+-----+

```

Note: We have a significant success rate for load hits with the pre-fetcher on the linear walk.

Memory Controllers and Row Buffers

Beyond our last level cache (LLC) sits the memory controllers that manage access to the SDRAM banks. Memory is organised into rows and columns. To access an address, first the row address must be selected (RAS), then the column address is selected (CAS) within that row to get the word. The row is typically a page in size and loaded into a row buffer. Even at this stage the hardware is still helping hide the latency. A queue of memory access requests is maintained and re-ordered so that multiple words can be fetched from the same row if possible.

Non-Uniform Memory Access (NUMA)

Systems now have memory controllers on the CPU socket. This move to on-socket memory controllers gave an ~50ns latency reduction over existing front side bus (FSB) and external Northbridge memory controllers. Systems with multiple sockets employ memory interconnects, QPI from Intel, which are used when one CPU wants to access memory managed by another CPU socket. The presence of these interconnects gives rise to the non-uniform nature of server memory access. In a 2-socket system memory may be local or 1 hop away. On a 8-socket system memory can be up to 3 hops away, where each hop adds 20ns latency in each direction.

What does this mean for algorithms?

The difference between an L1D cache-hit, and a full miss resulting in main-memory access, is 2 orders of magnitude; i.e. <1ns vs. 65-100ns. If algorithms randomly walk around our ever increasing address spaces, then we are less likely to benefit from the hardware support that hides this latency.

Is there anything we can do about this when designing algorithms and data-structures? Yes there is a lot we can do. If we perform chunks of work on data that is co-located, and we stride around memory in a predictable fashion, then our algorithms can be many times faster. For example rather than using bucket and chain hash tables, like in the JDK, we can employ hash tables using open-addressing with linear-probing. Rather than using linked-lists or trees with single items in each node, we can store an array of many items in each node.

Research is advancing on algorithmic approaches that work in harmony with cache sub-systems. One area I find fascinating is Cache Oblivious Algorithms. The name is a bit misleading but there are some great concepts here for how to improve software performance and better execute in parallel. This article is a great illustration of the performance benefits that can be gained.

Conclusion

To achieve great performance it is important to have sympathy for the cache sub-systems. We have seen in this article what can be achieved by accessing memory in patterns which work with, rather than against, these caches. When designing algorithms and data structures, it is now vitally important to consider cache-misses, probably even more so than counting steps in the algorithm. This is not what we were taught in algorithm theory when studying computer science. The last decade has seen some fundamental changes in technology. For me the two most significant are the rise of multi-core, and now big-memory systems with 64-bit address spaces.

One thing is certain, if we want software to execute faster and scale better, we need to make better use of the many cores in our CPUs, and pay attention to memory access patterns.

Update: 06-August-2012

Trying to design a random walk algorithm for all processors and memory sizes is tricky. If I use the algorithm below then my Sandy Bridge processor is slower but the Nehalem is faster. The point is performance will be very unpredictable when you walk around memory in a random fashion. I've also included the L3 cache counters for more detail in all the tests.

```
private static final long LARGE_PRIME_INC = 70368760954879L;

RANDOM_HEAP_WALK
{
    public int next(final int pageOffset, final int wordOffset, final
int pos)
    {
        return (int)(pos + LARGE_PRIME_INC) & ARRAY_MASK;
    }
};
```

Intel i7-2760QM @ 2.40GHz, 8GB RAM DDR3 1600MHz,
Linux 3.4.6 kernel 64-bit, Java 1.7.0_05

=====

```
0 - 29.06ns RANDOM_HEAP_WALK
1 - 29.47ns RANDOM_HEAP_WALK
2 - 29.48ns RANDOM_HEAP_WALK
3 - 29.43ns RANDOM_HEAP_WALK
4 - 29.42ns RANDOM_HEAP_WALK
```

Performance counter stats for 'java -Xmx4g TestMemoryAccessPatterns
3':

```
9,444,928,682 dTLB-loads
4,371,982,327 dTLB-misses
#    46.29% of all dTLB cache hits

9,390,675,639 L1-dcache-loads
1,471,647,016 L1-dcache-misses
#    15.67% of all L1-dcache hits
```

Event	core 2
INSTR_RETIRED_ANY	7.71171e+09
CPU_CLK_UNHALTED_CORE	1.31717e+11
L2_TRANS_ALL_REQUESTS	8.4912e+09
L2_RQSTS_MISS	2.79635e+09

Metric	core 2
Runtime [s]	55.0094
CPI	17.0801
L2 request rate	1.10108
L2 miss rate	0.362611
L2 miss ratio	0.329324

Event	core 2
LOAD_HIT_PRE_HW_PF	3.59509e+06

Event	core 2
L3_LAT_CACHE_REFERENCE	1.30318e+09
L3_LAT_CACHE_MISS	2.62346e+07

This article is from Martin Thompson's blog *Mechanical Sympathy* where you can leave him a comment. At GOTO Aarhus 2012 Martin Thomson is hosting the *Mythbusters track*, in which he also speaks about *Mythbusting modern hardware to gain "Mechanical Sympathy"*. Martin will also be presenting in the *Systems that Scale track* and running the training course *Lock-free and high-performance algorithms*.



win a goto hoodie

limited 2012 edition



1. Pick your favorite goto speaker or topic

Check the website for inspiration.
www.gotocon.com



2. Make a tweet containing

#gotoaar
#gotohoodie
@speakername
(if (s)he has a twitter account)
Example

I look forward to seeing the #keynote speakers at #gotoaar: @falkvinge, Damian Conway, @shanselman, Dirk Duellmann & Anders Hjelsberg - hope to win a #gotohoodie too



3. Enter the contest

During the contest period we will draw two GOTO hoodie winners each week. We will contact you via Twitter if you are one of the lucky winners. You participate in the competition each time you tweet and meet the tweet requirement - that is: The more you tweet the more chances you have to win!

Conversations with a few of our GOTO Con

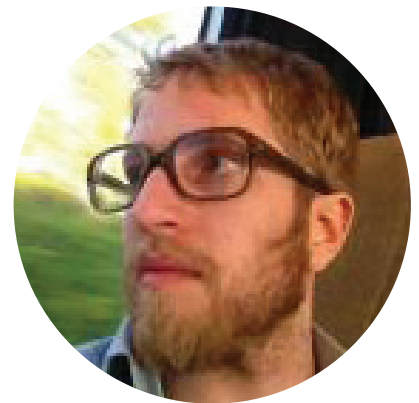
In May 2012 at GOTO Copenhagen, we had a chance to do a series of interviews with some of the speakers. These interviews are conversations with speakers around their particular area of expertise where they provide their own insights, interest and lessons learned.

Ian Plosker from Basho is interviewed by Stefan Edlich, author of two NoSQL books. Ian presents Riak, which is a distributed database that blurs the line between key-value and document-store. He also describes Bashos other products, which are all based to some extent on Riak. Anti-entropy is explained, and also why it is only necessary following some kind of failure.

"In the end it is all about providing the users with data in more interesting ways."



<http://www.youtube.com/watch?v=zjBRnWY4MIc>



Rich Hickey, the inventor of Clojure and Datomic is interviewed by Karl Krukow from LessPainful

Hear Rich tell the difference between place-oriented programmed and value-oriented programming, and the historical reasons for the two ways.

Inspired by Lisp, Rich finds a way to make parallelism easier in a JVM language. Clojure inherits a huge number of higher order functions from Lisp, that can take your collection, chew it, munge it, and spit it out in a new form. But they are all specified in a non-parallel way. How can we take these supremely useful library functions and adapt them for the multithreaded world?



<http://www.youtube.com/watch?v=sf-lrFERkvs>

Don Reinertsen, author of, Principles of Product Development Flow, is interviewed by Jesper Boeg from Trifork.

Don and Jesper talk about the impact of Dons book and the Toyota model and Lean in general. They also discuss why we keep the batch budgeting, big bang releases, and fixed budget contracts, when we have been told from empirical studies in agile development and Dons own more scientific approach that this does not work. Is it because of our training and school systems? Or is it human nature, like in the cases where we start too many things, because when an engineer is idle in 5 minutes, we find something for him to do.

In the end of the interview, Jesper asks for the difference between TOC (Theory of Constraints) and Kanban, the conclusion is that for product development, Kanban is much more efficient. But hear it for yourself!



<http://www.youtube.com/watch?v=G-NbrISyfoM>

Community Speakers



Chris Anderson, co-founder of Couchbase and an Apache CouchDB committer is interviewed by Prof. Dr. Stefan Edlich from Beuth HS of Technology Berlin. In this interview Chris talks about how he started with CouchDB as a user, had to teach himself Erlang to be a committer and ended up co-founding Couchbase. He discusses how the Membase server is now Couchbase server and the memcached protocol. He also shares the story of why they chose to reimplement the datapath of CouchDB from Erlang to C and C++ in order to get less CPU usage.

At GOTO Aarhus 2012 Chris Anderson speaks in the Navigating the BigData Ocean track about "Go Simple, Fast, Elastic with Couchbase Server and Document-oriented Data management". Chris is also a panel member in "The Aarhus 6" presentation, where six of the biggest names in NOSQL and NEWSQL technologies are the accused vendors who will need to prove to the public that their products CAN do what they "claim".



<http://youtu.be/ySpGGLrZt74>

Trisha Gee, a developer from LMAX, is interviewed by Simon Hem Pedersen and Ole Friis Østergaard from Trifork.

Trisha tells about her career, what she has been involved in in her 10+ years as a Java programmer. She does not envy the programmers working with other languages, she likes to spend time to become really good at one thing, and if need be, she could learn to use another language. The way LMAX tackles working with an "old" language like Java is by being part of the Java Committee, to voice their concerns over the language and the community. Trisha spends time on sharing her experience in the industry to younger people, because she hates to see the same mistake repeated again and again. In the end of the interview, Trisha shares with us her biggest aversion in IT.



<http://youtu.be/0OcczjVXQQc>



Kresten Krab Thorup og Aino Vonge Corry


What happens when the PC chair is only allowed to talk about three things from the program?

Kresten Krab Thorup, CTO of Trifork, and Aino Vonge Corry, founder of Metadeveloper play the role of the PC chair for GOTO Aarhus 2012 together. The program of speakers, talks and tracks is their baby. But in this video, there have been asked to talk about only three things each. See how they solve that impossible task of choosing, and what they say, when they only have a few minutes.




<http://www.youtube.com/watch?v=G7Euu-Y1t-Y&feature=plcp>

The System

 Barry O'Reilly
ThoughtWorks



 Lourenco Soares
ThoughtWorks

Manage value *not* cost

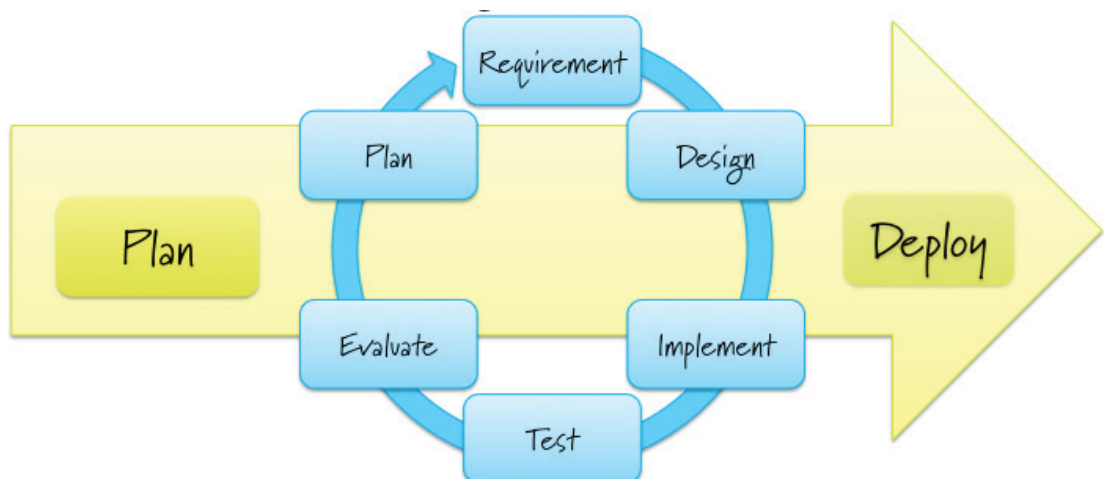
One of the foremost challenges we constantly encounter on development projects is the focus of teams and product managers on managing cost rather than value. In its most basic form this manifests itself daily in user stories where teams tend to concentrate on prioritisation, scope management and throughput in terms of the size of the work. The usual question I hear once a story is described is 'What size it is?', in short – how much is that going to cost?

Fundamentally these teams see software development as a 'cost' that must be managed. They forget that software development is a value creation process and hence the goal of our approach should be focused around creating and realising value, especially from the perspective of our users.

As Warren Buffet stated, 'Price is what you pay. Value is what you get'. What is needed is a change in mindset toward value creation, or at least exposing and making the idea of value visible to these teams and the wider organisation.

Realising value

Unlike waterfall approaches where all value is delivered at the very end of a long linear process of project analysis, development, testing and deployment, agile development emphasises the realisation of value much earlier and more often.



Systemico Model

Using techniques such as iterative development – always having working software, continuous design and delivery – the value created can be more frequently realised during the development process itself. However, challenges still exist in how to prioritise value for the user.

Teams continuously put energy into prioritisation and focus on throughput to achieve efficiency, but lose sight of effectiveness. No matter how efficiently we work, if we are continually building software that is of no value to our end users we are essentially creating waste, regardless of the pace we achieve.

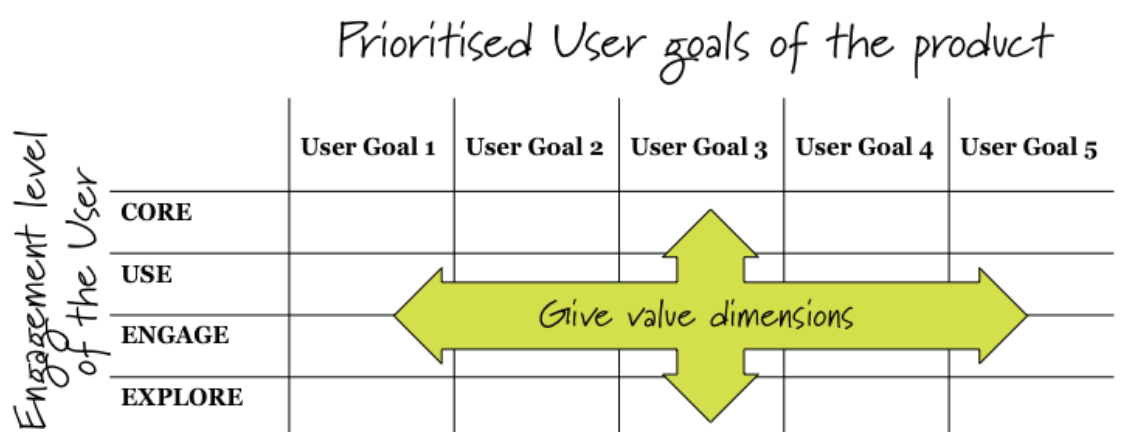
Teams and product managers need to take a holistic view of the entire product to support a user-centric approach to understanding, capturing and delivering value.

Challenge to prioritising value

Teams and product managers tend to favour grooming the product backlog through prioritisation and story sequencing to create a production line of user stories for the development team to consume. This approach appeals to the rational human mindset that development is a linear process. We create stories for the team to pick up one at a time, as prioritised in sequence. But this approach is flawed in the context of value creation. Value is not linear, and thus we need to take a systemic and holistic approach to application development when looking to create value for our users.

Systemico Model

To visualise the concept of value creation as systemic, holistic and in terms of the user, we have developed a framework we call the Systemico Model. The name is designed to reinforce the idea that value creation is not an isolated linear process. The model aims to make the requirements of the product visible to teams and product managers in terms of how it addresses user goals and engagement levels. We have found the Systemico Model to be especially useful when working on new products and domains that need to be customer and/or user-centric, especially when there is little or unknown validated learning in the product space.



User goals

Traditional analysis and design methods are often application-centric, defining the product in terms of what it will do, followed by how the user will interact with it. Instead, user-centric methods focus on *why* we should have a piece of functionality and *how* we can implement it.

To support the latter approach we define the product in terms of user goals and how it will assist the user in meeting those goals. This helps to focus our understanding on what motivates a user to use our product, and what features they will find valuable.

By defining and prioritising the requirements of the product in terms of the users' goals, we ensure that only functionality relevant to the user is created and visible to the team and product manager. Examples of user goals for a website may be to find content, add content, or purchase.

A challenge when assigning user stories to user goals is that there is not always a direct mapping of individual stories to goals – in some instances user stories can support multiple user goals. In this situation, we advise the team to agree on the goal that it believes to be the most appropriate based on the project landscape and customer insights they currently have. Should that view change in the future, the story can be moved to the most relevant user goal to reflect the latest available information.

User engagement

Along with user goals we consider the perspective of user engagement to analyse the level of interaction between the user and the product. We associate different observed behaviours with different degrees of user engagement.

Core: Functionality that satisfies the users' basic needs. These are the minimal expected features that users believe are standard on all products in a specific context *e.g.* user login/logout

Use: Enhanced functionality that increases usability of the product. Without these features the product has minimal appeal to the user *e.g.* displaying content on a page, user editing/manipulation features

Engage: Functionality that draws the user to interact further with the product. These features draw the user to return to the product in the future *e.g.* contribute to a site, provide reviews/ratings

Explore: Functionality that entices the user to go beyond simple interactions. These features build the strength of connection between the user and the product *e.g.* personalised services or recognition/rewards

The varying degrees of user engagement are not designed to indicate an increasing level of value from 'Core' to 'Explore'. Instead they are designed to support the focus of the proposition to resonate with the users' interaction with the product, thus offering value for the system.

A challenge when defining the degrees of user engagement is that they are subjective. This requires conversation between the team and product manager to create a shared understanding and definition of each category.

Applying user stories to the Systemico Model

By using the two perspectives of user goals and user engagement we are able to attribute extra value dimensions to user stories, aside from the requirement and size or cost of the work. This provides a deeper insight into the value a story will deliver, and will support the team and product manager to understand where to prioritise and invest effort.

AS A PRODUCT OWNER

I WANT TO ADD A PICTURE TO MY
PROFILE

SO THAT OTHER USERS CAN SEE MY
BEAUTIFUL FACE

Goal: 2 (Find content)

Engagement: Engage

Estimate: 5

We ask the product manager to prioritise what they believe to be the key user goals of the system and assign each story to a goal. Then we challenge the product manager to define the level of engagement each story is targeted at, in order to provide a second dimension to the user story.

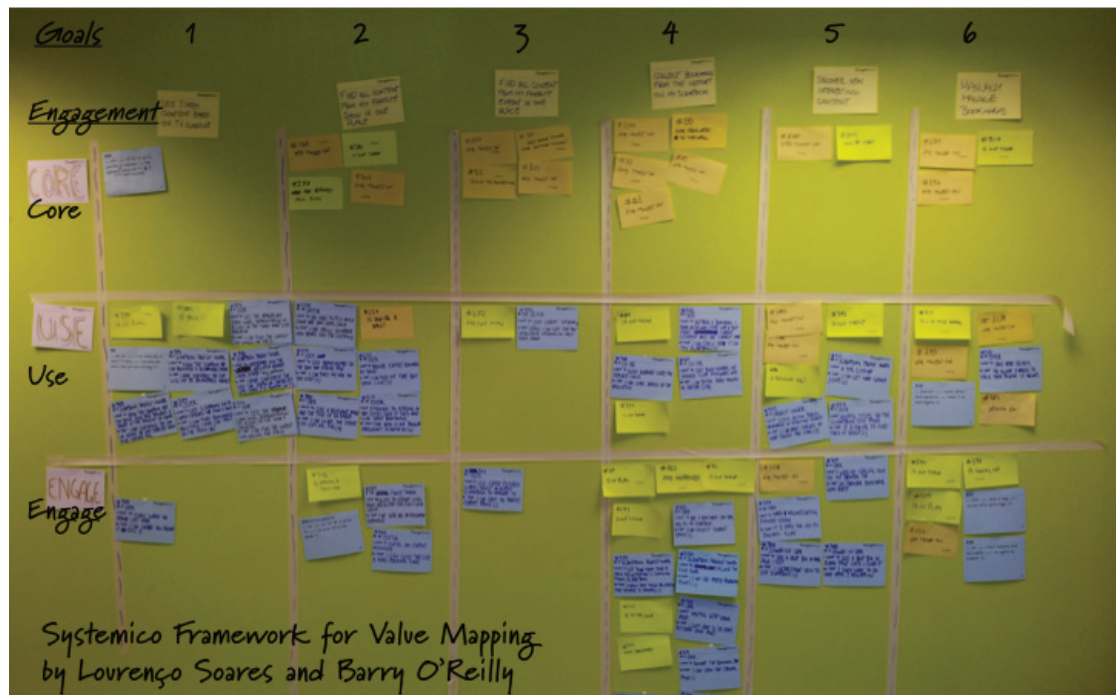
The end result is a value map of all user stories defined with extra dimensions beyond cost that relate specifically to the user of the product.

	User Goal 1	User Goal 2	User Goal 3	User Goal 4	User Goal 5
CORE	#121 #124	#143	#122	#158	#125
USE	#126	#130 #108		#129	
ENGAGE	#190			#127 #186	#132
EXPLORE		#133	#179		

Visualise value and effort

Another feature of the Systemico Model is its function as a visual tool to represent how and where effort has been invested in the product.

All user stories are mapped on a wall against the framework to visualise the user goal and engagement they are targeted at. We use blue cards to represent user stories that have not yet been picked up, replacing them with a yellow card when they enter our story wall



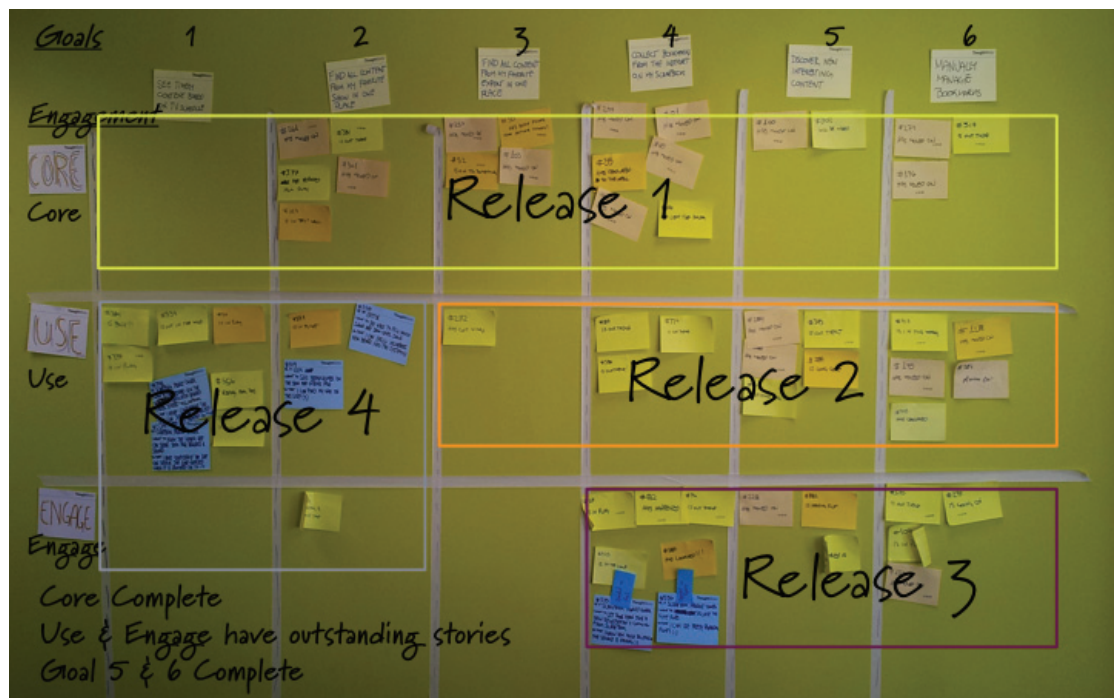
By providing more dimensions to user stories and mapping them to the framework, the team are provided with a visual, holistic, systemic and engaging view of the entire product which is centered on the user.

This serves as a visual reminder to the team of what areas have been worked on to date. It also supports decisions on how much effort should be invested in each user goal/engagement component to satisfy a user requirement before focusing on other areas of the product.

Good enough to go live

The Systemico Model also reinforces the concept of creating just enough functionality to showcase a value proposition to the user for validation before investing further in the feature.

We challenge our product managers to release the product as soon as viable combinations of user goals and user engagement components are completed by the team. In many instances, especially when working on new product development, completing the “Core” engagement component of a user goal should be sufficient to release the product to gain validated learning and insights from the market place.



A danger with traditional approaches to software development is that teams and product managers spend too much time and effort drilling into features that support individual user goals before presenting it to users for feedback and validation e.g. trying to make it perfect before release. This can lead to large amounts of waste if users later deem the feature undesirable or of low value.

By focusing on completing thin user goal/engagement combinations of the product and then releasing immediately, rapid feedback can be collected from users themselves on which features are appealing or more desirable. Once we obtain this feedback we can clarify and validate our user goals, and build greater depth to features that further enhance the users' engagement in that component.

The concept is referred to as 'do less [of what doesn't matter], deliver more [of what does]'. By focusing on what is truly important to our users we eliminate the creation of value propositions that are undesired and not useful to our customers.

The key concepts to remember are;

- the sooner the product is delivered
- the sooner you can get feedback from users
- the sooner you can improve it

Conclusion

The Systemico Model and user goal/engagement components are designed to support the team and product manager along the value creation process by providing a visual framework that allows further dimensions to be added to value propositions. The goal is to enable the team to leverage their development process to maximise value realisation, while minimising waste and over-engineered solutions that do not fit user needs.

Databases &



Stefan Edlich

Beuth University of Technology Berlin (App.Sc.)



With the explosion of demands due to new web business, the pressure on the database sector did also increase dramatically. It is more and more evident, that the traditional model of a 'one-size-fits-all' database used in the 90s is not a sustainable solution any more. And we all do know the reasons why database areas such as NoSQL and NewSQL has emerged: big data, massive write performance and availability, fast KV access, flexible scheme, easy maintainability, no single point of failure and so on. This has been perfectly captured by Matthew Aslett in the 451 Group Report "NoSQL, NewSQL and Beyond" [1].

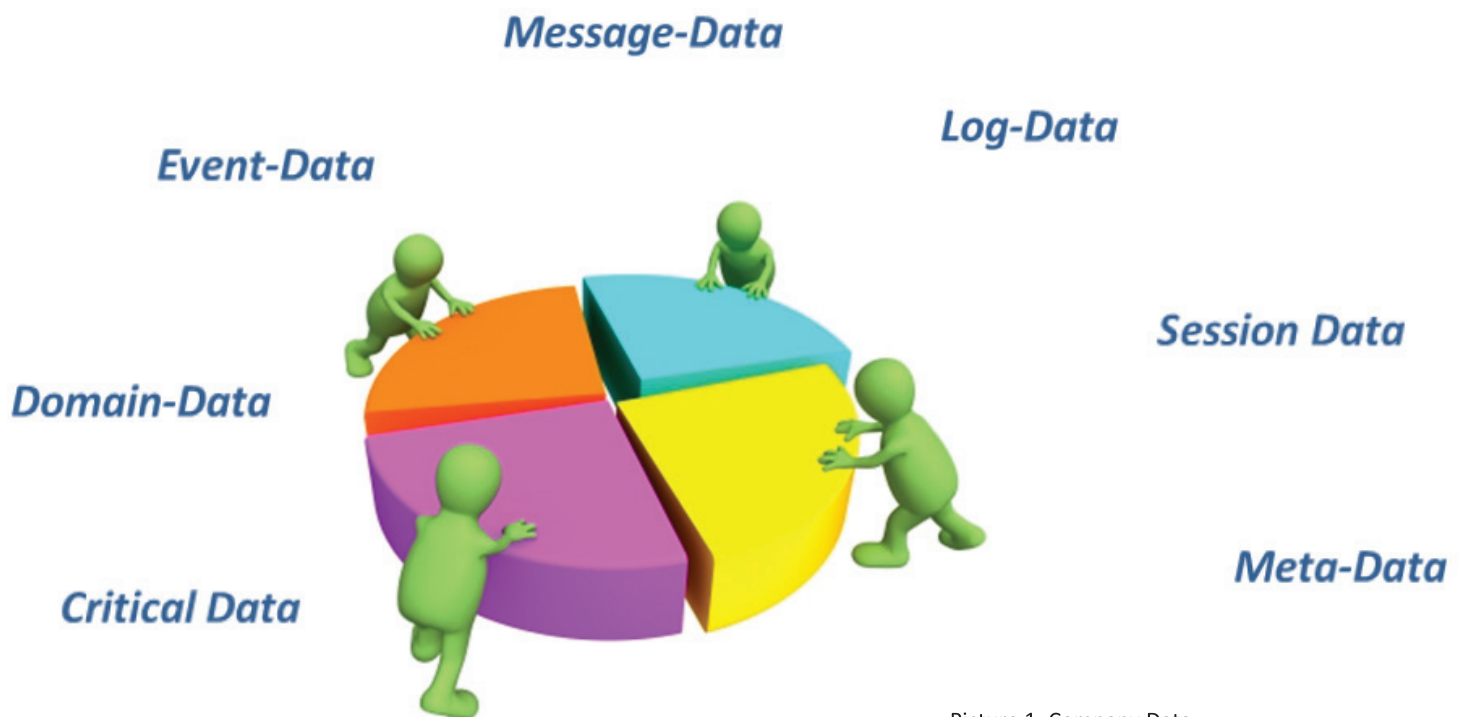
We also tried to capture persistence solutions in this area since 2009 on nosql-database.org [2]. There we do cover currently over 140 persistence solutions not belonging to the classical relational / SQL area. And indeed many companies do feel the need to reinvent their persistence approach. This gave us the unique opportunity to talk with many customers and help with consulting in Germany. And indeed many of the problems in data persistence were originated by the points mentioned above as read or write performance or schema flexibility.

The classic approach

Of course it is always essential to understand the problem and the business before being able to make any judgments on technology or a new persistence stack in general. So what we first

did was a considerable amount of database comparisons. This resulted in large tables covering databases on one axis and attributes on the other axis like: schema free, real time, performance, scaling, ring / shard, self-tuning, production tools, aggregations, queries, full text search, file management, community, APIs, support, document management and others. This served well as a first approach especially if you create spider diagrams on each persistence solution. But still we had the feeling that a universal approach is needed. Unfortunately the Internet did not reveal anything fruitful except a list of questions published in June 2011 on the highscalability blog [3] which came too late for our purposes.

Psychology



Picture 1: Company Data

The comprehensive checklist

To address this issue we developed a comprehensive checklist and identified six areas which must be covered in the analysis:

1. Data
2. Consistency / Transactions
3. Queries
4. Performance
5. Architecture
6. non-functional requirements

This all developed to around 70 questions we inserted into a printed form for customer discussions and evaluations. Let's dig a little into these six areas.

1) Data Analysis: The diversity of data in a company with different requirements has grown enormously since the last decades. As you can see in the first picture, there is a variety of different data, where several groups of data have special needs. Building clusters here and discussing needs is essential, but rarely done in companies.

Also the storage model itself and the degree of structure and

schema enforcement as shown in [4] have to be determined. Furthermore data and type constraints must be taken into account as: data navigation, data complexity, data amount (we had typically ranges from 10 GB to 1 TB here), and schema requirements / flexibility. Finally there are many questions around the persistence design itself as memory, B-Trees, SSTables, Durability, etc.

2) Consistency Model: In this area we ask questions about ACID / BASE and CAP tradeoffs in general because there are all kinds and many consistency approaches and needs. Here it's funny to see that customers always ask for ACID and all kinds of the strongest consistency even this is not needed for many parts of their business / data and sometimes already violated by caches and data replication in the company.

3) Performance: Performance has to be analyzed in its entire parts as e.g. latency, request behavior / -distribution, throughput or high concurrency demands in general.

Databases & Psychology

4) Query requirements: Of course customers must ask many questions about: typical queries, SQL requirements, BI / analytics, MapReduce, ad-hoc queries, secondary indexes, range queries, aggregations, views and so on. Hence mostly customers do insist that complex queries are mandatory. But the astonishing issue here is that in many cases a relational model had been chosen many years earlier and this often implicates complex join-like queries. But for some parts of the company data a relational model is by far not mandatory and thus complex queries might vanish with a different model.

5) Architectural issues: First you need of course to understand if the application has a local, parallel, distributed / grid, service, cloud, mobile, p2p, etc. architecture. It might also be hosted, in the cloud or on a local datacenter. But the Data Access Patterns do have the same importance:

- What's the read / write distribution?
- Do we have random or sequential access?
- What are the typical data access patterns?

6) Non-functional requirements: This is separated into A) Integration / Tools as: replication, robustness, load-balancing, partitioning, auto-scaling, text-search integration. B) real non-functional requirements as: refactoring frequency, 24/7 live add / remove, developer qualification, db simplicity (installation, configuration, development, deployment, upgrade), company restrictions, DB diversity allowed, security (authentication, authorization, validation), license model, vendor trustworthiness, community support, documentation, and many more. Then we have C) costs as: db-support, responsiveness, SLAs, general costs, scaling costs and sysadmin costs. Finally some argue that operational costs D) are most important exceeding all other categories in the long run. Meaning safety, backup / restore, crash resistance, disaster management and monitoring. This all has been captured in our NoSQL book for further reading [5].

Where psychology enters

So it looks like life would be easy with a huge catalogue. Together we understand the problem, we answer all questions in all six categories and finally we build a prototype to evaluate the solution decisions. Unfortunately it doesn't work like this and non-functional requirements are always a huge obstacle. Guess

what the most common arguments against new persistence solutions or polyglot persistence [6] are to our observation:

1. "More than one database is not manageable!"
2. "We do have a 3 year contract with a big dinosaur!"
3. (Database) Solution XY is better!" (and perhaps we are already using it).

Especially the third argument matches typical human nature. Companies are built on expert employees with specific knowledge. What we sometimes are asking is to give up their personal knowledge i.e. assets for a new solution. Why should someone agree to drop a personal knowledge advantage? It's quite natural to fight changes addressing your personal strengths.

Conclusion

A good approach analyzing and evaluating technical problems is one issue. The other side of the coin is to identify important employees with a specific background and personal ambitions. And even more to get them into the boat, get them to embrace changes and convince them of the personal benefit of a change. And this is hard work. Ideally you have to be a psychologist consultant with unlimited access and time to talk with all company employees and help them build new solutions. Unfortunately this is not always the case and thus many times the reason for suboptimal solutions then technical reasons. But we hope that the awareness for these kind of psychological and non-functional requirements increases, leading to much better solutions. Of course not only in the database / persistence space but in any area. However a persistence solution is sometimes the core, the backbone, the heart of a company and that's why barriers often look twice as high.

Acknowledgements

We wish to thank the IFAF-Berlin Institute for applied research for supporting this work.

Further reading

- [1] http://blogs.the451group.com/information_management/2011/04/15/nosql-news-what-is-beyond
- [2] <http://nosql-database.org>
- [3] <http://highscalability.com/blog/2011/6/15/101-questions-to-ask-when-considering-a-nosql-database.html>
- [4] <http://pragprog.com/magazines/2012-05/beyond-the-bit-bucket> Edlich, Friedland, Hampe, Brauer, Brückner, "NoSQL", (in German), Hanser Verlag, 2010 / 2011 (Chapter 8)
- [5] <http://www.thoughtworks.com/radar>

Simplifying critical decision making



www.systematic.com
SYSTEMATIC

Get Better Connected

Connect to
CouchDB

Connect with
Spring Integration

Connect to
Oracle

Connect to
Riak

Connect to
Datastax

Connect to
VoltDB

Connect with
Mule

Connect with
Fuse/Camel

Connect to
MongoDB

Connect with
Your Infrastructure

Connect to
Your Endpoints

C24
www.C24.biz

Integration
Objects
Get Better Connected

Try it now



Robb, iOS Developer at SoundCloud

"Some of the **smartest people** I've ever met are the people at the desk next to me"



Phil, Developer at SoundCloud

"Each line of code you write, each configuration you make, is going to **impact** a lot of people's lives"



Duana, Backend Engineer at SoundCloud

"There are seriously great **technical opportunities** if you're interested in learning"

Hear more about working at SoundCloud

Developers and engineers at SoundCloud live for the sound of a deploy and high fives from the team. We're trying to Unmute the Web. Visit our booth or hear more online if you're interested in learning more.

soundcloud.com/jobs



When it comes to managing high velocity data,

ONE SIZE DOES **NOT** FIT ALL

IMAGINE MILLIONS OF DATABASE OPERATIONS PER SECOND

- ✓ ACID transactions
- ✓ Realtime analytics
- ✓ "Five nines" availability
- ✓ Multi-level durability
- ✓ Cloud ready

voltdb.com

@voltdb

Tame your data tsunamis with VoltDB

VoltDB

The NewSQL database for high velocity applications

A Requested For The Inte

If I had a penny for every time the entertainment distribution industries went to politicians and asked them to regulate the internet in order to “save jobs”, I would already have a lifetime of activism paid for. Yet, this is one of the most stupid and counterproductive policy any government could follow.

I like to look at historic patterns. We can observe that all of this has happened before - and all of this will most likely happen again. Every time, it feels just as unique as last time. When incumbent industries are threatened by a new and disruptive technology, they will use any justification imaginable to kill it in its infancy, trying to convince legislators that their particular special interest is a public interest. It always ends badly.

You have all heard the cries of the entertainment distribution industries (which are not “creative industries” in any sense of the word, with the notable exception of accounting practices) about how vital they are to the entire economy, and how it would be reasonable that they be compensated with an amount of money exceeding the world’s entire GDP for the mere existence of LimeWire, to mention but one example.

Looking at history, as industries become threatened by new technology, they typically embrace it in public and talk passionately about its potential, but only in terms of how the new technology can support the existing and incumbent industries. Under absolutely no circumstances must the new technology be allowed to come into a position to replace the currently dominant industries.

A famous example of this is the Locomotives Act of 1865 in the United Kingdom, better known as the Red Flag Act. It was a law that limited the speed of the new so-called automobile to 2 miles per hour in urban areas, and required them to always have a crew of three: a driver, a stoker (engine operator), and a man who would walk before the automobile waving a red flag. This effectively limited the speed of the automobile to walking

speed, negating any possible inherent speed advantage of the new mode of transportation.

The car was fantastic, people would say in public, but only as long as it didn’t threaten the railroad or stagecoach industries. These very industries, it turned out much later, were behind the lobbying that led to the Red Flag Act. The fledgling automobile industry stood to make the older industries obsolete, or at least significantly smaller, which could not be permitted. Therefore, they went to Parliament and argued how tremendously important their industries were to the public good and the economy as a whole, and claimed that their special interest was a public interest. Just like the entertainment distribution industry lobby does today.

d Red Flag ernet



Rick Falkvinge,
Founder of the Swedish and first Pirate
Party



Essentially, the stagecoach and railroad industries tried to limit the permissible use of the automobile to carry people and goods the last mile to and from the stagecoach and railroad stations, and only at walking speed. That wouldn't threaten the existing dominant industries, and they could pretend in public to embrace its usefulness.

Today, the entertainment distribution industries - perhaps more accurately described as the copyright industry - pretends to embrace the Internet, but only inasmuch as they can keep operating as they always have. Any other use needs to be outlawed and criminalized, and such laws harshly enforced.

And sure enough, British Parliament agreed in its time that the stagecoach

and railroad industries were important. But the Parliament made the mistake of seeing yesterday as the present time and eternal: those industries were only important before the technology shift that the car brought, a shift which was already underway. The special laws that these industries pushed through — with emphasis on the Red Flag Act — caused the inevitable technology shift to delay in United Kingdom, and therefore, the car industry of the United Kingdom lost considerable competitive edge against its foreign competition, being ten to fifteen years late into the game.

The moral of the story that keeps repeating itself is that an industry troubled by technological advances should neither be allowed special laws nor be confused with the public interest, but

instead be permitted to die as swiftly as possible, so that new industries and new jobs can take its place. If you do the opposite and keep that industry alive with artificial respiration and repressive legislation, you not only hurt respect for the law, but also the future economy and competitive capability.

“Saving jobs” means that politicians take resources by force from vital, innovative, and competitive industries, and give those resources to ailing, failing, and obsolete industries. It's not very good policy. We shouldn't have to fight to keep the red flag mentality off the internet, but we do.

Rick Falkvinge will open GOTO Aarhus 2012 with the Monday Morning Keynote. No doubt it will be interesting and entertaining!

goto;

conference

Aarhus Sept 30 - Oct. 5, 2012



Anders Hejlsberg Dan North
Rick Falkvinge Damian Conway
Scott Hanselman Martin Fowler

Jutta Eckstein Michael Nygard Martin Thompson

Chris Anderson Adrian Cockcroft Sam Newman

Linda Rising Steve Freeman Brian Cantrill Kasper Lund

Andy Gross Gabrielle Benefield Simon Brown Steve Vinoski Randy

Bias Jez Humble Jonas Boner Dion Almaer Ben Galbraith Brian LeRoux

 Follow us on Twitter @GOTOcon

www.gotocon/aarhus-2012



Enterprise-grade switches, gateways and cloud solutions for massively scalable payment, messaging and data solutions.

Products

Open Messaging Platform

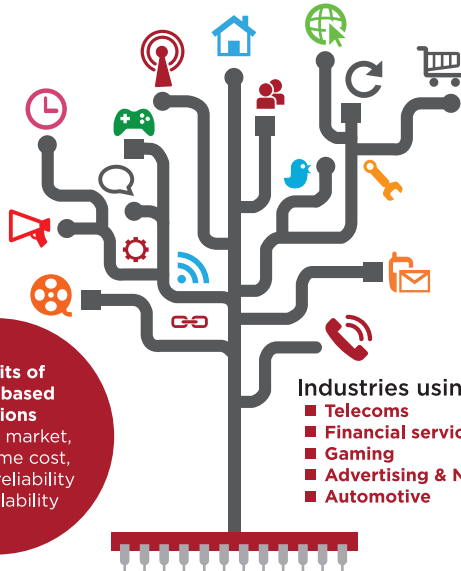
Scalable, high performance, high volume messaging solution

Buzzard

Reliable, high performance, carrier grade messaging platform supporting SMS messaging, billing and payments

Riak

Open source, highly scalable, fault-tolerant distributed database



Benefits of Erlang based solutions

Speed to market, low lifetime cost, extreme reliability and scalability

Industries using Erlang

- Telecoms
- Financial services
- Gaming
- Advertising & New Media
- Automotive

Erlang Solutions Ltd.

E: info@erlang-solutions.com T: +44 (0)20 7456 1020

W: www.erlang-solutions.com

London | Stockholm | Krakow | Copenhagen | Aarhus | Zurich

```
-module(job).
-export([start/1]).
-import(World, [developers/0]).
start(Fun) ->
  try
    [apply(Dev, [now()])
     Dev <- developers(), Fun(Dev,
                             klarna.com/career)]
  end
```

Oh really... yes...

catch? end. -> nocatch

klarna
klarna.com/career

Award-Winning Vendor of Developer Productivity Tools



jetbrains.com



GET
unstuck
WITH JBOSS
ENTERPRISE MIDDLEWARE
▶ redhat.com/jboss/getunstuck/



Copyright © 2012 Red Hat, Inc. Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, and RHCE are trademarks of Red Hat, Inc., registered in the U.S. and other countries. Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

goto; conference

INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE

- made for developers by developers...proudly presenting the best speakers & brightest attendees

www.gotocon.com



AARHUS

Training : Sept 30, Oct 4-5 // Conference : Oct 1-3, 2012



ZÜRICH

Conference : April 10-11, 2013



CHICAGO

Conference : April 23-24 // Training : April 25-26, 2013



AMSTERDAM

Conference : June 18-19 // Training : June 20, 2013



STAY IN TOUCH...

Follow us on
twitter

Sign up for newsletter
www.gotocon.com