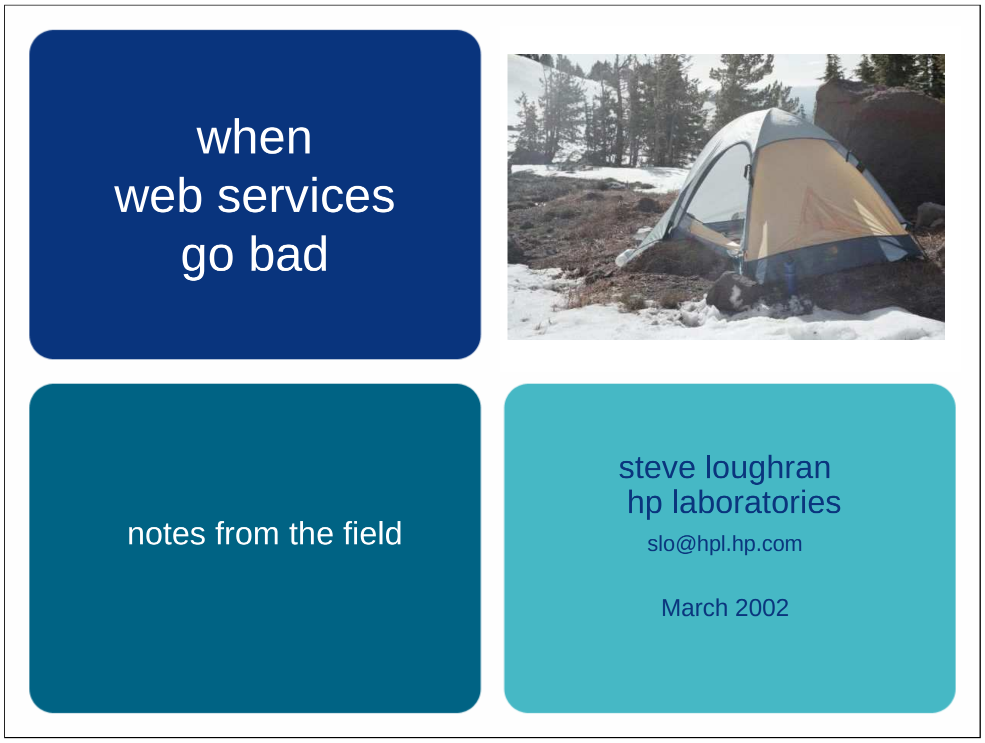


when web services go bad

notes from the field

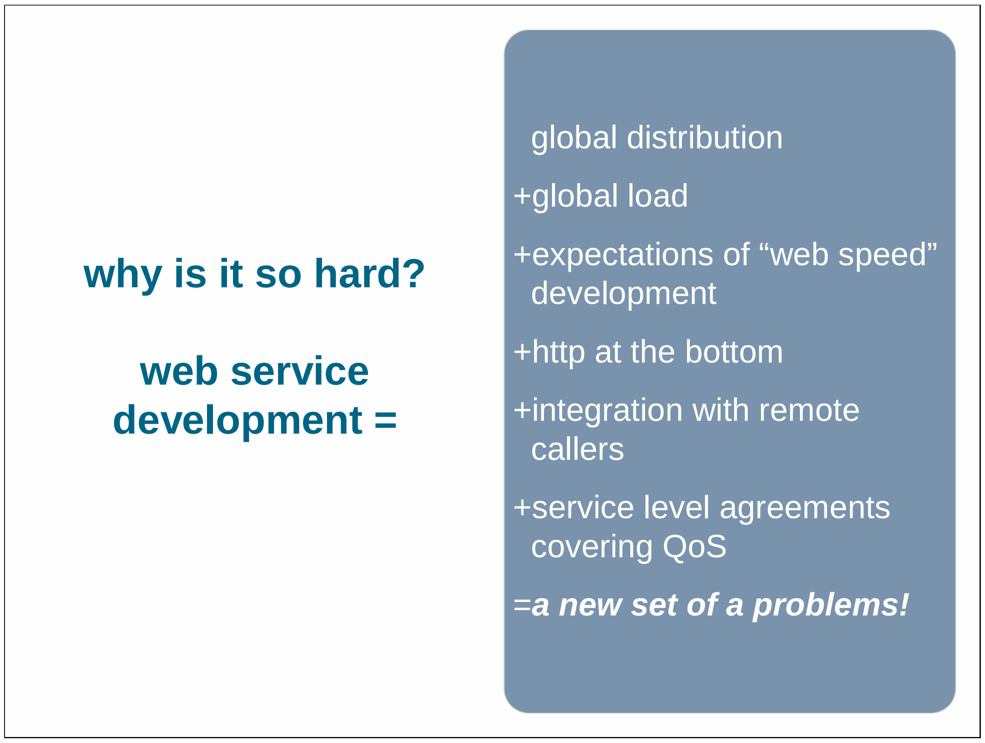
steve loughran
hp laboratories
slo@hpl.hp.com

March 2002

This is meant to conjure up the vision of some late night cable TV show “we take you behind the scenes of colocation sites, finding the worst web services in existence”, interviewing the people using them, managing them, integrating them, before finally catching up with the developer team on their doorsteps, asking them “why did you produce such a nightmare?”

If such a show existed, would you be on it? I am going to tell you how to avoid that, without getting the government to give you a new identity under the Developer Relocation Program.

Would I be on it? I would have liked to have been on it before the project was over. This is a photo of me 6000 foot up one of the cascade peaks on a technical spring mountaineering weekend, and I was getting voicemail about config problems. It would have been nice to have had a new identity then. As it is I had feign cellphone coverage failure



why is it so hard?

web service development =

global distribution

+global load

+expectations of “web speed” development

+http at the bottom

+integration with remote callers

+service level agreements covering QoS

=***a new set of a problems!***

Those who are about to deploy, we salute you!

None of the aspects of web service dev are new to the software world; talk to anyone doing telecoms software and hear about their problems -then listen to their processes for testing for three months before product releases.

What is new is that everyone is being encourage to join in, and the integration problems harder while the underlying protocols are still evolving. Previous projects like ‘a global CORBA based application with dynamic service location’ was clearly left to the experts (and even then success was a bit hit and miss)

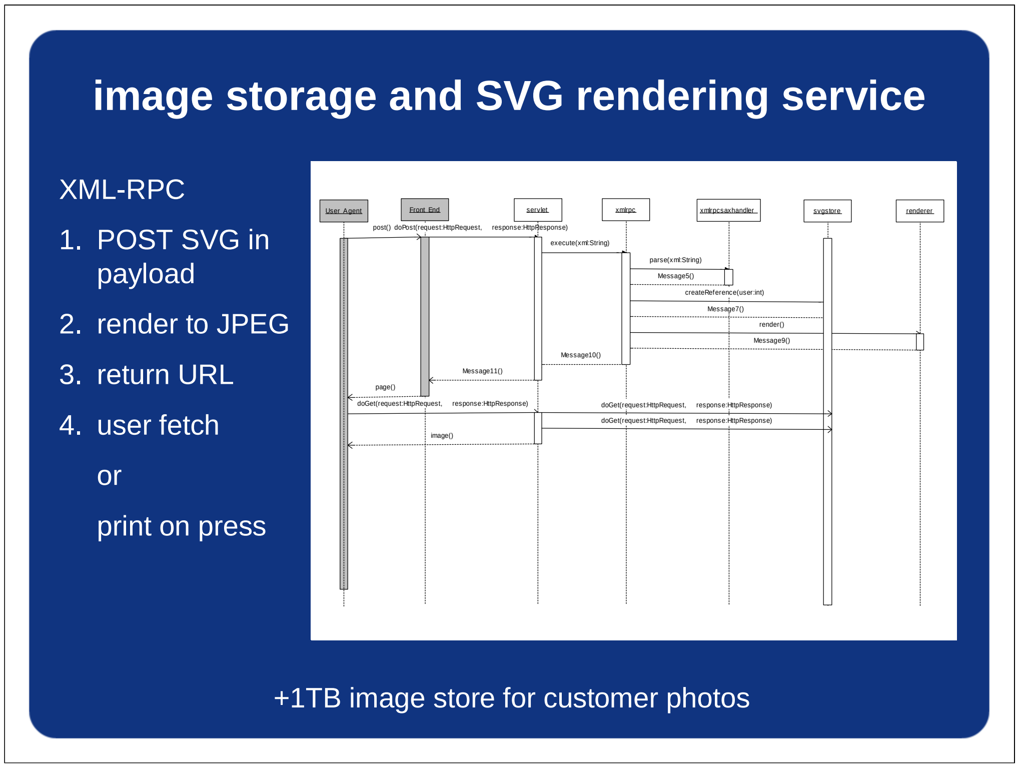


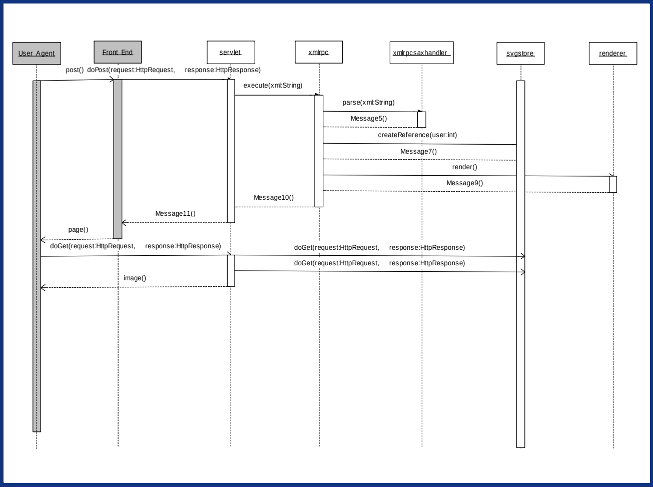
image storage and SVG rendering service

XML-RPC

1. POST SVG in payload
2. render to JPEG
3. return URL
4. user fetch

   or

   print on press


User Agent
Front End
servlet
xmlrpc
xmlrpcsaxhandler
svgstore
renderer
post()
doPost(request:HttpRequest, response:HttpResponse)
execute(xml:String)
parse(xml:String)
Message5()
createReference(user:int)
Message7()
render()
Message9()
Message10()
Message11()
page()
doGet(request:HttpRequest, response:HttpResponse)
doGet(request:HttpRequest, response:HttpResponse)
doGet(request:HttpRequest, response:HttpResponse)
image()

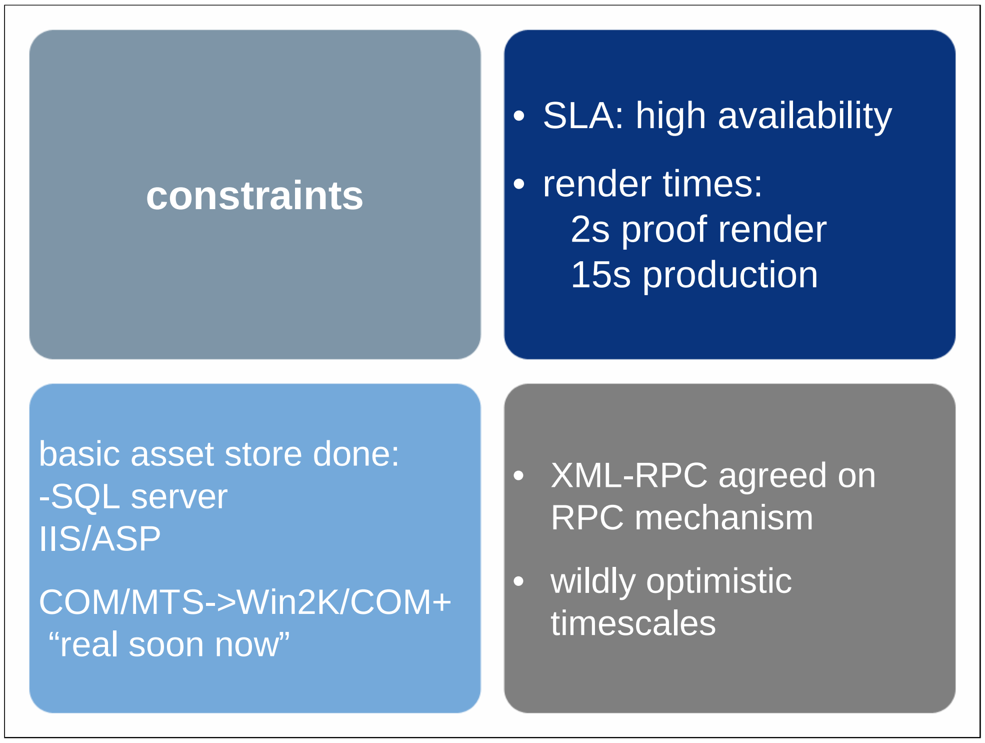
+1TB image store for customer photos

So here is what we did as a service.

It is almost your definitive “easiest RPC mechanism ever” example – idempotent RPC calls, one method “render this XML for that person”, returning either a URL or an error string.

As far as implementation goes, this is “my first use case”class of RUP, which is good as we used “my first UML modeler”, visio, to host the design and crank out the framework. Easy Huh?

This a) made it nearly viable on the timescales marketing made up at random and b) meant all the problems we had were more related to deployment than RPC stuff.



constraints

- SLA: high availability
- render times:
  2s proof render
  15s production

basic asset store done:
-SQL server
IIS/ASP

COM/MTS->Win2K/COM+
"real soon now"

- XML-RPC agreed on RPC mechanism
- wildly optimistic timescales

Here are some of things that we had to meet to satisfy the customer.

-High Availability is universal; you want to use a web service you expect it there all the time. We had the basics from the colocation service of twin power plants and the like; it was up to us to ruin their availability promises.

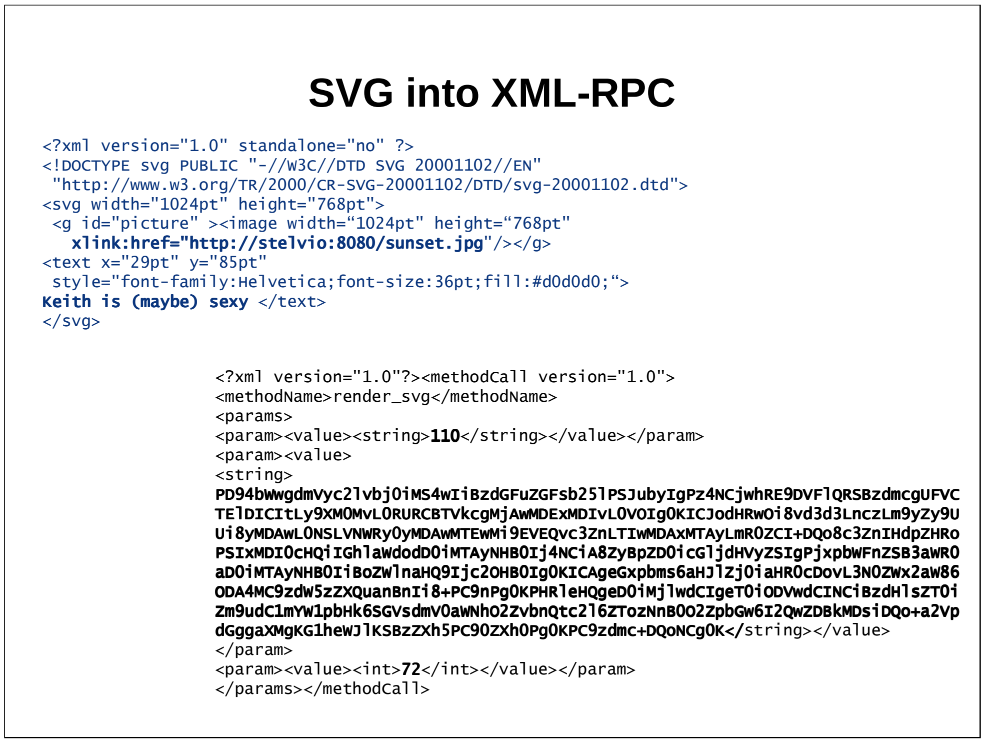
-The render times are an interesting one. SVG can be used for insanely complex graphics; the 2 sec is for the expected use (photo album) and under 'normal' load, not peak. Even then, it was a tough one.

We were given the basic asset store "proven technology" from another part of the company who had built it and put the deal together. This probably created more issues than anything else. By the time we realised what a mess it was, it was too late to fix.

XML RPC was the already agreed on mechanism. This had two main flaws for us –you cant pass XML inside it unencoded, and the exception specification is very weak. But it worked and took two days to implement, which cant be bad.

Timescales. Team convened feb, first deliverable march, early june for live, second phase late Aug. This could have been viable if the asset store was robust enough. Actual timescales: live in August, phase II in Jan '02…partially because phase I worked so well there was no need to upgrade, also because Xmas was the busy time…

The customer is actually a 'strategic account', which means that if they aren't happy their CEO calls our CEO, and voicemails trickle down to us. This used to cause a lot of grief, but now we can all say what project we worked on and anyone up the entire management chain takes a step back and goes "oh, that project". At the time it meant that if we screwed anything up, everyone got to know of it.



SVG into XML-RPC

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
 "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
<svg width="1024pt" height="768pt">
 <g id="picture" ><image width="1024pt" height="768pt"
   xlink:href="http://stelvio:8080/sunset.jpg"/></g>
<text x="29pt" y="85pt"
 style="font-family:Helvetica;font-size:36pt;fill:#d0d0d0;">
Keith is (maybe) sexy </text>
</svg>
```

```
<?xml version="1.0"?><methodCall version="1.0">
<methodName>render_svg</methodName>
<params>
<param><value><string>110</string></value></param>
<param><value>
<string>
PD94bWwgdmVyc2lvbj0iMS4wIiBzdGFuZGFsb25lPSJubyIgPz4NCjwhRE9DVFlQRSBzdmcgUFVC
TElDICItLy9XM0MvL0RURCBTVkcgMjAwMDExMDIvL0VOIg0KICJodHRwOi8vd3d3LnczLm9yZy9U
Ui8yMDAwL0NSLVNWRy0yMDAwMTEwMi9EVEQvc3ZnLTIwMDAxMTAyLmR0ZCI+DQo8c3ZnIHdpZHRo
PSIxMDI0cHQiIGhlaWdodD0iMTAyNHB0Ij4NCiA8ZyBpZD0icGljdHVyZSIgPjxpbWFnZSB3aWR0
aD0iMTAyNHB0IiBoZWlnaHQ9Ijc2OHB0Ig0KICAgeGxpbms6aHJlZj0iaHR0cDovL3N0ZWx2aW86
ODA4MC9zdW5zZXQuanBnIi8+PC9nPg0KPHRleHQgeD0iMjlwdCIgeT0iODVwdCINCiBzdHlsZT0i
Zm9udC1mYW1pbHk6SGVsdmV0aWNhO2ZvbnQtc2l6ZTozNnB0O2ZpbGw6I2QwZDBkMDsiDQo+a2Vp
dGggaXMgKG1heWJlKSBzZXh5PC90ZXh0Pg0KPC9zdmc+DQoNCg0K</string></value>
</param>
<param><value><int>72</int></value></param>
</params></methodCall>
```

This is the SVG generated by the caller; linking to images and containing text to be rendered. This is a much more simple example than production SVG, those compose whole pages by merge many pictures using alpha blending, proprietary fonts, etc. I think they do the designs with some template mechanism from a database, filling it in with content from the current session.

To make this a request we have to base 64 encode it; then put it in the xml-rpc message. we need to encode it because xml-rpc isnt namespace aware, and we need to preserve the whitespace.

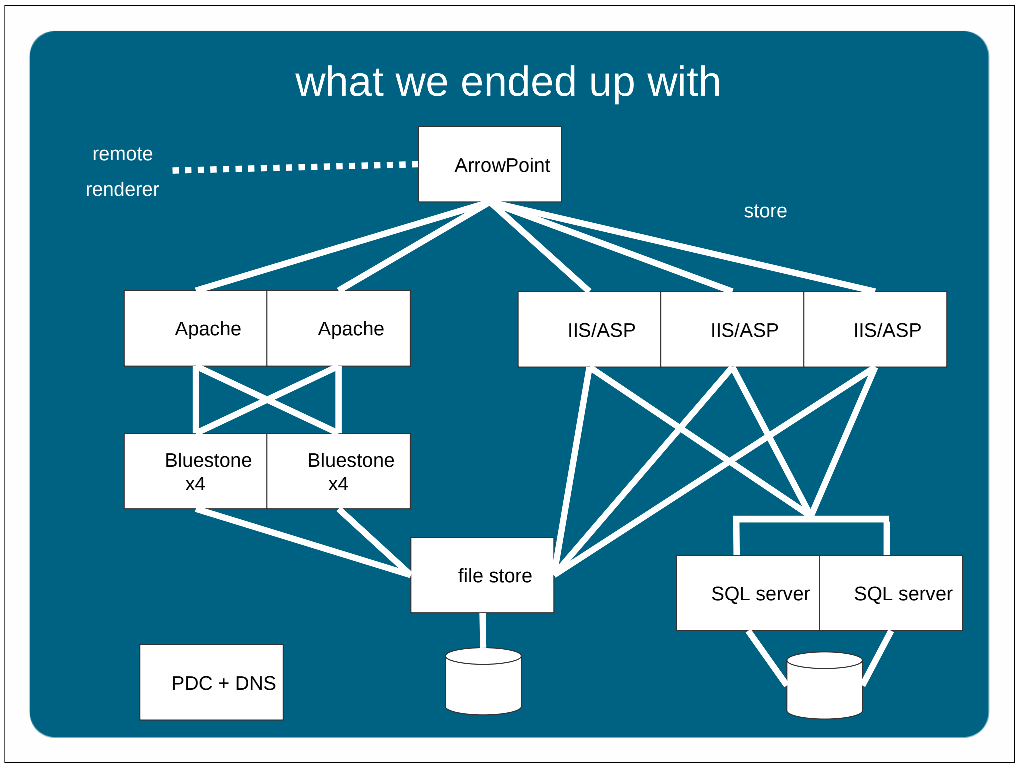
Response

```
<?xml version="1.0"?>
<methodResponse>
<params>
<param><value><string>2453</string></value></param>
<param><value><string>
http://stelvio:8080/hpsvgservlet/fetch?ref=_2_ec5132765b_4553.jpg-110
</string></value>
</param>
</params>
</methodResponse>
```

The server comes back with the rendering time, and the URL to where the image can be retrieved (usually a FQDN, I had to configure the laptop to return a shortname so that it works while roaming).

This image can only be retrieved by a user with a cookie containing an (encoded) user ID of 110, and then only for an hour or so.

Who is keith and why is he (maybe) sexy? It’s Keith Ballinger, Microsoft, and he says so, so it must be true.



what we ended up with

If you look at what we have built, we have the old with the new: ASP on one side, java on the other. But under that java code is a win2K box, and we have a COM object doing the work for us. Redundancy gives availablility, but adds to the system complexity.

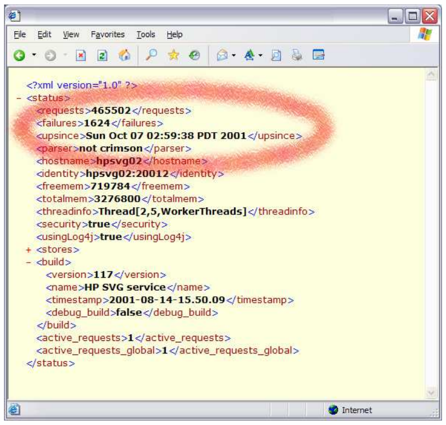
Why didn't we use C#/.Net? Risk: this was in Feb 2001 , with deploy in June/July; If we had used .NET it would have probably added instability and hence work on the render side, but it would have benefited the legacy side, as they would have had a better migration path, adding new ASPX pages and slowly replacing the existing code. That would ultimately have given a unified system on both sides. Or we could have used java across the board, but this would have hit our incremental dev process badly.

NB: Although it looks like the two sides are independent, there are a couple of ways they talk to each other. One is cookie auth; the java app decrypts cookies for both sides, the other is that we cache previewed images served up by the ASP page and fetch them via the file store.

The Apache front ends are extraneous. Even if they are 'cheap boxes with no security issues', they remove half the value of the arrowpoint and add installation/setup delay.

If load increases the rendering back end could probably scale to 8-10 boxes before you had to worry about backbone traffic, I don't know about the storage side of things. Not shown: remote rendering site accessing the same content; it is 1500 miles away next to a printing press

what worked?



```
  <?xml version="1.0" ?>
- <status>
    <requests>465502</requests>
    <failures>1624</failures>
    <upsince>Sun Oct 07 02:59:38 PDT 2001</upsince>
    <parser>not crimson</parser>
    <hostname>hpsvg02</hostname>
    <identity>hpsvg02:20012</identity>
    <freemem>719784</freemem>
    <totalmem>3276800</totalmem>
    <threadinfo>Thread[2,5,WorkerThreads]</threadinfo>
    <security>true</security>
    <usingLog4j>true</usingLog4j>
  + <stores>
  - <build>
      <version>117</version>
      <name>HP SVG service</name>
      <timestamp>2001-08-14-15.50.09</timestamp>
      <debug_build>false</debug_build>
    </build>
    <active_requests>1</active_requests>
    <active_requests_global>1</active_requests_global>
  </status>
```

Web Service Model!

✓XML based RPC

✓HTTP for transport

✓SVG for request

✓ANT build & deploy

✓JUnit for unit tests

✓incremental development

Before we get into where things went wrong, lets look at some of the things we did which worked. Overall, the whole system works: here is a screenshot from 19/feb showing that we have been running this JVM (about 1/8 of the SVG system) non stop since early October: no reboots, no breaks, no crashes. slick. if you look closely the load looks a bit low, but those are all serious processing requests with a reasonable sell-through at the end; each req runs a the CPU flat out for 1-2 seconds at the best of times.

So what works. Obviously a language very resistant to leakage is one key. The others are

-XML as the marshalling for RPC, worked nice and simply, especially XML. XML is easy to parse, easy to handle the message, easy-ish to generate a response.
XML RPC is limited; no metadata other than the API spec; very limited in format of fault responses

-bug reporting; we'll get to that in a mo'

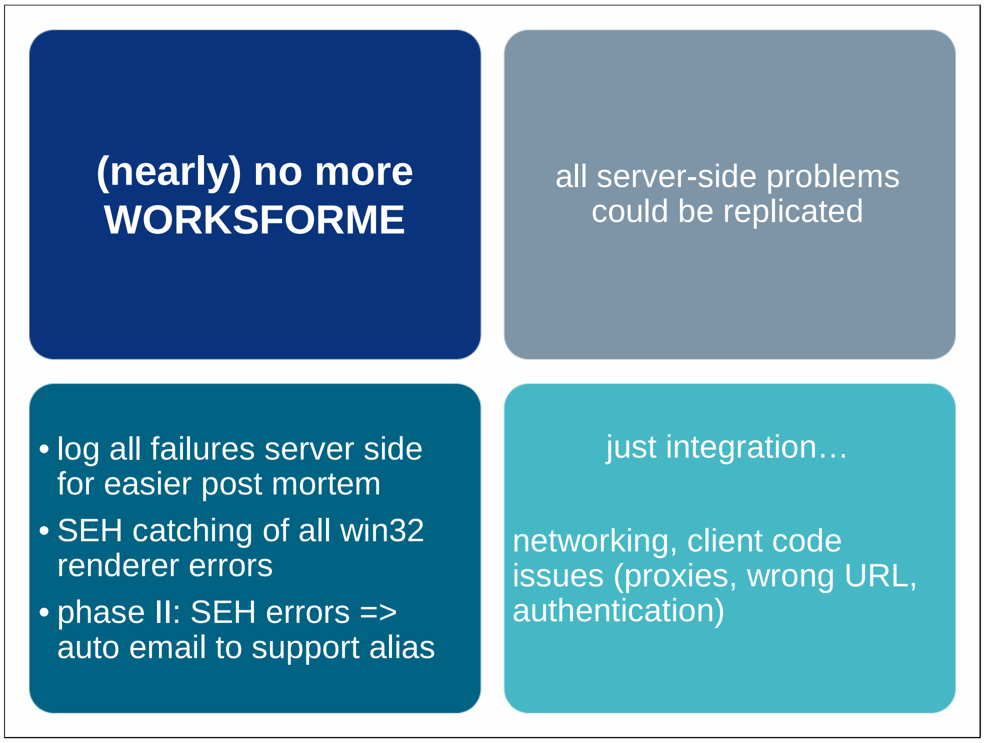
-Ant. of course

-JUnit. hand in hand with Ant

-Log4J. best logging tool in Java I've come across

-COM integration via exec(). may seem ugly, works nicely. no leakage or stability issues, see.

-incremental development of a web service. I have some issues with incremental deployment, but incremental dev went ok.



(nearly) no more WORKSFORME

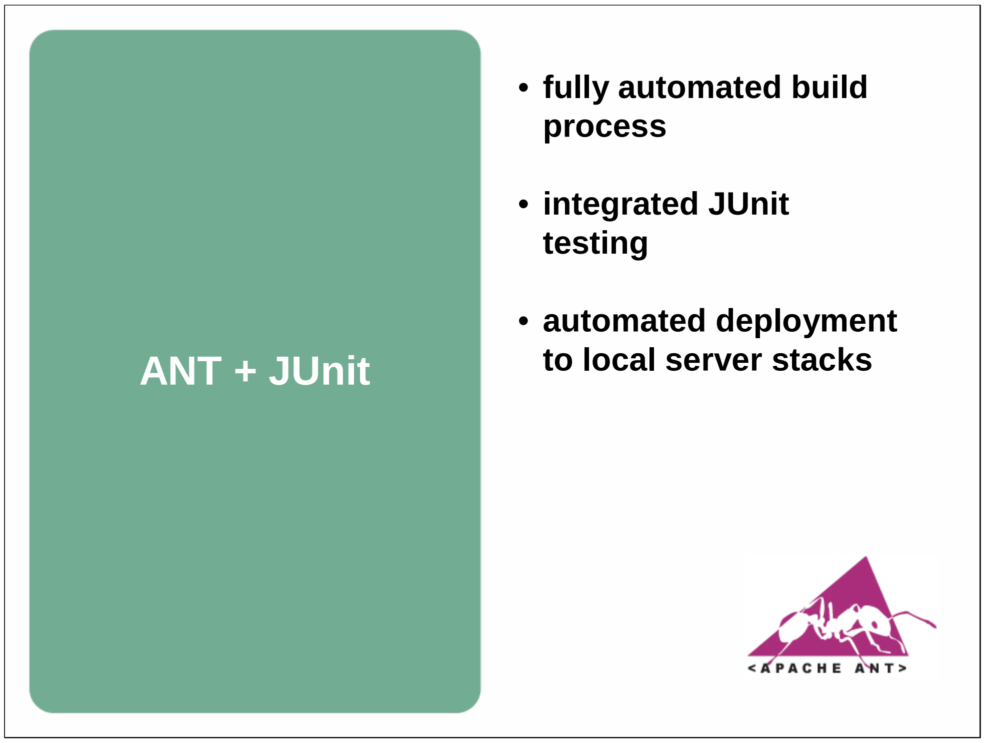
all server-side problems could be replicated

- log all failures server side for easier post mortem
- SEH catching of all win32 renderer errors
- phase II: SEH errors => auto email to support alias

just integration…

networking, client code issues (proxies, wrong URL, authentication)

1. Web services are great for replication of bug reports; unless it is a network error or caller coding, WORKSFORME is not how you get rid of bugs, provided they give you all the data
2. Auto-email of failures. We were getting the odd GPF with the COM object, so we wrote an SEH handler to catch them and return the error code, and save to log, even being devious and ignoring the error if it happened after the rendering was successful
3. Then we added auto-email ourselves with the payload that caused the problem. This lets you get the complete data to replicate the bug, and you can be fixing it before anyone even reports it. So when they do report the bug a few days later, you can appear to be more responsive, but really you have just had a longer lead time without anyone hassling you for fixes



ANT + JUnit

- **fully automated build process**
- **integrated JUnit testing**
- **automated deployment to local server stacks**

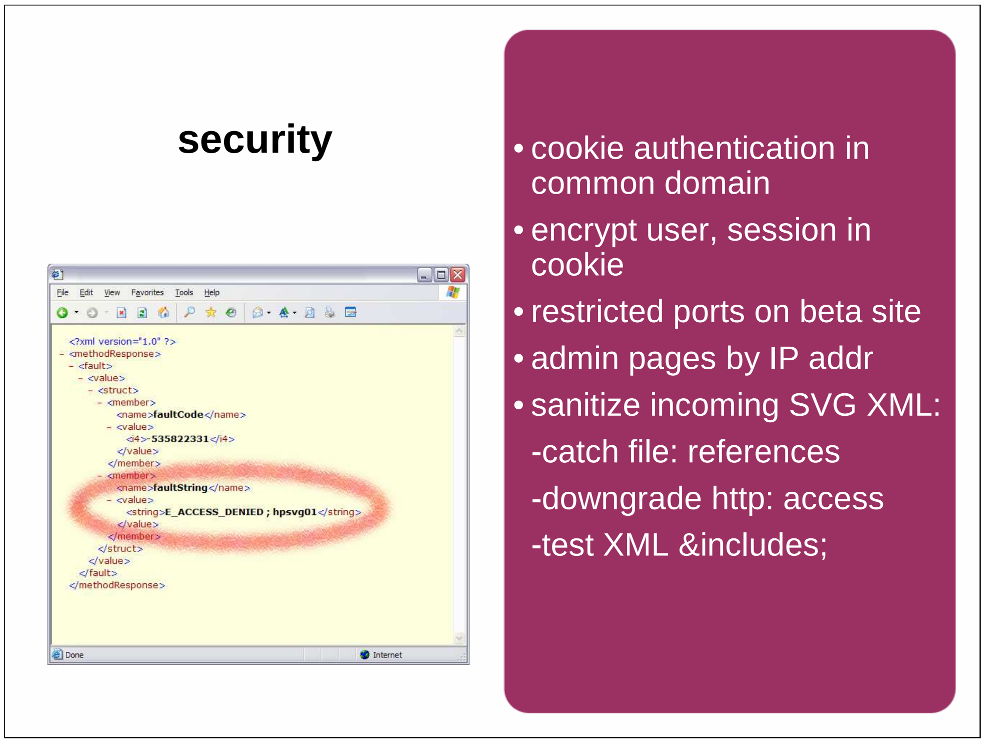
We managed to push the envelope in what Ant could do; I'd been using the tool since early '00, but this build process of compile-test-remote deploy for the service was pretty serious, as was the win32 build and the CD disk image process.

Deployment is where it was slick; FTP up of files, remote install including a process of archiving files as they were installed.

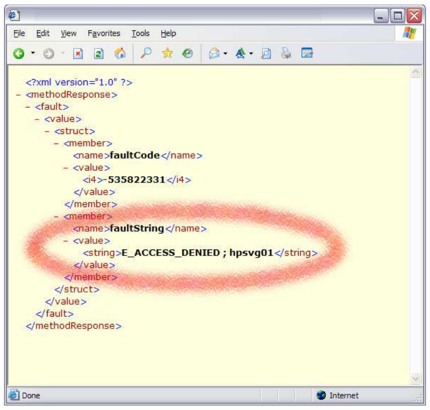
JUnit was cool for unit testing; we never go into the advanced cactus/http unit stuff because the test suite team chose an alternate design, based on database storage of tests in a more declarative syntax, also with storage in the DB of expected result images. This worked well, with some issues (the DB only had one set of expected results; you really need a different set of expected results for every app/render version. Next time I'd push httpunit more)

Also, Log4j: you want a configurable, scalable, lightweight logging system?

For .NET folk, Ant does C# compiling, though it could be improved; Even if you build with Studio.Net, there is nothing to stop you using Ant for the automated build and deploy. There is NUnit and Log4net on sourceforge for you too.



security



```
  <?xml version="1.0" ?>
- <methodResponse>
  - <fault>
    - <value>
      - <struct>
        - <member>
            <name>faultCode</name>
          - <value>
              <i4>-535822331</i4>
            </value>
          </member>
        - <member>
            <name>faultString</name>
          - <value>
              <string>E_ACCESS_DENIED ; hpsvg01</string>
            </value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

- cookie authentication in common domain
- encrypt user, session in cookie
- restricted ports on beta site
- admin pages by IP addr
- sanitize incoming SVG XML:

  -catch file: references

  -downgrade http: access

  -test XML &includes;

Because we lacked WSDL, there is no way to determine the RPC API short of packet sniffing, and a little bit of VPN keeps the #of RPC packets going over the Internet unencrypted down to a minimum. So it is hard for anyone to hack the system without inside knowledge. The challenge was to make it hard for anyone with inside knowledge of the system or web server to get a toehold.

This is the stuff we didn't tell ops everything about. If they knew that at first you could render any file in the HDD to a JPEG, they would have been unhappy. But fair is fair, they never told us they had turned off the port restrictions on the beta site till a week later, so we had to suddenly implement filtered security (not easy in servlet 2.1).

We also needed to be sure that no XML inclusion or file ref mechanism could get text into our doc; I think we've done it, but just to be safe, we run in a downgraded user *and* with Java security manager restricting rights further.

The issue we have here is the perennial defence in depth problem: you can make any point secure, but can you make all points secure?

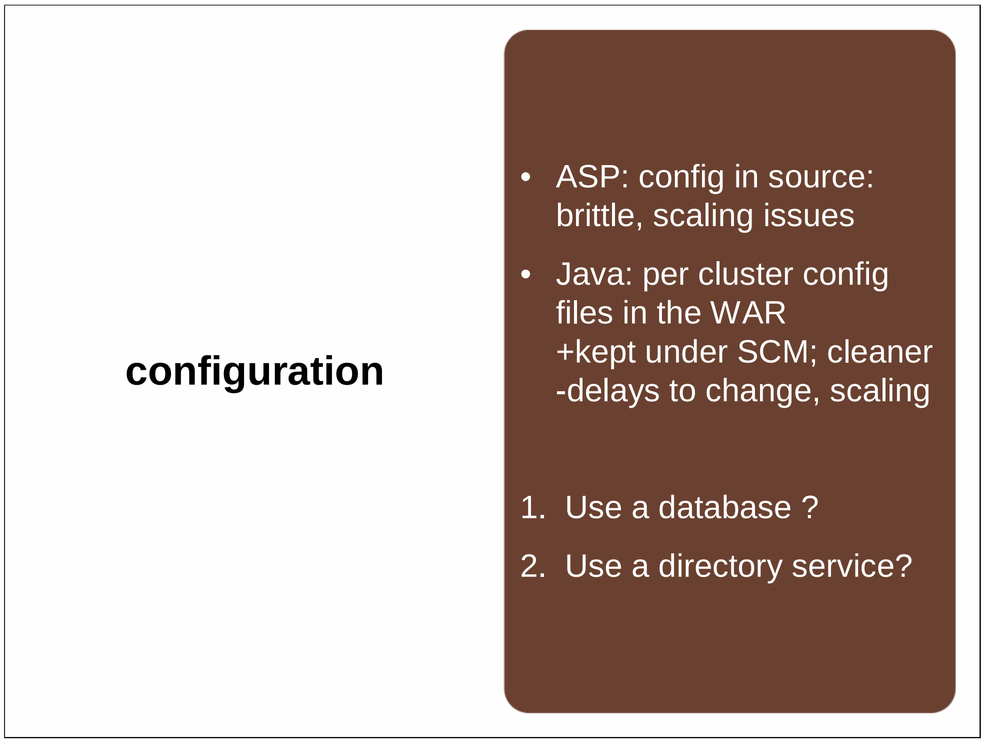
-back door access to WAR file,

-DoS attack by making 200 render requests then breaking the connections. Dunno why, we think we fixed it

-without explicit http filters (servlets 2.2), all JSP pages are a potential security hole.

-Lots of cookie cross authentication issues. Domain sharing involves too many people, we had to exchange code to get it all working, etc. This took both teams on the phone to debug.

If you look at weak points because we don't track originating IP addr, you could steal someone's session and see their pictures. But you cant track IP addr when the end users are coming to you from somewhere like AOL where it can choose a different proxy on an apparent whim.



configuration

- ASP: config in source: brittle, scaling issues
- Java: per cluster config files in the WAR
  +kept under SCM; cleaner
  -delays to change, scaling

1. Use a database ?
2. Use a directory service?

Our project manager preferred the VB approach; one CD for all systems, their problem to customise for different boxes. Ops hated that, it was way too brittle, and they ended up giving staging and production the same machine names to minimise differences. The Java side approach was different: app determined machine name, read a file there which contained config data -later just the name of the cluster config file containing the real data. We could configure systems and give them a disk which was nominally locked down.

problems with this: need to know target names, IP addresses before building the disk. If anything does need changing it can be done server side, but the changes need to feed back to the client. And it doesn't scale well either.

A lot of people have used database configuration, that is the ideal method when you have a DB always to hand. But I want to propose an alternative, using an LDAP directory server

+replication for scaling and availability

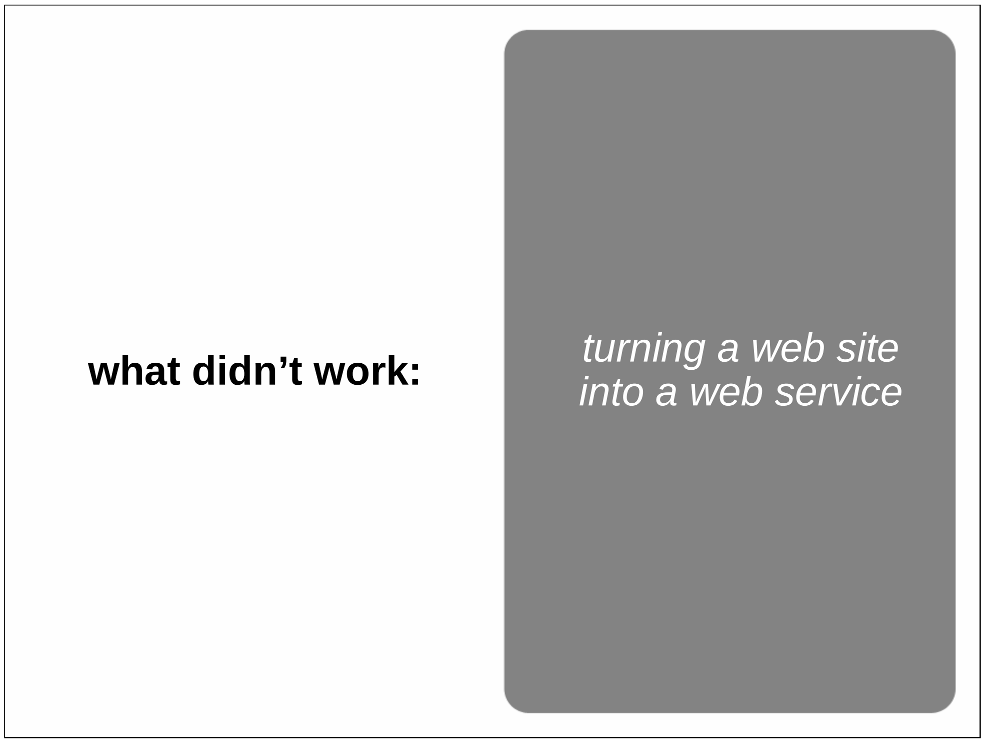
+comes free with a PDC

+cross platform, language

+good heirarchical config model

+OK editors.

I haven't used this enough to be aware of the drawbacks other than it can be fiddly to talk to, and the 'bootstrap' problem of a cluster and a server still exists -what is the LDAP url; how does it first get populated. Plus how do you stop ops from making changes…



what didn’t work:

turning a web site into a web service

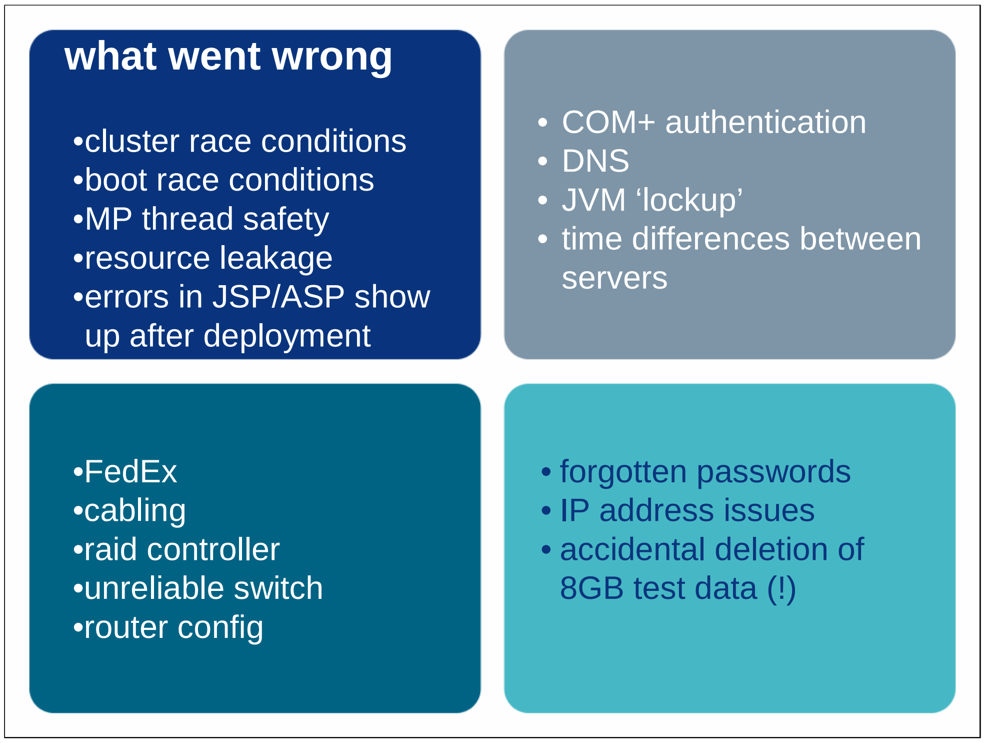
Do not try this at home.

This was the other half of the problem, getting the system which existed already out the door.

Do not attempt to take an existing web site, so leaky that every server had to be rebooted every four hours, delivering HA through many servers, with no test suite and no automated install process, and try and turn it into a profitable web service.

Its not just the functionality, which was there, it was the difficulty in installing, running and keeping running the system that nearly killed the team. Getting this beast stable want so much an SEI-CMM level 1 process “individual heroics” as level 0: “individual heroics are insufficient”.

Summary: it was the operational aspect of the system, not the pure use-case functionality which was the issue.



what went wrong

- cluster race conditions
- boot race conditions
- MP thread safety
- resource leakage
- errors in JSP/ASP show up after deployment

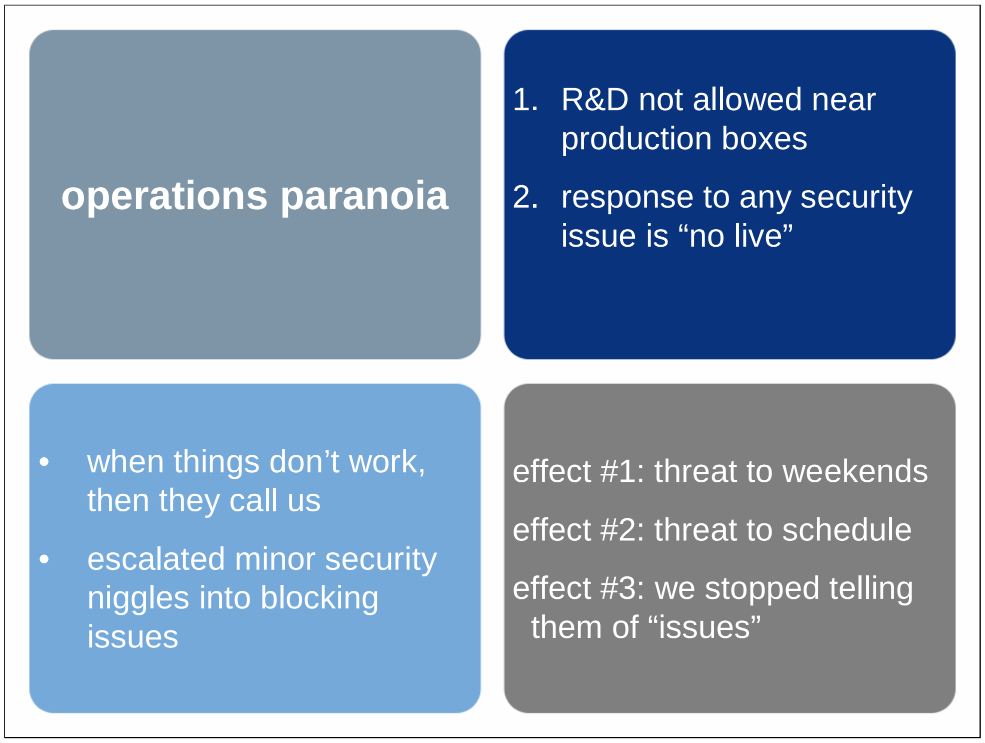
- COM+ authentication
- DNS
- JVM ‘lockup’
- time differences between servers

- FedEx
- cabling
- raid controller
- unreliable switch
- router config

- forgotten passwords
- IP address issues
- accidental deletion of 8GB test data (!)

Some of our problems, split into code, config, process and hardware

I wont go into the details on most of these, except to say they range from low level hardware -cables, RAID- through code -threading and leakage, to system DNS. The 8GB test data deletion was a process failure, somehow the web page which could clean up the system was visited the day before the live date and the data deleted. This proved that ops are good at backup and retrieval purposes as it only took a couple of hours to recover. Needless to say, that web page was soon removed.



operations paranoia

1. R&D not allowed near production boxes
2. response to any security issue is “no live”

- when things don’t work, then they call us
- escalated minor security niggles into blocking issues

effect #1: threat to weekends

effect #2: threat to schedule

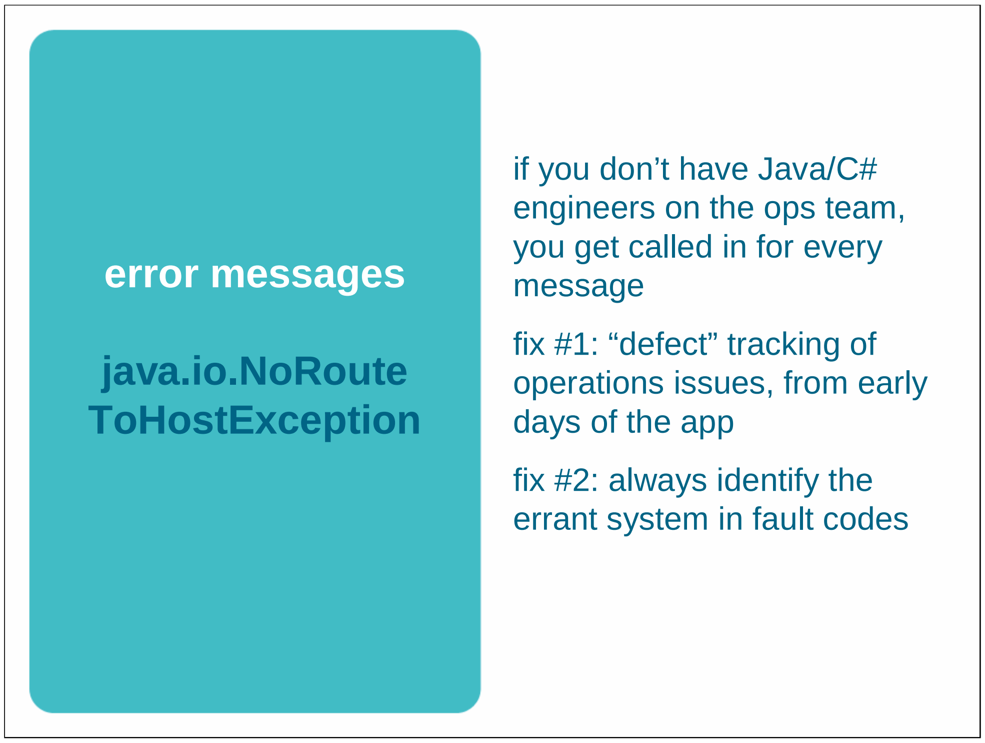
effect #3: we stopped telling them of “issues”

Being in operations must be like writing device drivers: you never get credit, only blame. Nobody says: ‘flaky graphics card, the device driver compensates for the card and OS quirks nicely’. Same for web service ops; nobody goes ‘awful web service, great operations’, even when they struggle valiantly to meet the 7x24 requirements of the contract.

It seems to me that our operations team’s processes were unduly paranoid. This is good for a secure system, but their paranoia seemed to exemplify itself in a complete distrust of engineering. Now they were right to mistrust our code, but not to the extent of treating us as malicious.

All this did was delay things and ruin our weekends, in one case I got called back on the first day of my one vacation during the project, only to find the problem wasn’t my code, it was a file permissions issue somewhere. Now if it ops had more experience in debugging this stuff, rather than just installing it and escalating they would be better at fixing problems. But how do you achieve that on a deadline? Early involvement? But they were so busy with the legacy part of the project that Java side got neglected by virtue of being “relatively” easy.

Another example: I found a way of getting access to other peoples files. I fixed this, but I never told them about the problem or the fix because they’d have taken the server off line, even during the beta test phase, putting schedules at risk. So we rolled out the fix, added the tests and never mentioned it. Which is dangerous: I’d have been happier them knowing, and not overreacting.



error messages

java.io.NoRoute ToHostException

if you don’t have Java/C# engineers on the ops team, you get called in for every message

fix #1: “defect” tracking of operations issues, from early days of the app

fix #2: always identify the errant system in fault codes

This is a funny. One day DNS went down, error messages were going into the log “java.io.NoRouteToHostException’, the kind of thing a developer would recognise as network trouble. But ops assumed that any java message was trouble with the java app, so their response was to keep rebooting -which seemed to fix it, but only because the error was intermittent.

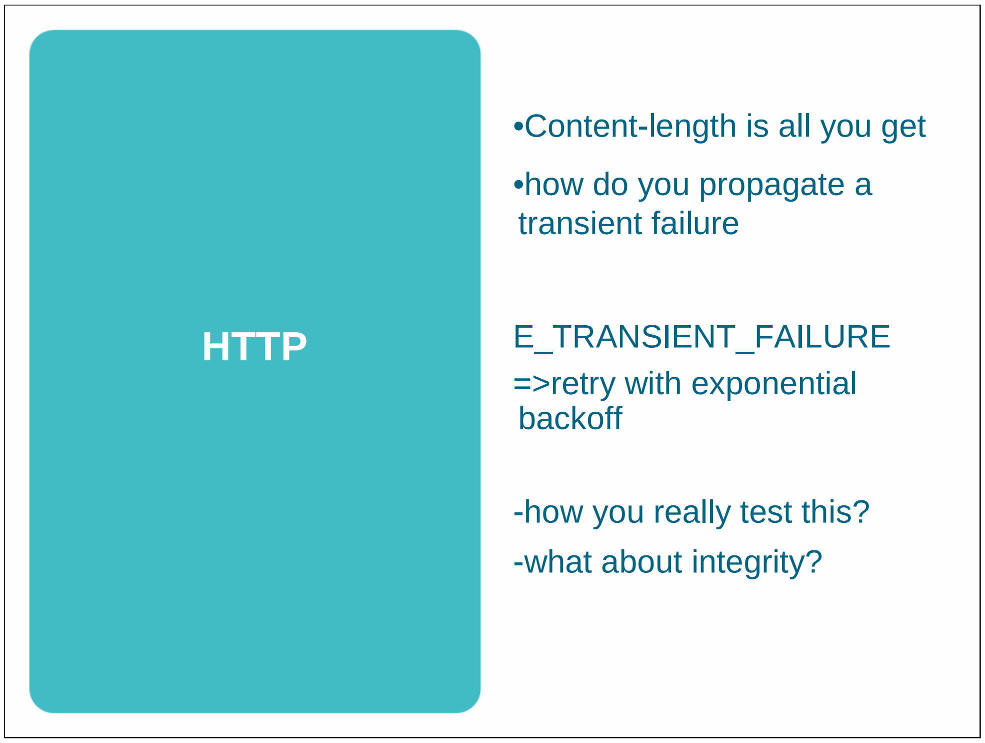
Can you guess who it escalated to in the end? That’s right –engineering. But the problem was that the colocation folk were doing scheduled DNS maintenance…

Fixes.

-bugzilla like tracking of error messages to causes; start doing this early on in development, then ops get a operations defect tracking DB, just like a code defect tracking DB.

After the DNS came back, one JVM had stale DNS info, so kept on failing. It looked like an intermittent failure after the router round robin called it, so we were getting ‘there is an intermittent failure’ bug, not ‘instance #3 keeps failing’

-XML RPC requests to include the identity of the server, lets recipients help collect data for you.



HTTP

•Content-length is all you get

•how do you propagate a transient failure

E_TRANSIENT_FAILURE
=>retry with exponential backoff

-how you really test this?

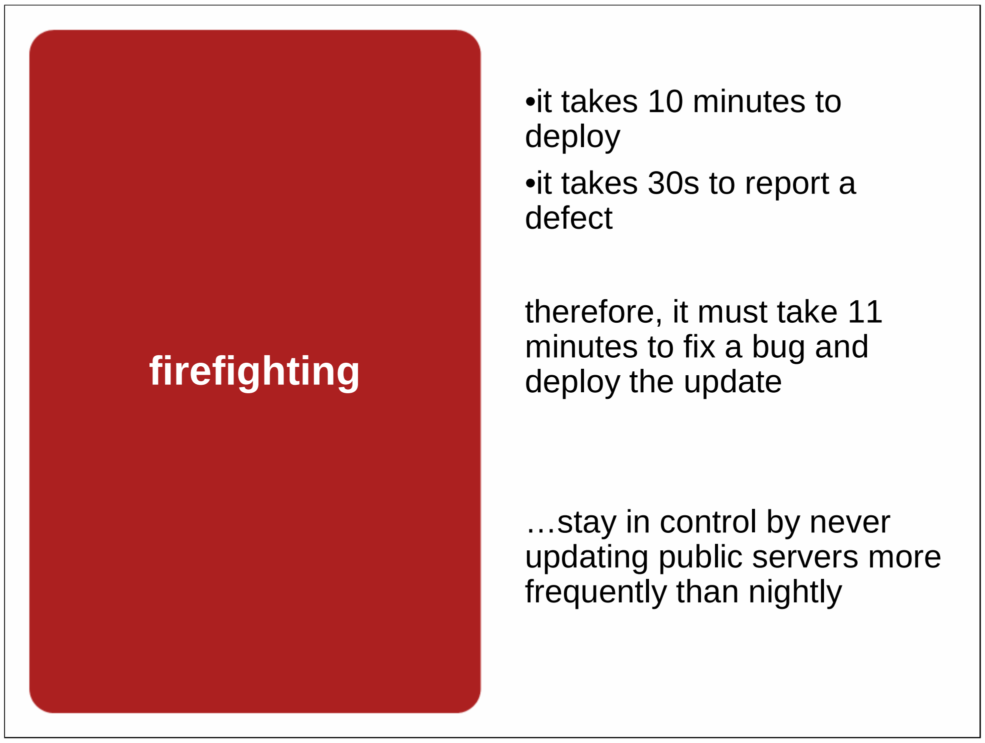
-what about integrity?

This is the only major defect that only hit us after production…the print renderer out on the east coast somewhere got an image containing incomplete sub images.

We had to make sure everything sent content length stuff, and handled it, and propagate trouble upstream

propagating trouble is tricky; we had to change the specs on the RPC to indicate that there was an error which could be addressed by retrying

even then, we cant deal with data corruption: what do we do with transposition? We don’t care right now, but other people will have to.



firefighting

- it takes 10 minutes to deploy
- it takes 30s to report a defect

therefore, it must take 11 minutes to fix a bug and deploy the update

…stay in control by never updating public servers more frequently than nightly

This made may and june somewhat hectic. It wasn't that the workload was particularly high (we got home at weekends), but that there was no control over the process. You'd come in at 10am in the morning and voicemail and email boxes would be full of issues, and there was an expectation from the customer that we could have a fix out in a snap. Management picked up the same impression and would continually pressurize us for progress and status.

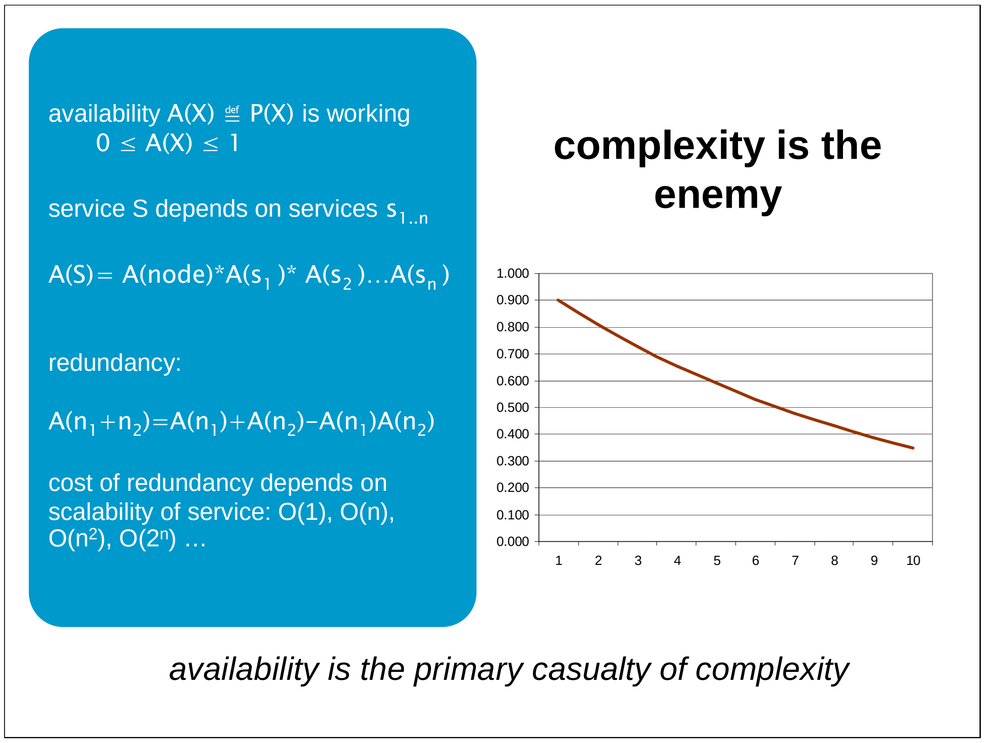
We probably created this impression by rolling out a few simple fixes to blocking issues in a rush; setting expectations which were impossible to sustain over time and with harder bugs.

The effects was to stop us prioritizing which bugs to fix, in which order; we were being expected to fix things as fast as they came in, and complete the new features for the next release of the system.

I don't think holding back early releases would have helped, but here are my tactics for future work

-let the customers have direct access to the defect tracker (we now run one outside HP for exactly this purpose)

-restrict the rate of change of the server to max one update a day; maybe a lower rate (once every 3, 5 or 7 days). Cite mechanical deployment processes "It's scheduled overnight update" rather than "we include delays to keep your expectations low"



complexity is the enemy

availability $A(X) \overset{\text{def}}{=} P(X)$ is working
$0 \leq A(X) \leq 1$

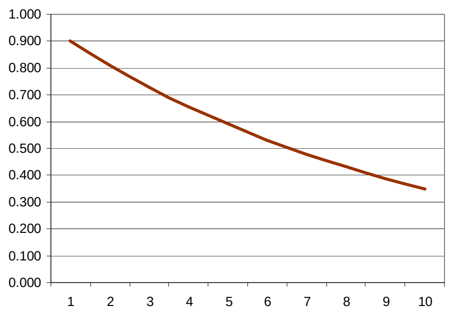
service S depends on services $s_{1..n}$

$A(S) = A(node) * A(s_1) * A(s_2) \ldots A(s_n)$

redundancy:

$A(n_1 + n_2) = A(n_1) + A(n_2) - A(n_1)A(n_2)$

cost of redundancy depends on scalability of service: O(1), O(n), O(n^2), O(2^n) …



availability is the primary casualty of complexity

I now want to scare people with a bit of mathematics.

This is the one page of theory; probably one of the hacker laws, which if it isn't already known I claim as mine, Steve's "why your web service is hosed" law.

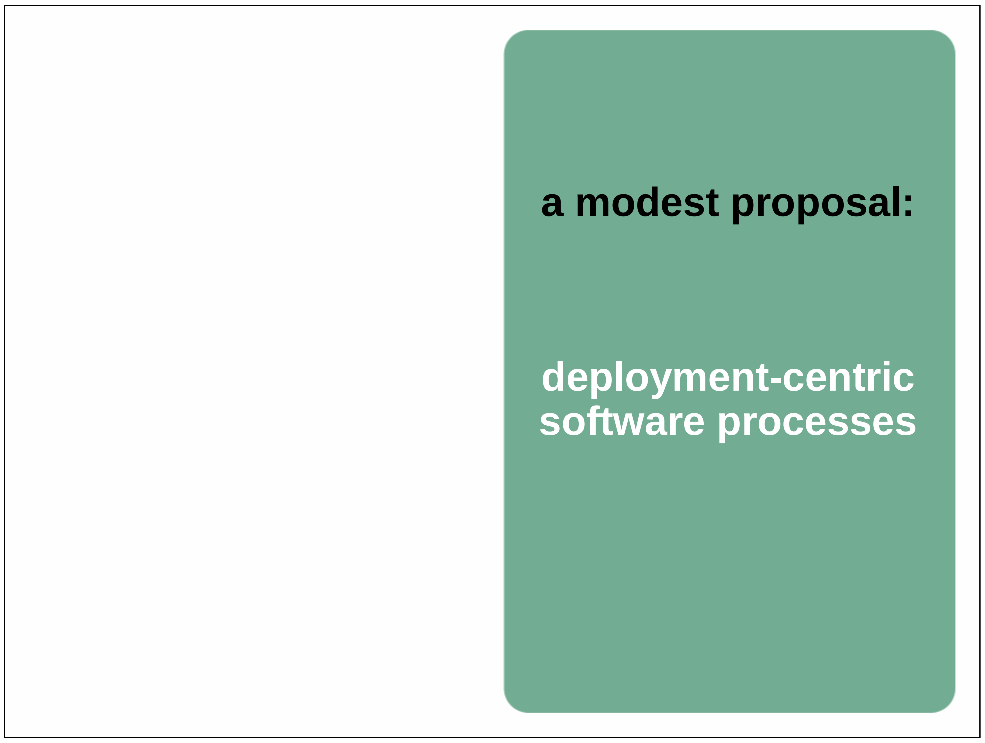
It means that your latency gradually goes up the more services you depend on, but your availability goes down the tubes, as it is the product of the availabilities of all the services, whose availability is the product of their availabilities, and so on. Redundancy is the normal way to increase reliability, but it doesn't automatically guarantee 100% availability, not if the underlying system is flaky. So you add more boxes, and then you have even more installation and operational grief.

Even if you think you have no dependencies, what about routers, DNS, a PDC, IIS, a database, a file store, the network in general, …etc. You do need some of this stuff, but if you are self sufficient and robust (local host table as well as a DNS), then you are in control.

Strip out everything you don't need. Anything you are only loosely coupled to via an async messaging service, is off the critical path of availability. And keep your installation as simple as you can be.

I have left out any equation on integration I(X) is the effort to integrate with a component. You'd think it is a simple sum equation, but it never seems like that, probably because debugging a problem increases exponentially with complexity, and it is defect location that is a key problem in integration.

Also, installation is O(n), tracking down problems probably O(n*n)



a modest proposal:

deployment-centric software processes

A good web services is one that works so well you forget where it is.

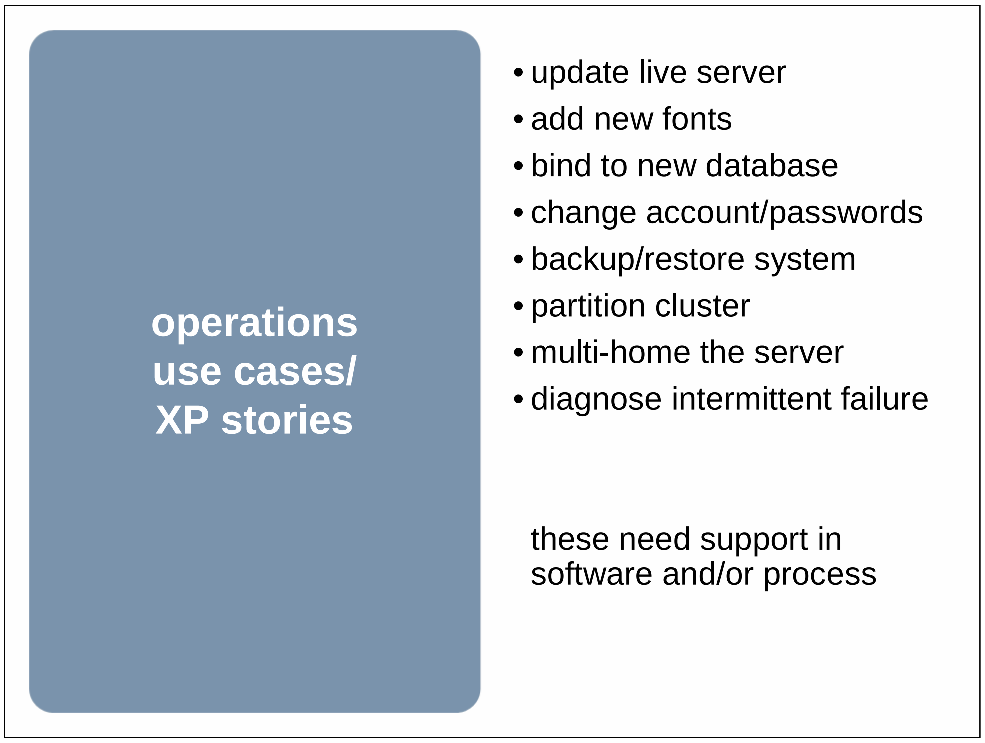
You look at RUP and it has this nice little evolutionary loop and then the software is finished and you just hand it off to get used somehow - it reverts to waterfall right at the end.

Turn to XP and you have minimal upfront design, just evolutionary coding with a rigorous test framework. But that assumes that all design errors can be fixed by a tractable amount of coding, where tractable means ‘in the time you have left’. if you have written everything with an autoincrementing Int32 as the object ID in the DB, and you want to partition the DB into two, so you need a GUID instead, you have a very serious problem that coding cant always fix without breaking your clients.

Both are flawed for a service where availability and ease of operations make the difference between success and failure.

The RUP needs to add deployment/operations as a discipline to work on during the cycle [c.f Scott Ambler: The Enterprise Unified Process, http://www.ronin-intl.com/publications/unifiedProcess.htm].

Both RUP and XP need to incorporate the needs of operations into the requirements they use to design and build their software



operations use cases/ XP stories

- update live server
- add new fonts
- bind to new database
- change account/passwords
- backup/restore system
- partition cluster
- multi-home the server
- diagnose intermittent failure

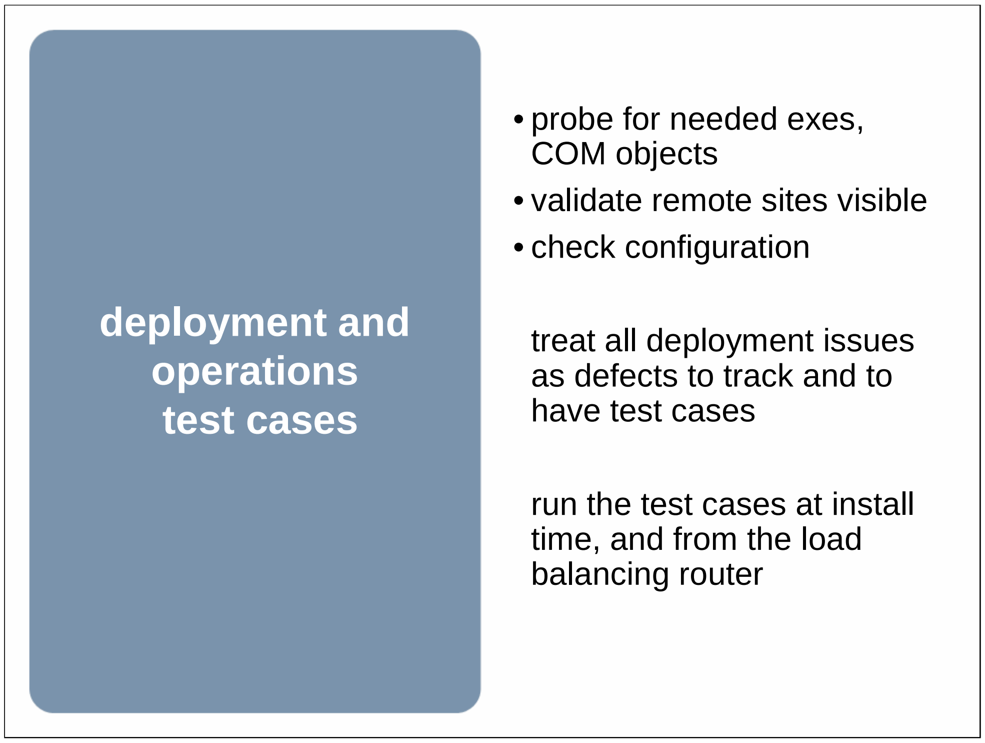
these need support in software and/or process

Operations tasks need use cases (RUP), stories (XP).

Talk to ops, find out the things they need to do to a system. Many are standard for all boxes; for us ‘install new fonts’ turned out to be a common task specific to our service.

The more automated support for these tasks are, the lower the operational cost of the system, the happier ops are with you.

Some use cases are very fundamental: multi-homing; partitioning. Better think about those early on.



deployment and operations test cases

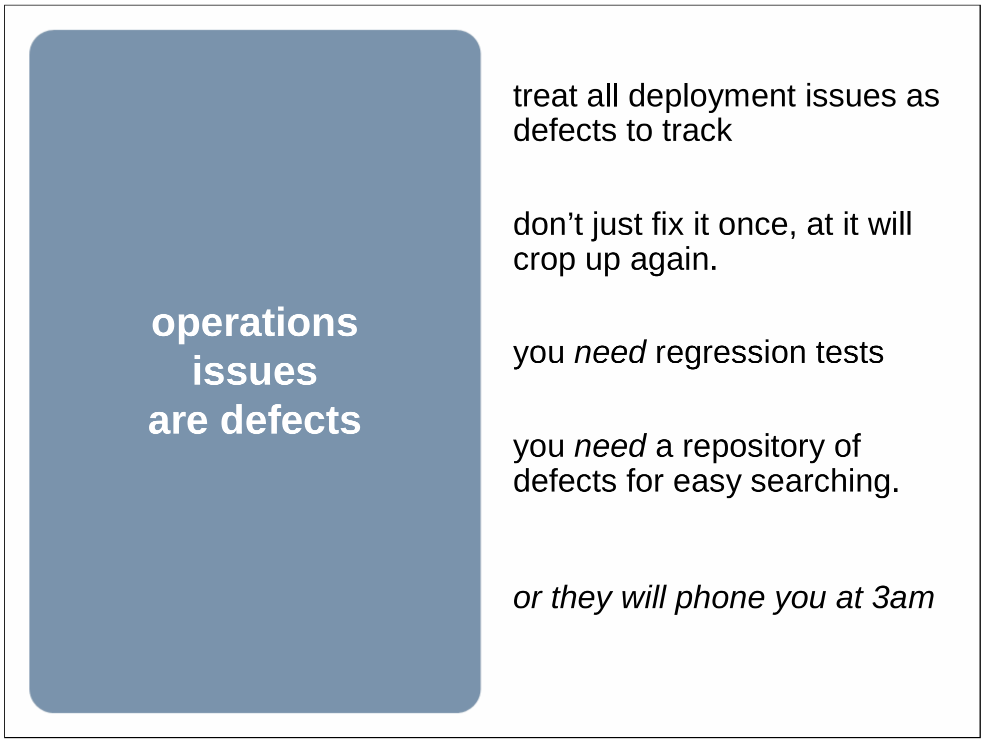
- probe for needed exes, COM objects
- validate remote sites visible
- check configuration

treat all deployment issues as defects to track and to have test cases

run the test cases at install time, and from the load balancing router

If you are going to have stories in XP, you need tests for them.

Anything which breaks the remote system needs a test case. To get to some of our problems, like the clock, we now probe for that on server startup and log an error. Same for access to remote apps, local bits we need. In fact we have a test case which renders something in all the expected fonts and validates it against a cached bitmap.



operations issues are defects

treat all deployment issues as defects to track

don’t just fix it once, at it will crop up again.

you *need* regression tests

you *need* a repository of defects for easy searching.

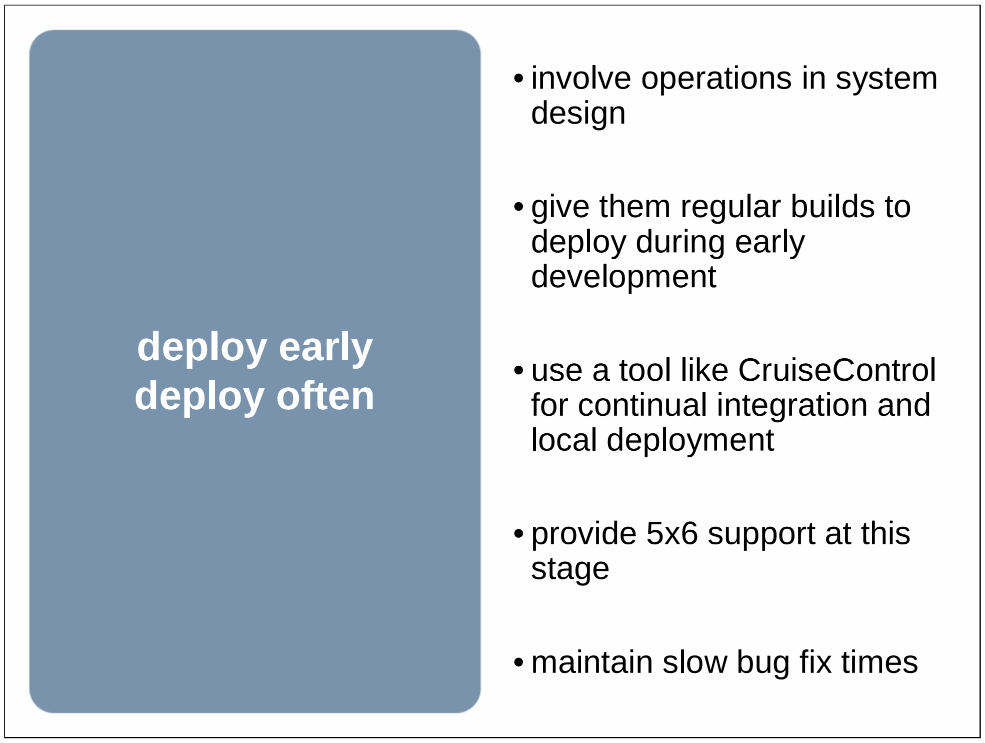
or they will phone you at 3am

It aint enough to fix a config issue, because management and ops seem to have a shorter memory than you (they don’t, they just have more things to worry about and understand the system less well, not having brought up so many installations as you developers have)

Every recurrent config problem ‘clock was wrong’, ‘hostname set up wrong’, ‘someone gave me the wrong IP addr’ needs logging with symptoms of the problem.

the next time the problem arises, they can search the db for the symptoms and see possible causes which have happened in the past, and then maybe fix them without escalating to you.

For this archive to be ready in time for deployment, you need to start filling in the defects from the beginning of the project, which means as the system is first being brought up.



deploy early deploy often

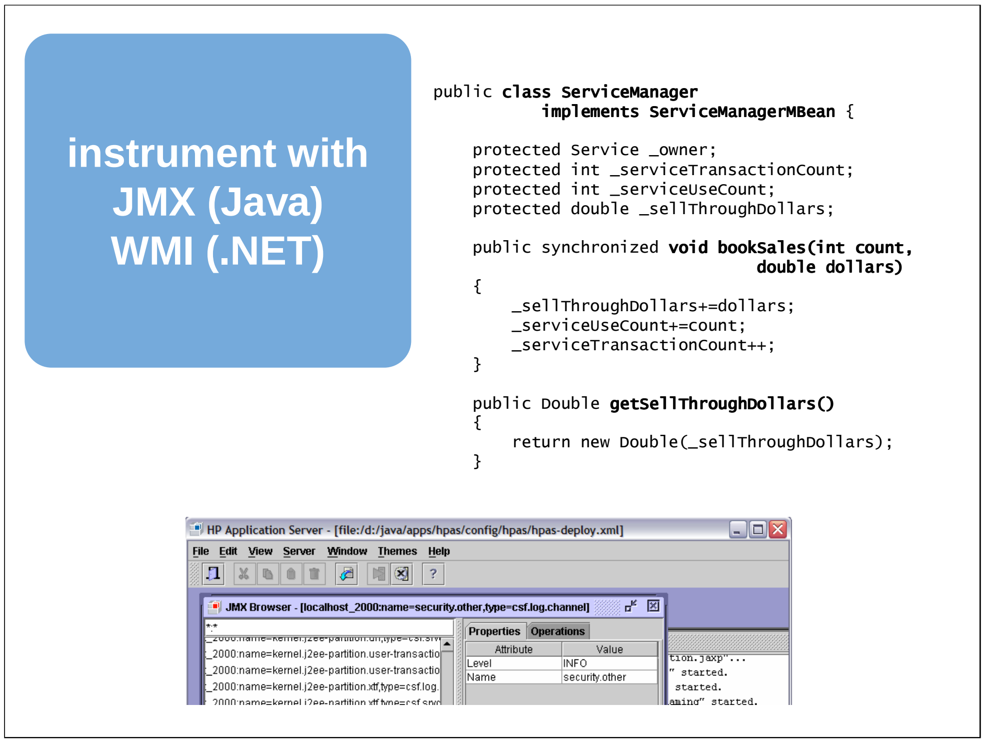
- involve operations in system design
- give them regular builds to deploy during early development
- use a tool like CruiseControl for continual integration and local deployment
- provide 5x6 support at this stage
- maintain slow bug fix times

Ops need practice in deploying; you can give it to them by offloading the task to them early and getting the process honed. Yes, that means listening to their problems and making it easier to deploy.

I am also going to point Java developers at CruiseControl (sourceforge) and the GUMP (jakarta.apache.org) for automated build and unit testing; both of these can be extended to auto-deploy after the tests are passed.

Engineers may work funny hours, but that does not mean that they are on call: keep the SLA minimal, working days, working hours maximum, preferably something less. Customers may complain but emphasize that they are trading off support coverage for final product go live dates.

likewise, suspend your morals. even if it is a five minute fix always take 24 hours to roll out the update, because you need to keep their expectations low enough to handle the ‘oops, we need to rewrite half the system’ defects.



instrument with JMX (Java) WMI (.NET)

```
public class ServiceManager
           implements ServiceManagerMBean {

    protected Service _owner;
    protected int _serviceTransactionCount;
    protected int _serviceUseCount;
    protected double _sellThroughDollars;

    public synchronized void bookSales(int count,
                                 double dollars)
    {
        _sellThroughDollars+=dollars;
        _serviceUseCount+=count;
        _serviceTransactionCount++;
    }

    public Double getSellThroughDollars()
    {
        return new Double(_sellThroughDollars);
    }
```

HP Application Server - [file:/d:/java/apps/hpas/config/hpas/hpas-deploy.xml]

File Edit View Server Window Themes Help

JMX Browser - [localhost_2000:name=security.other,type=csf.log.channel]

```
*:*
_2000:name=kernel.j2ee-partition.user-transactio
_2000:name=kernel.j2ee-partition.user-transactio
_2000:name=kernel.j2ee-partition.xtf,type=csf.log.
```

Properties | Operations

Attribute	Value
Level	INFO
Name	security.other

```
tion.jaxp"...
" started.
 started.
aming" started.
```

We put off learning JMX because of schedule pressure, but when we had time to look we realized it was easy.

-You write an interface ending in MBean; which uses simple well known object types rather than proprietary objects or the low level datatypes (Double over double)

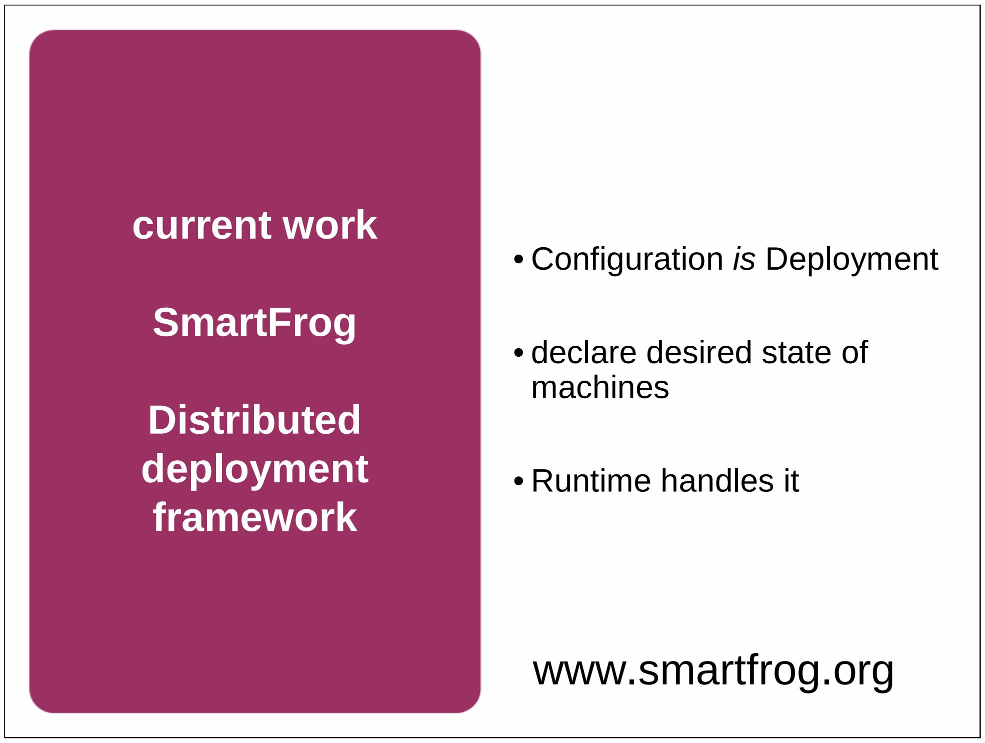
-provide an implementation

-implement any in-app entry points -like bookSales()

-at startup register the app with the manager singleton, bind your instances to objects it needs to work with.

The only issue is how to use this stuff on your server without fancy management products like HP OpenView? Well, the HP Bluestone Server *is* a JMX manager built in, so you can do all this stuff locally and let ops sort out their side of things. The nice thing about JMX is that with the right tools it scales: you can see the whole cluster at a glance.

Coding this stuff has always been fiddly, but there is <xdoclet> support now



current work

SmartFrog

Distributed deployment framework

- Configuration *is* Deployment
- declare desired state of machines
- Runtime handles it

www.smartfrog.org

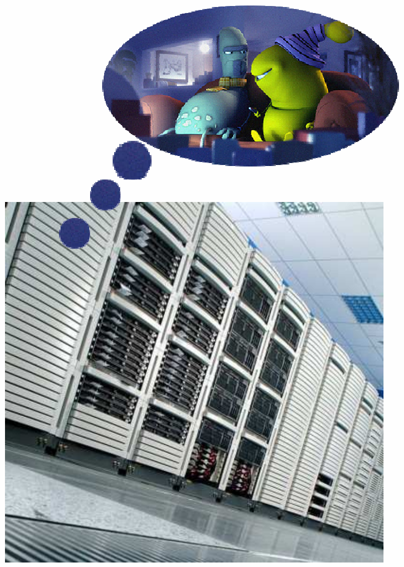
HPLabs research

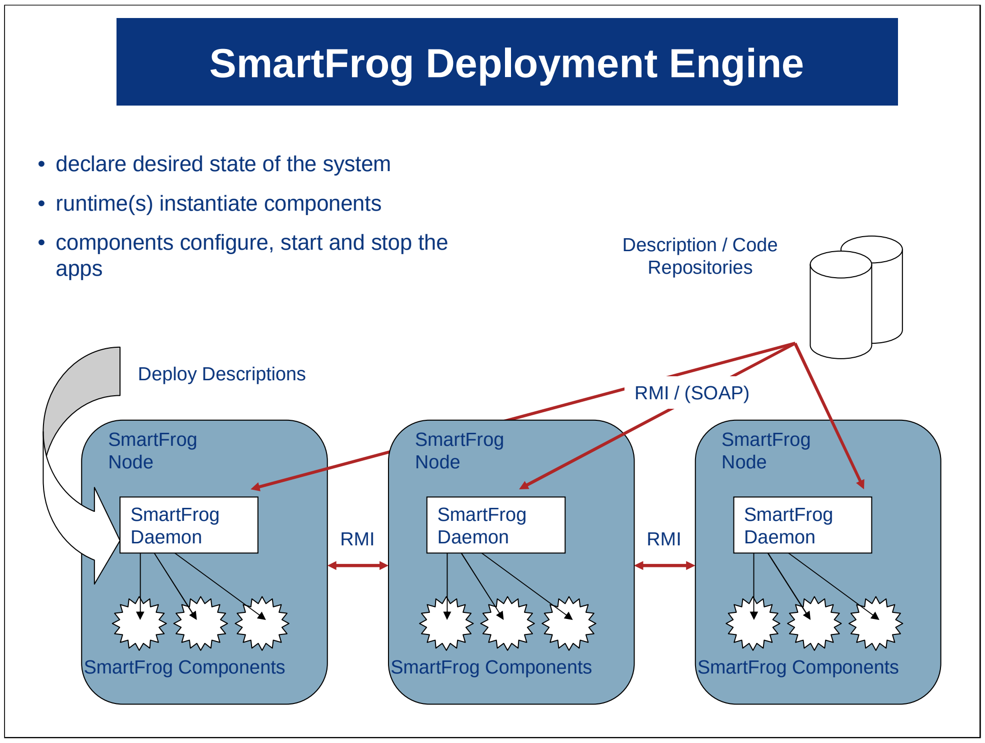
How to host big applications across distributed resources

- Automatically / Repeatably
- Dynamically
- Correctly
- Securely

How to manage them from installation to removal

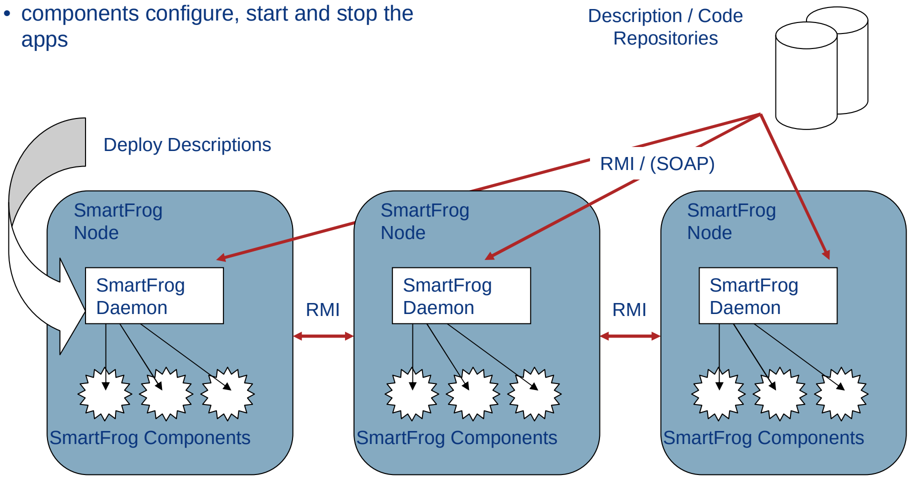
How to make grid fabrics useful for classic server-side apps

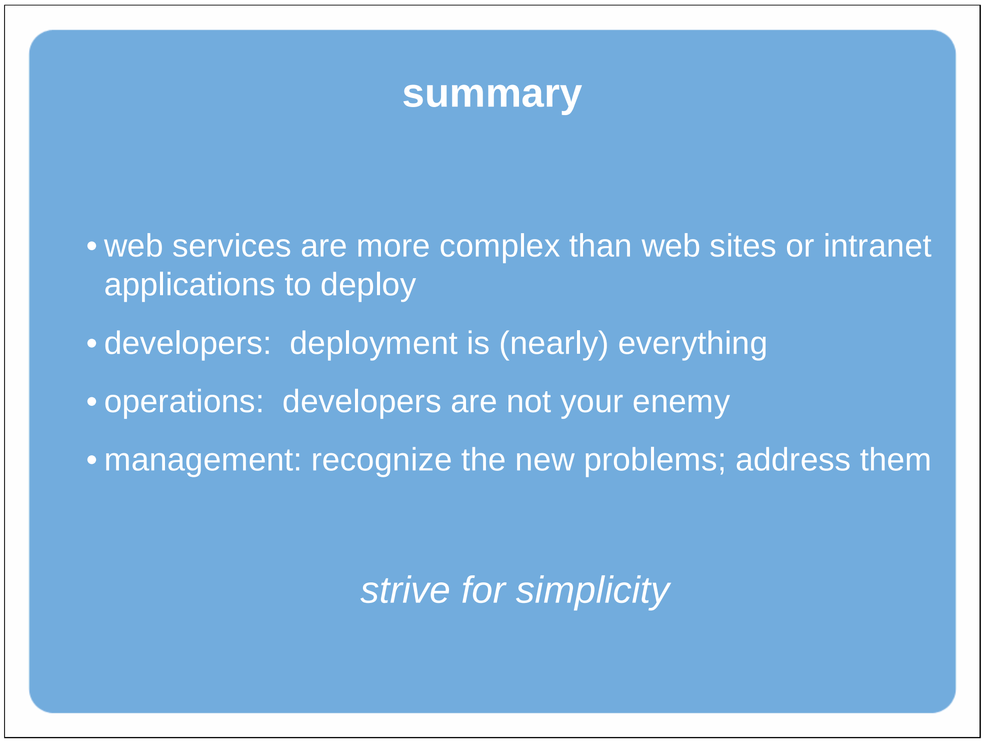




SmartFrog Deployment Engine

- declare desired state of the system
- runtime(s) instantiate components
- components configure, start and stop the apps





summary

- web services are more complex than web sites or intranet applications to deploy
- developers: deployment is (nearly) everything
- operations: developers are not your enemy
- management: recognize the new problems; address them

strive for simplicity

Remember that complexity is the enemy: seek simplicity in all that you can do. As the XP people say, refactor mercilessly. I say apply that to everything in the project: the hardware, the cluster configuration, the entire module design, the management change. Refactor and test; fight complexity by extracting simplicity.