

coding {the} architecture

Software architecture for developers

What is software
architecture?

What is the **role** of
a software architect?

How do you **define**
software architecture?

How do you **share**
software architecture?

How do you **deliver**
software architecture?

A developer's guide to load testing



simon.brown@codingthearchitecture.com

@**simonbrown** on Twitter

coding
(the)
architecture

Software Architecture for Developers (.com)



What is software architecture?

What is the role of a software architect?

How do you define software architecture?

How do you share software architecture?

How do you deliver software architecture?

coding
{the}
architecture

Why Software Projects Fail

www.codingthearchitecture.com



Simon Brown
Hands-on software
architect



coding
{the}
architecture

It should be the
architect
(somebody has to do it and that's why
we get paid the big bucks)

Why software projects **fail...**

...architects are here
to **help**, not to hinder

s-on software
itect can be
for preventing
failure

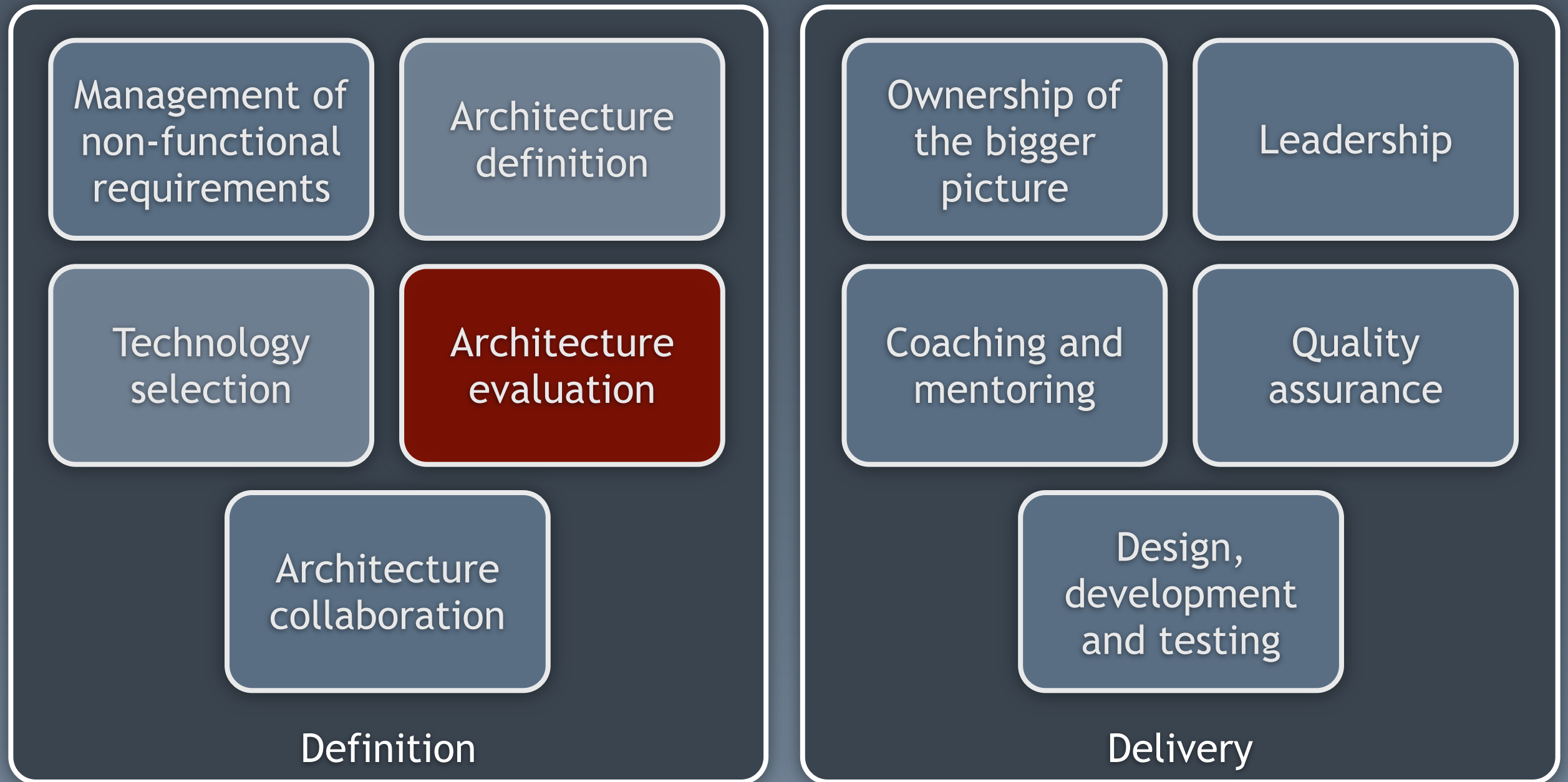
Software projects **fail**
for a number of reasons

Iterative and agile techniques
solve *some* problems...

coding
{the}
architecture



The role of a software architect



Why is quality
important?

Reputation

of the business

(e.g. driven by customer satisfaction)

Service level agreements & key performance indicators

(e.g. between suppliers, between systems,
non-functional requirements, etc...)

Reputation

of the development team



It's important
that we know
what
we're releasing

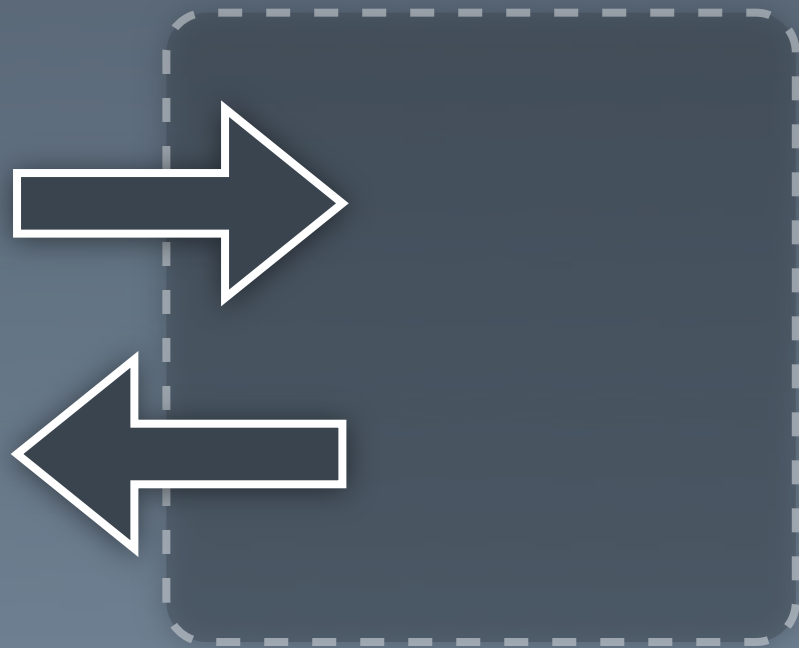
It's important
that the
software we
release
“works”

That's us!

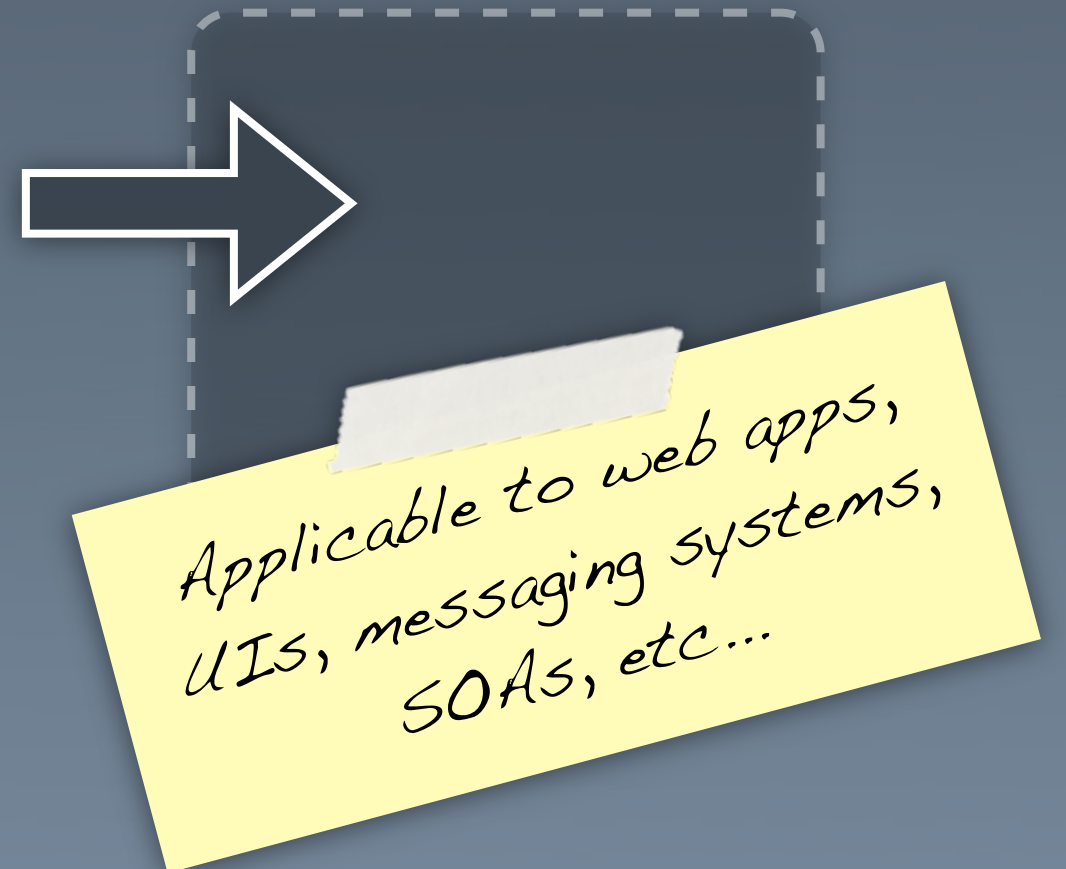
What is
performance?

Performance is about how
fast something is

Response time



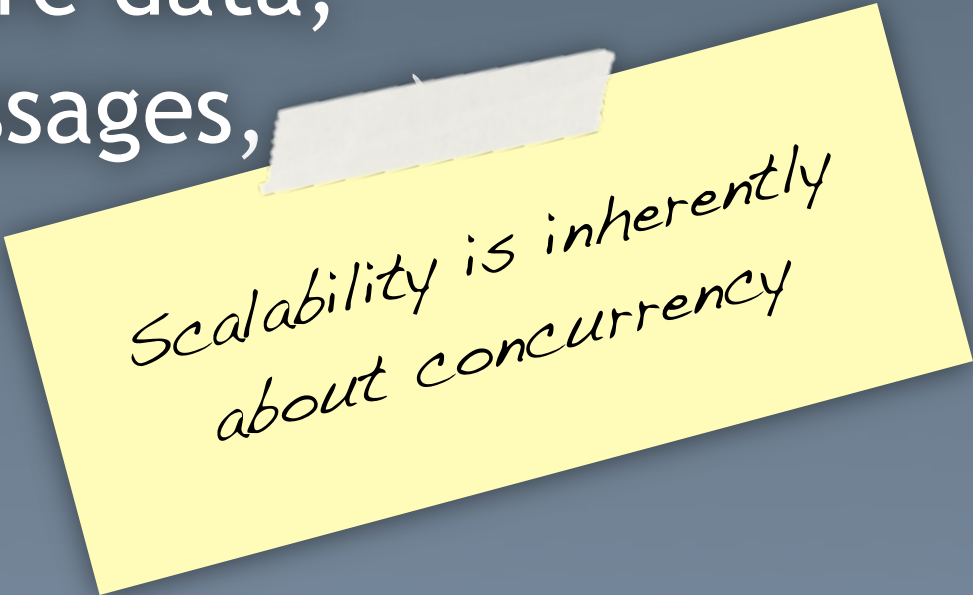
Latency_{and} throughput



What is
scalability?

Scalability is about doing more

(more requests, more data,
more users, more messages,

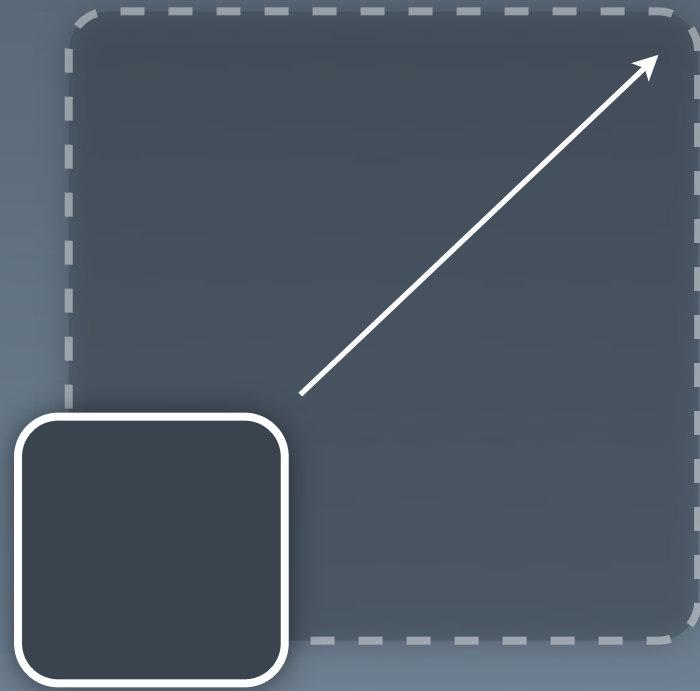


*Scalability is inherently
about concurrency*

Vertical

scalability

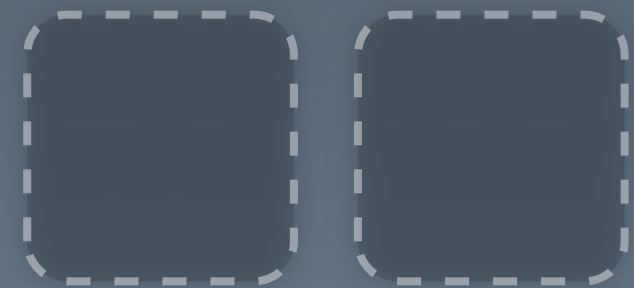
(scale-up)



Horizontal

scalability

(scale-out)



*All components
scale differently;
scale-up is easy*

Scalability Principles

Posted by [Simon Brown](#) on May 21, 2008
 Community [Architecture](#) Topics [Performance & Scalability](#) Tags [Concurrency](#)

At the simplest level, [scalability is about doing more of something](#). This could be responding to more user requests, executing more work or handling more data. While designing software has its complexities, making that software capable of doing lots of work presents its own set of problems. This article presents some principles and guidelines for building scalable software systems.

Related Vendor Content

- [5 Ways to Ensure Application Performance](#)
- [Comprehensive Threat Protection for REST, SOA, and Web 2.0 Applications](#)
- [The Agile Project Manager](#)
- [The Agile Checklist](#)
- [Adobe® Flash® Platform Overview PDF](#)

1. Decrease processing time

One way to increase the amount of work that an application does is to decrease the time taken for individual work units to complete. For example, decreasing the amount of time required to process a user request means that you are able to handle more user requests in the same amount of time. Here are some examples of where this principle is appropriate and some possible realisation strategies.

- **Collocation** : reduce any overheads associated with fetching data required for a piece of work, by collocating the data and the code.
- **Caching** : if the data and the code can't be collocated, cache the data to reduce the overhead of fetching it over and over again.
- **Pooling** : reduce the overhead associated with using expensive resources by pooling them.
- **Parallelization** : decrease the time taken to complete a unit of work by decomposing the problem and parallelizing the individual steps.
- **Partitioning** : concentrate related processing as close together as possible, by partitioning the code and collocating related partitions.
- **Remoting** : reduce the amount of time spent accessing remote services by, for example, making the interfaces more coarse-grained. It's also worth remembering that remote vs local is an explicit design decision not a switch and to consider the first law of distributed computing – do not distribute your objects.

As software developers, we tend to introduce abstractions and layers where they are often not required. Yes, these concepts are great tools for decoupling software components, but they have a tendency to increase complexity and impact performance, particularly if you're converting between data representations at each layer. Therefore, the other way in which processing time can be minimised is to ensure that the abstractions aren't too abstract and that there's not too much layering. In addition, it's worth understanding the cost of runtime services that we take for granted because, unless they have a specific service level agreement, it's possible that these could end up being the bottlenecks in our applications.

2. Partition

Decreasing the processing time associated with a particular work unit will get you so far, but ultimately you'll need to scale out your system when you reach the limits of a single process deployment. In a typical web application, scaling out *could* be as easy as starting up additional web servers to handle the user requests and load balancing between them. What you might find, however, is that parts of your overall architecture will start to become points of contention because everything will get busy at the same time. A good example is a single database server sitting behind all those web servers. When that starts to become the bottleneck, you have to change your approach and one way to do this is to adopt a partitioning strategy. Put simply, this involves breaking up that single piece of the architecture into smaller more manageable chunks. Partitioning that single element into smaller chunks allows you to scale them out and this is exactly the technique that large sites such as eBay use to ensure that their architectures scale. Partitioning is a good solution, although you may find that you [trade-off consistency](#).

As to how you partition your system, well that depends. Truly stateless components can simply be scaled out and the work load balanced between them, ideally with all instances of the component running in an active manner. If, on the other hand, there is state that needs to be maintained, you need to find a workload partitioning strategy that will allow you to have multiple instances of those stateful components, where each instance is responsible for a distinct subset of the work and/or data.

3. Scalability is about concurrency

Scalability is inherently about concurrency; after all, it's about doing more work at the same time. Technologies such as the early versions of Enterprise JavaBeans (EJB) attempted to provide a simplified programming model and encouraged us to write components that were single-threaded. Unfortunately, these components typically had dependencies on other components and this led to concurrency problems. If concurrency isn't thought about, you have systems where data can easily become corrupted. On the other hand, too many guards around concurrency lead to systems that are essentially serial in nature and limited in the degree to which they can scale. Concurrent programming isn't *that* hard to do, but there are some simple principles that can help when building scalable systems.

- If you do need to hold locks (e.g. local objects, database objects, etc), try to hold them for as little time as possible.
- Try to minimize contention of shared resources and try to take any contention off of the critical processing path (e.g. by scheduling work asynchronously).
- Any design for concurrency needs to be done up-front, so that it's well understood which resources can be shared safely and where potential scalability bottlenecks will be.

4. Requirements must be known

In order to build a successful software system, you need to know what your goals are and what you're aiming for. While the functional requirements are often well-known, it's the non-functional requirements (or system qualities) that are [usually absent](#). If you do genuinely need to build a piece of software that is highly scalable, then you need to understand the following types of things up-front for the critical components/workflows.

- Target average and peak performance (i.e. response time, latency, etc).
- Target average and peak load (i.e. concurrent users, message volumes, etc).
- Acceptable limits for performance and scalability.

What do you
want to know?

(and why?)

*This new system will work
because we have a*

spreadsheet

*showing performance figures
from a past project*



Testing provides
confidence

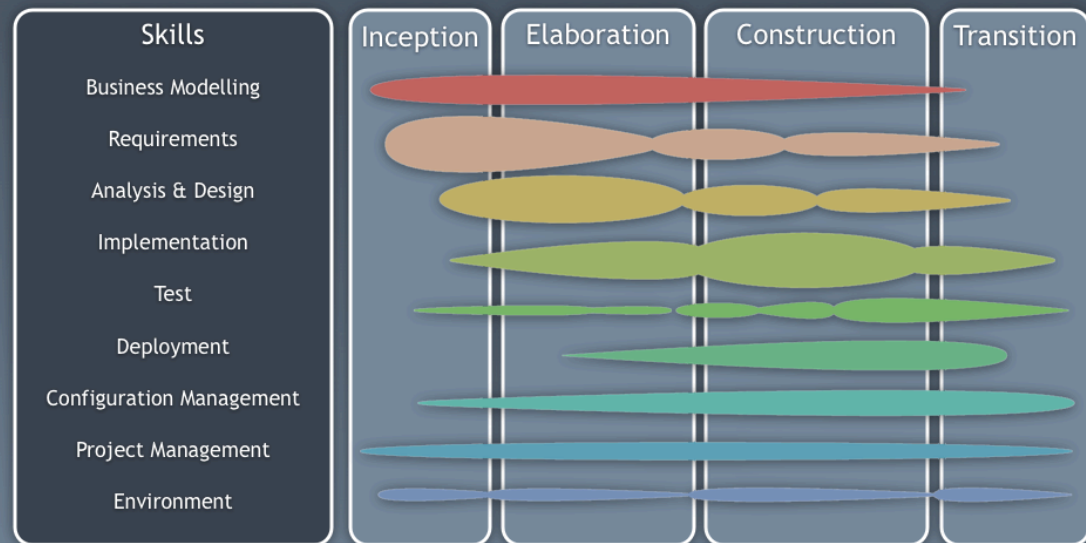
Load testing is one way to

evaluate

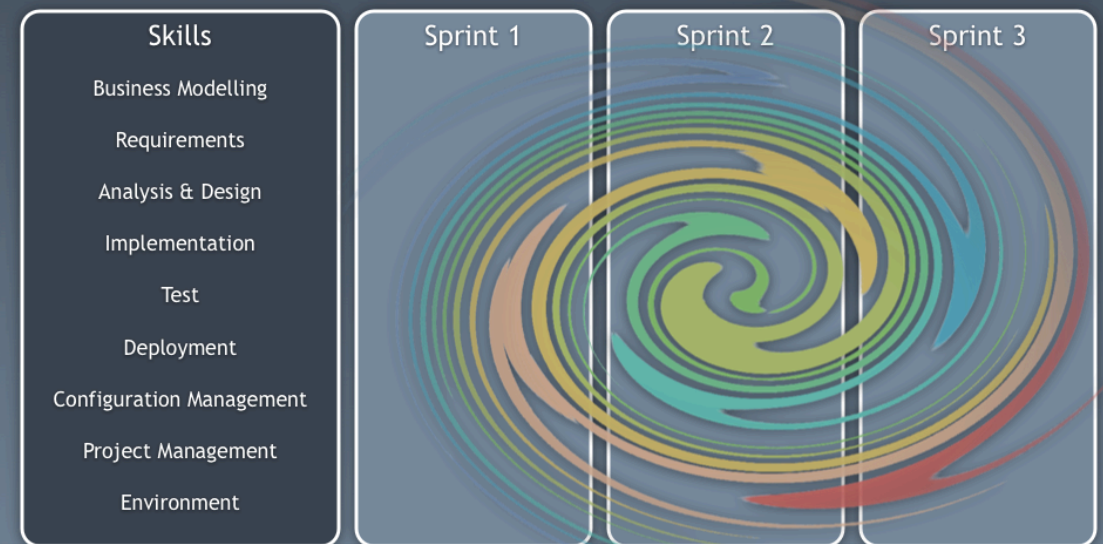
your architecture

(if performance and/or scalability is important to you)

Rational Unified Process (RUP)



Scrum



Load test early

You should have a
reason
for load testing

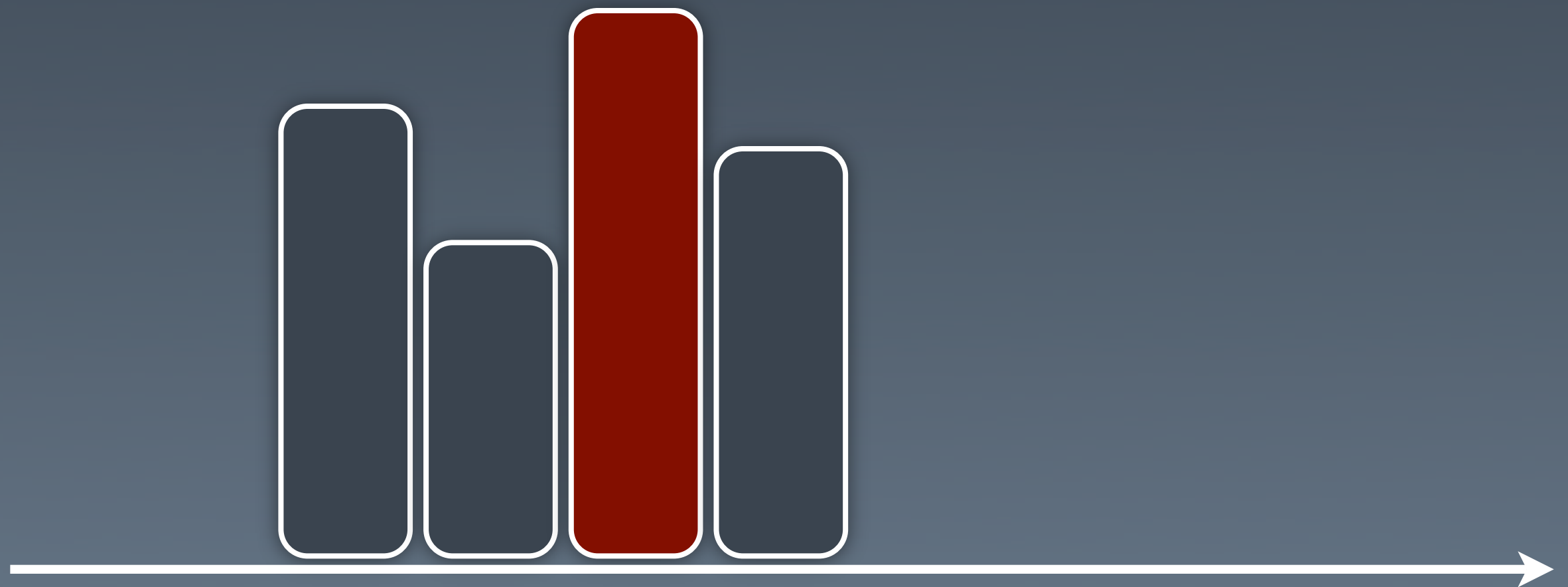
What do you want to
learn
about your system?

How fast is it?
How does it scale?
Where does it break?

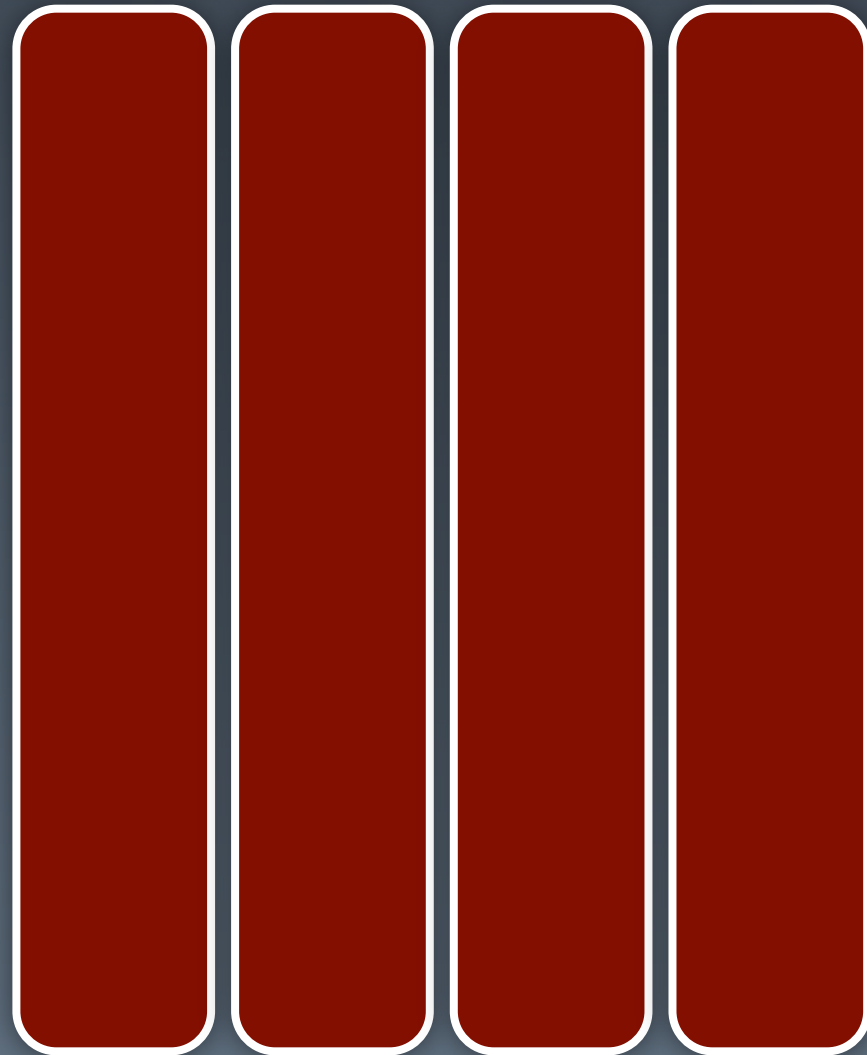
What do you want to
prove
about your system?

Response times are X.
Throughput is Y.
Scales to Z users.

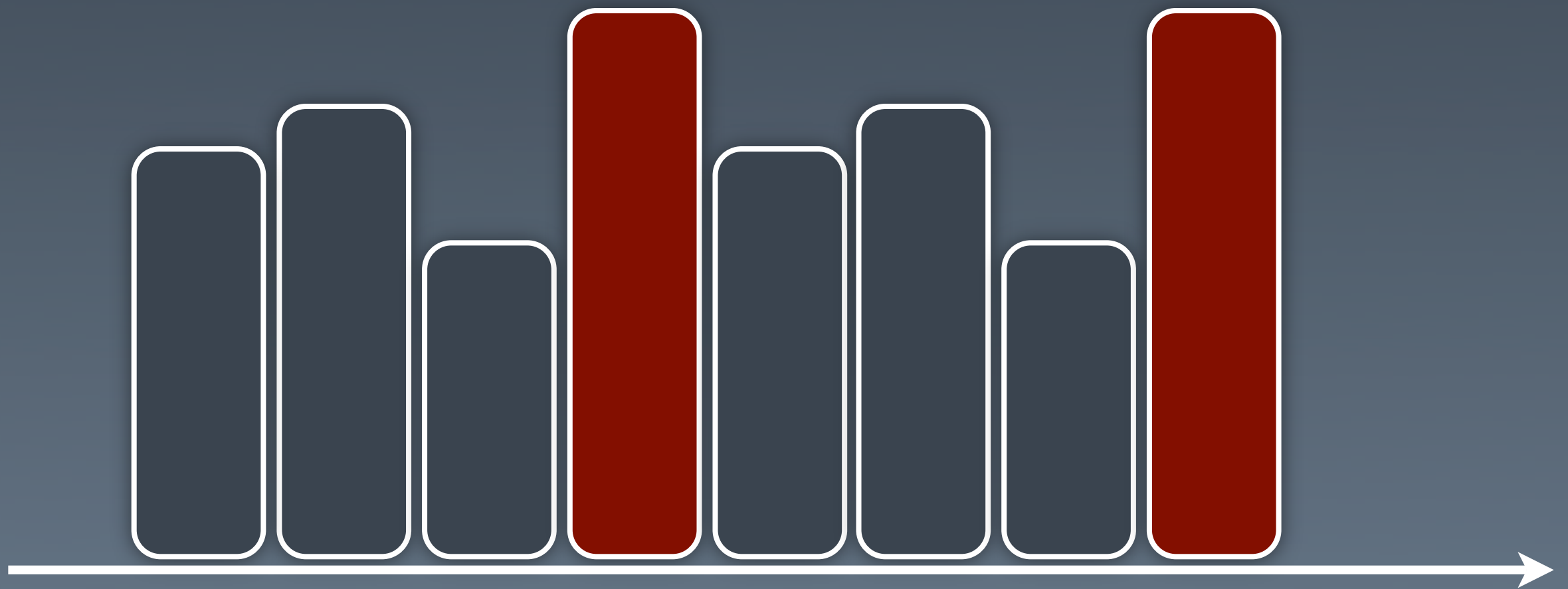
What is
load testing?



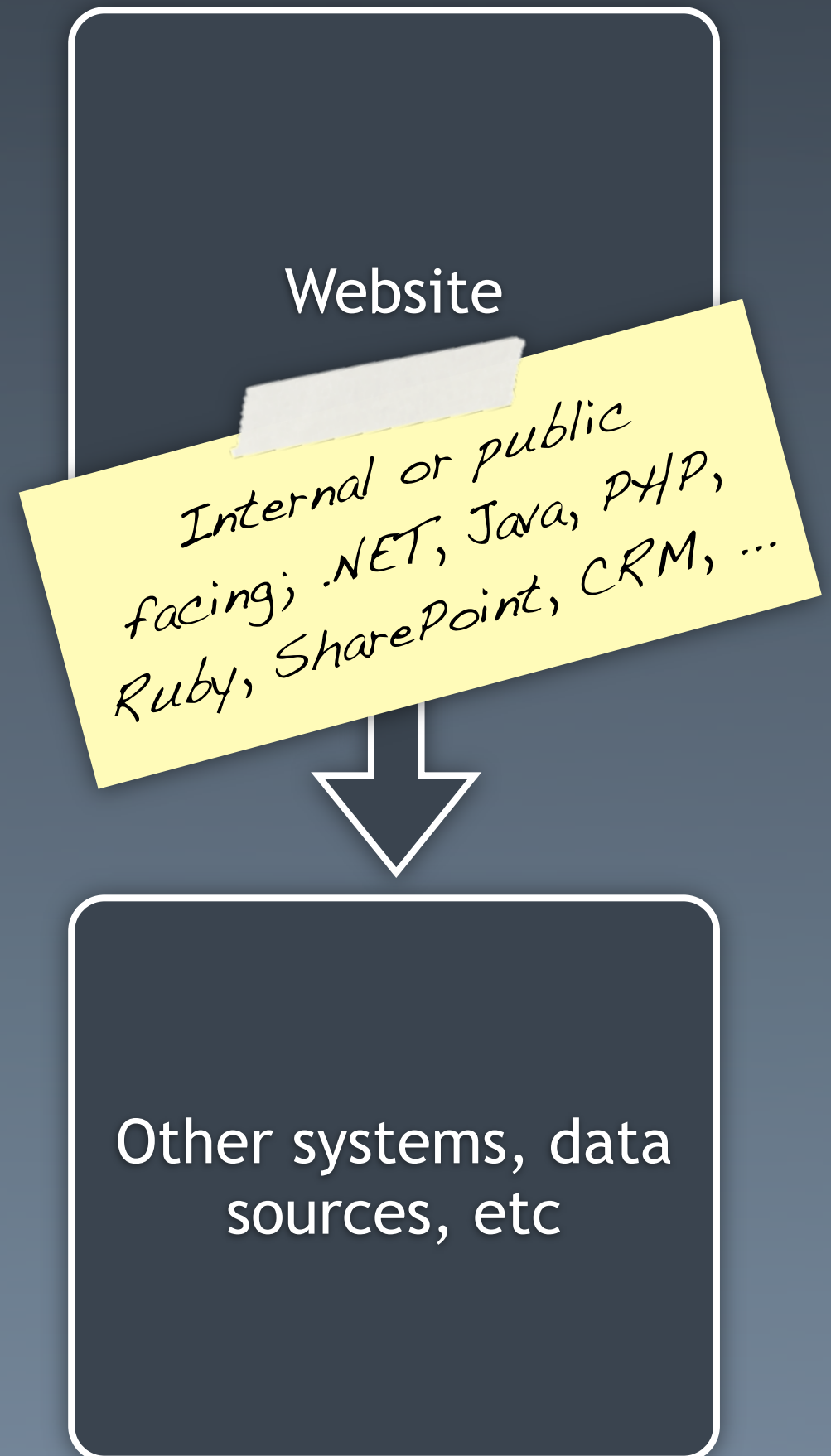
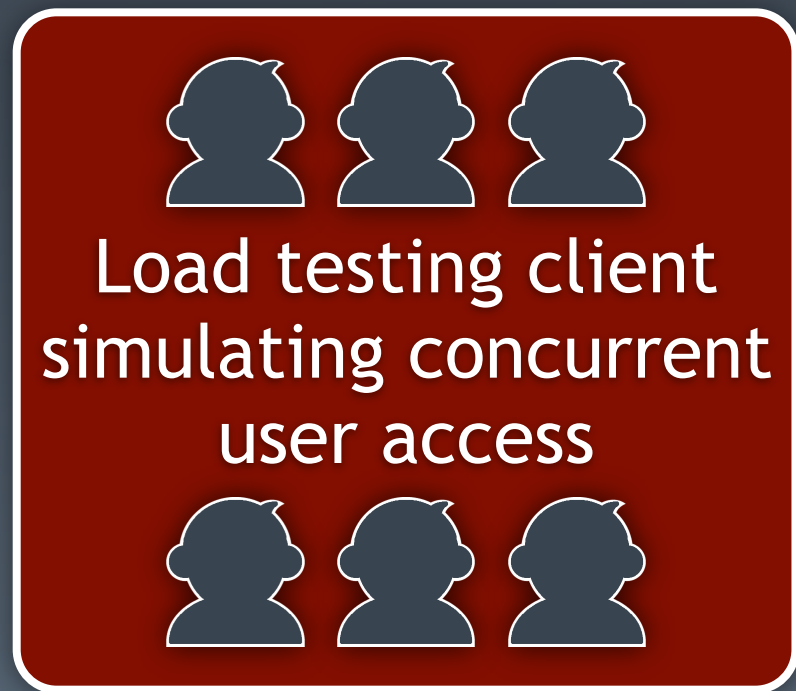
Load testing is asserting how the architecture performs under load with a view to monitoring the response times for key transactions



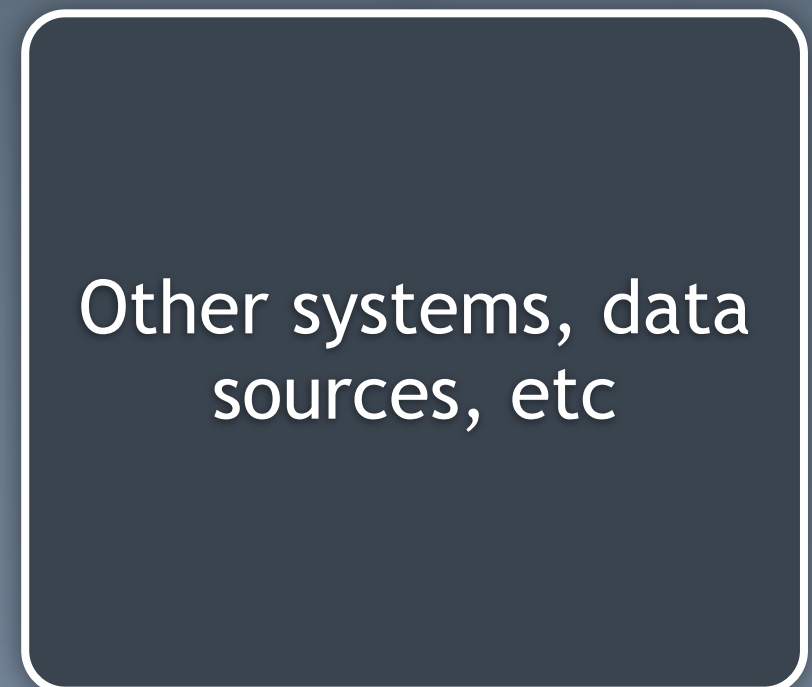
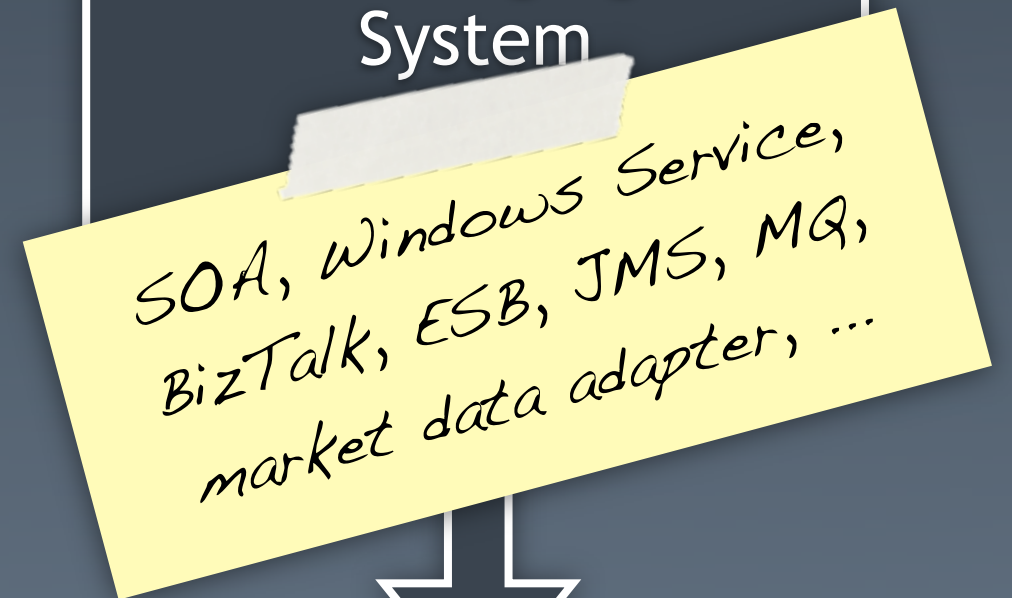
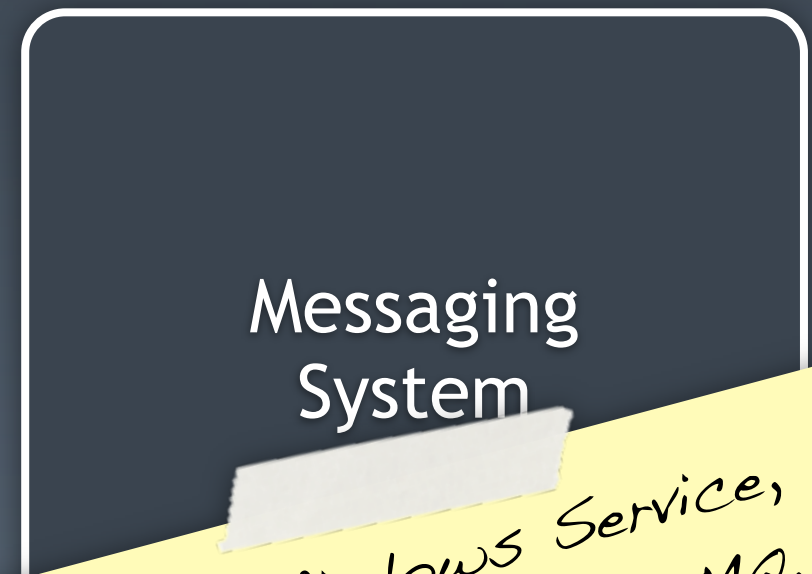
Stress testing is asserting what the upper bounds are for the scalability of the architecture, understanding how it reacts when stressed



Soak testing is asserting that the performance of the architecture remains stable over longer periods of time



Simulate multiple users
with a
typical usage profile,
preferably with an environment as near
to production as possible



Generate
representative input
messages, preferably with an
environment as near
to production as possible

Step one

Understand what you want to learn/prove

Performance and scalability characteristics,
typical usage profiles, etc.

Use log files, audit logs or
analytics to understand
usage patterns

Step two

Define the test script

Determine the actions to simulate from the test script and implement with a load testing tool.

Don't forget to add
post-request assertions

Step three

Schedule and configure environment

Book testing slots to minimise disruption and configure/clear data for load testing.

Beware of running
load tests on
production LANs!

Step four

Determine metrics to record and monitor

Ensure that the test script captures the appropriate statistics and determine the system characteristics to monitor (e.g. CPU, RAM, IO, etc).

Don't forget to monitor the load testing client(s)

Step five

Execute pre-tests

Test the test scripts and monitor,
refining if necessary.

This is where you'll
find reliance on data
or the environment

Step six

Execute tests

Clear down the environment, warm it up,
execute tests at varying levels of
concurrent usage.

Try to run the same test
a number of times
to ensure consistency

Step seven

Analyse results

Calculate useful statistics (average, maximum and 95th percentile response times), draw graphs and make conclusions.

post-processing the raw results provides more flexibility

Analysing the results

timeStamp, elapsed, label, responseCode, threadName, success, URL

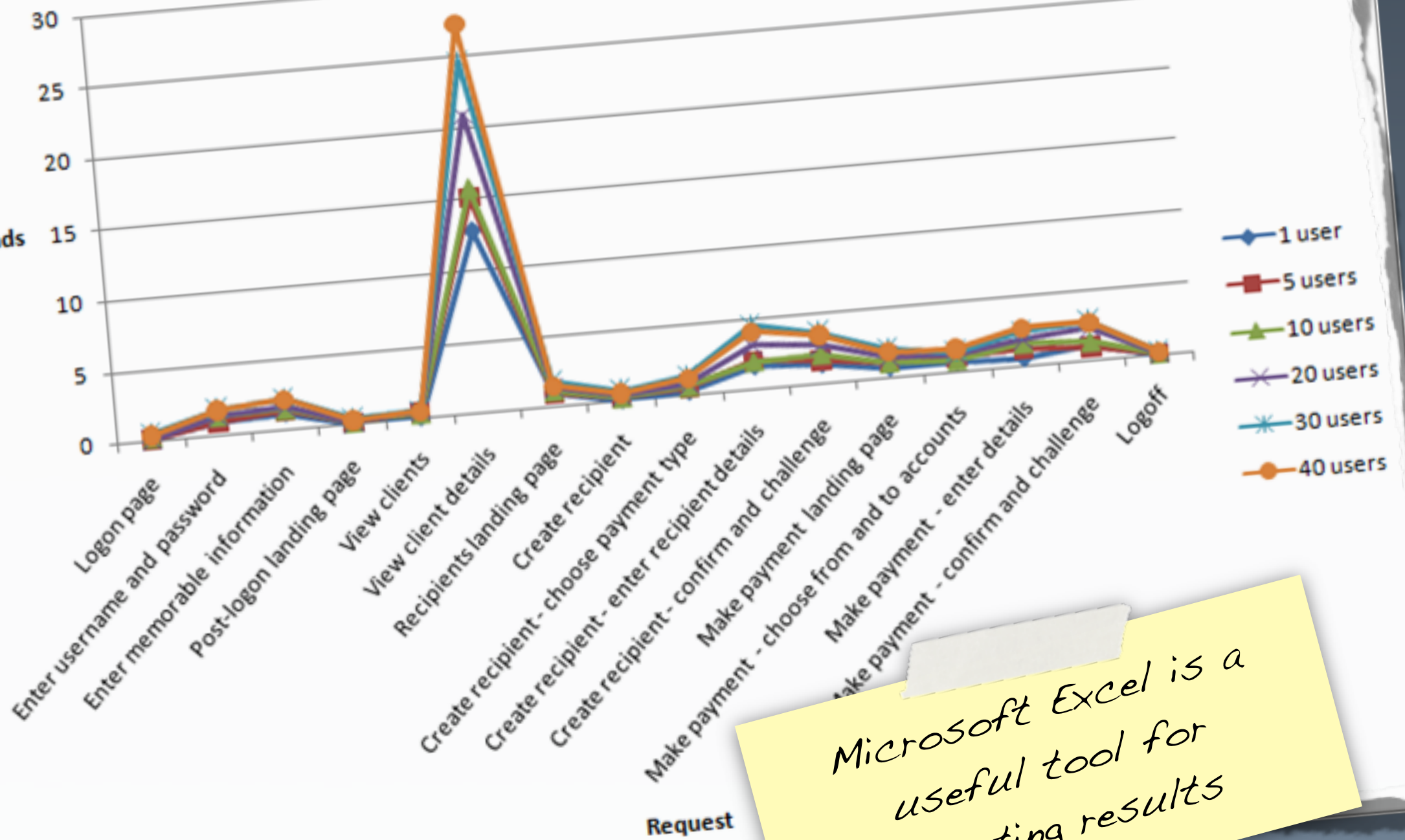
1242921099104,77,01. Logon page,200,Corporate user 1-1,true,http://ib4currencydev/user/Logon.aspx
1242921099318,684,02. Enter username and password,200,Corporate user 1-1,true,http://ib4currencydev/user/Logon.aspx
1242921100101,1057,03. Enter memorable information,302,Corporate user 1-1,true,http://ib4currencydev/user/Logon.aspx
1242921101170,10,04. Post-logon landing page,200,Corporate user 1-1,true,http://ib4currencydev/user/Logon.aspx
1242921101191,42,05. View clients,200,Corporate user 1-2,true,http://ib4currencydev/view/relationship.aspx
1242921104394,13,01. Logon page,200,Corporate user 1-2,true,http://ib4currencydev/user/Logon.aspx
1242921104420,788,02. Enter username and password,200,Corporate user 1-2,true,http://ib4currencydev/user/Logon.aspx
1242921105232,872,03. Enter memorable information,302,Corporate user 1-2,true,http://ib4currencydev/user/Logon.aspx
1242921106120,9,04. Post-logon landing page,200,Corporate user 1-2,true,http://ib4currencydev/view/relationship.aspx
1242921106141,715,05. View clients,200,Corporate user 1-3,true,http://ib4currencydev/user/Logon.aspx
1242921109600,166,01. Logon page,200,Corporate user 1-3,true,http://ib4currencydev/user/Logon.aspx
1242921109775,582,02. Enter username and password,200,Corporate user 1-3,true,http://ib4currencydev/user/Logon.aspx
1242921110377,835,03. Enter memorable information,302,Corporate user 1-3,true,http://ib4currencydev/user/Logon.aspx
1242921111221,9,04. Post-logon landing page,200,Corporate user 1-3,true,http://ib4currencydev/view/account-list.aspx
1242921101254,10312,06. View client details,200,Corporate user 1-3,true,http://ib4currencydev/view/relationship.aspx
1242921111239,925,05. View clients,200,Corporate user 1-1,true,http://ib4currencydev/recipients/Default.aspx
1242921111584,1612,07. Recipients landing page,200,Corporate user 1-1,true,http://ib4currencydev/recipients/Create.aspx
1242921113208,51,08. Create recipient,200,Corporate user 1-1,true,http://ib4currencydev/recipients/Create.aspx
1242921113274,268,09. Create recipient - choose payment type,200,Corporate user 1-1,true,http://ib4currencydev/recipients/Create.aspx
1242921113558,1613,10. Create recipient - enter recipient details,200,Corporate user 1-1,true,http://ib4currencydev/recipients/Create.aspx
1242921114709,480,01. Logon page,200,Corporate user 1-4,true,http://ib4currencydev/user/Logon.aspx
1242921115197,626,02. Enter username and password,200,Corporate user 1-4,true,http://ib4currencydev/user/Logon.aspx
1242921115857,957,03. Enter memorable information,302,Corporate user 1-4,true,http://ib4currencydev/user/Logon.aspx
1242921116823,11,04. Post-logon landing page,200,Corporate user 1-4,true,http://ib4currencydev/user/Logon.aspx
1242921116842,37,05. View clients,200,Corporate user 1-4,true,http://ib4currencydev/view/relationship.aspx

Post-process

the raw results

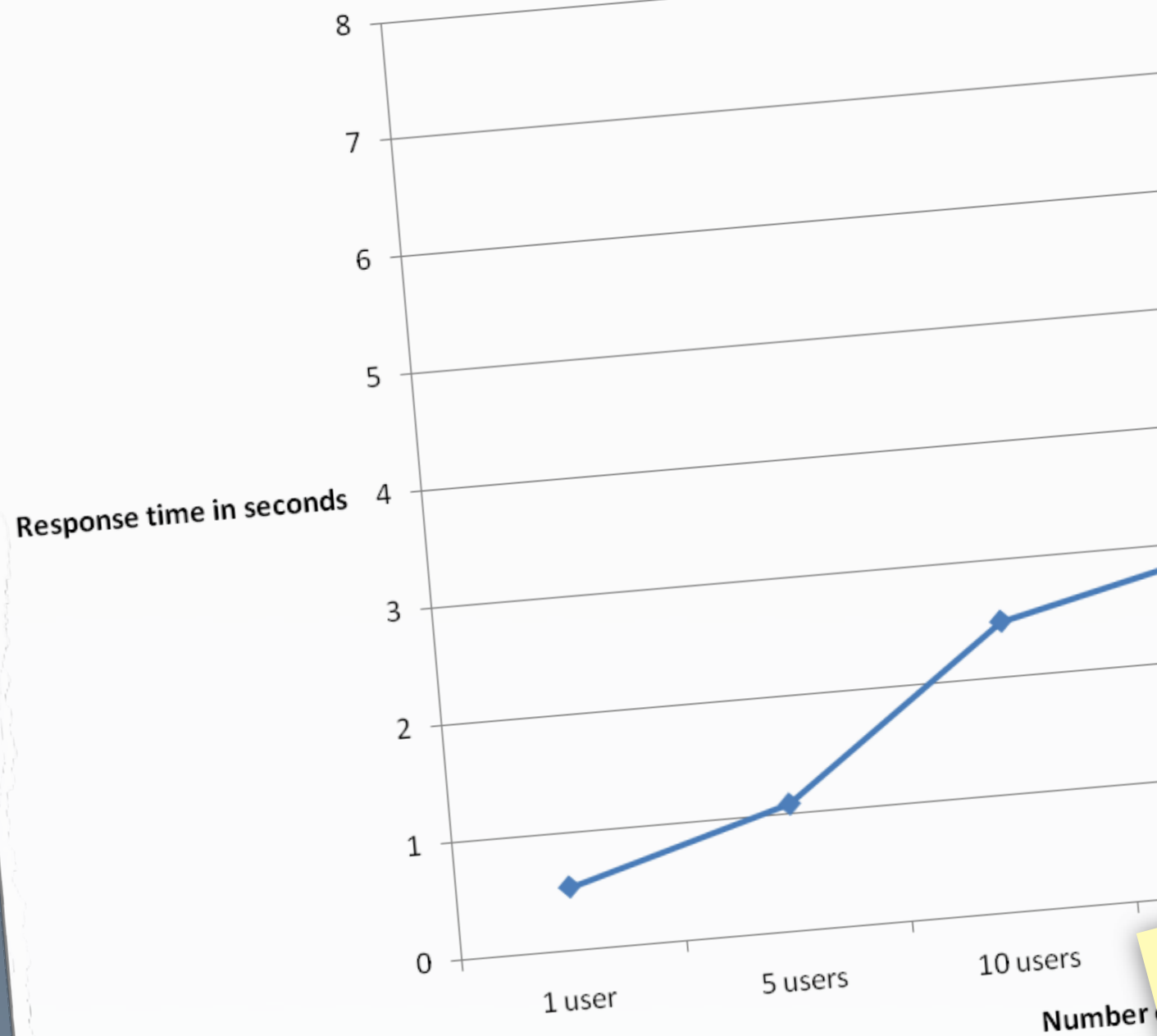
Average response times (raw)

Response time in seconds

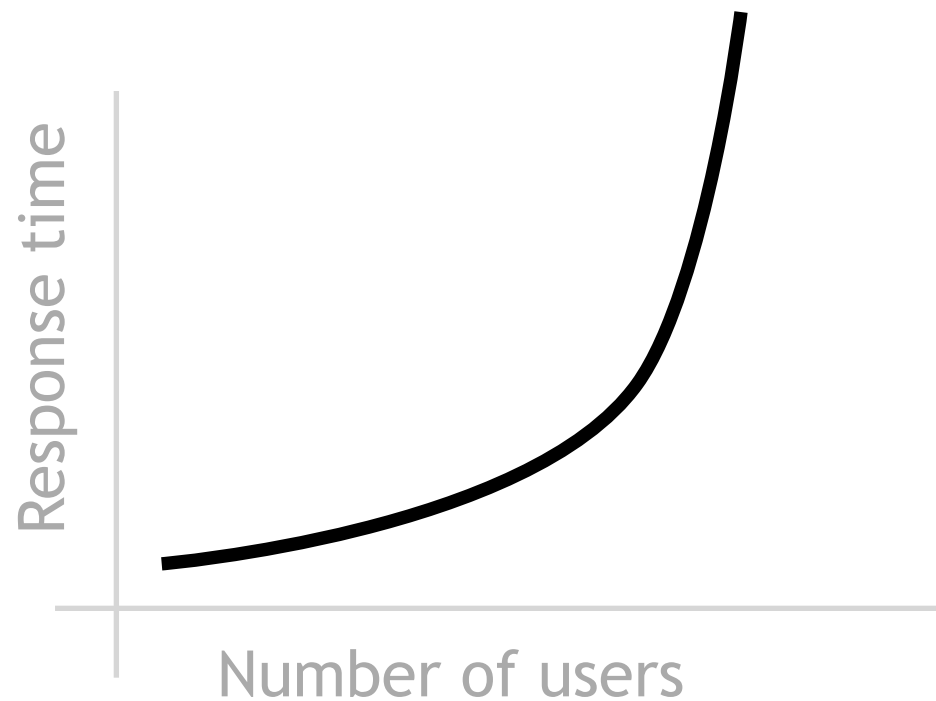


Microsoft Excel is a useful tool for charting results

Scalability: Make Payment - enter details

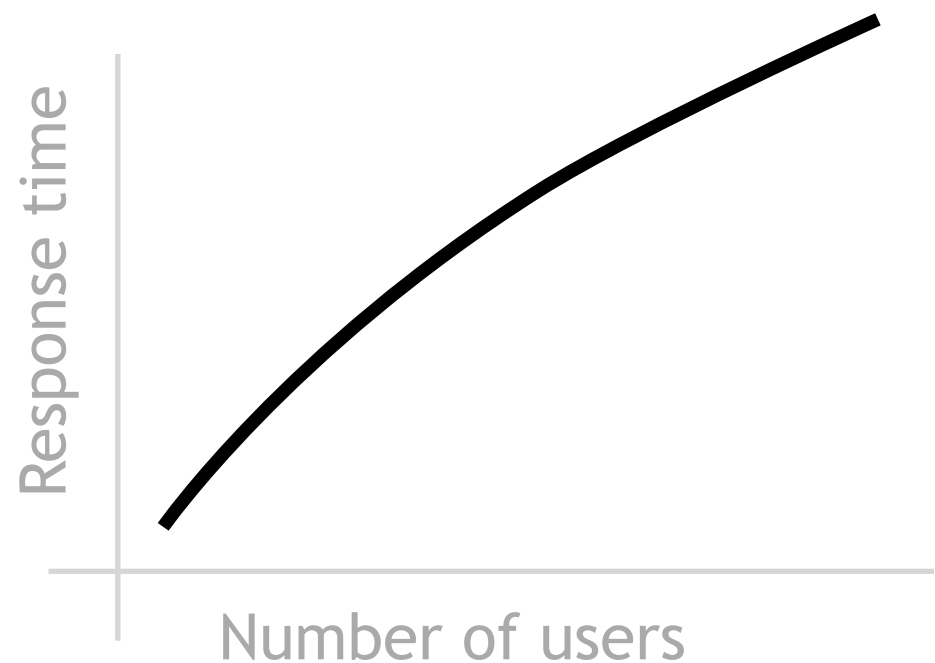


Draw this sort of graph if you need to understand scalability



This shows poor scalability because response times have started to increase exponentially

This shows good scalability because response times increase in a predictable way



Diagnosing problems

What happens if you find
performance
problems?

Look at your monitoring data
and log files,
run profilers,
etc...

Maybe the environment
needs tuning

[Memory | Connection pools | Worker thread pools |
Database indexes & hints | etc]

The **big** picture and
the **detail**
are equally as important

My Java application has
intermittent
poor performance

*Repeat the test with
garbage collection
logging enabled*

My application is slow, but it's
not using any CPU

Repeat the test and
monitor IO
(network and disk)

My application is

slow

Find the bottleneck by
profiling your application

Gotchas


```
Terminal — bash — 97x32
raw=1263861049883 | formatted=19-Jan-2010 00:30:49:883 | frequency=16152 | delta=1ms
raw=1263861049884 | formatted=19-Jan-2010 00:30:49:884 | frequency=16154 | delta=1ms
raw=1263861049885 | formatted=19-Jan-2010 00:30:49:885 | frequency=16156 | delta=1ms
raw=1263861049886 | formatted=19-Jan-2010 00:30:49:886 | frequency=16158 | delta=1ms
raw=1263861049887 | formatted=19-Jan-2010 00:30:49:887 | frequency=16160 | delta=1ms
raw=1263861049888 | formatted=19-Jan-2010 00:30:49:888 | frequency=16162 | delta=1ms
raw=1263861049889 | formatted=19-Jan-2010 00:30:49:889 | frequency=16164 | delta=1ms
raw=1263861049890 | formatted=19-Jan-2010 00:30:49:890 | frequency=16166 | delta=1ms
raw=1263861049891 | formatted=19-Jan-2010 00:30:49:891 | frequency=16168 | delta=1ms
raw=1263861049892 | formatted=19-Jan-2010 00:30:49:892 | frequency=16170 | delta=1ms
raw=1263861049893 | formatted=19-Jan-2010 00:30:49:893 | frequency=16172 | delta=1ms
raw=1263861049894 | formatted=19-Jan-2010 00:30:49:894 | frequency=16174 | delta=1ms
raw=1263861049895 | formatted=19-Jan-2010 00:30:49:895 | frequency=16176 | delta=1ms
raw=1263861049896 | formatted=19-Jan-2010 00:30:49:896 | frequency=16178 | delta=1ms
raw=1263861049897 | formatted=19-Jan-2010 00:30:49:897 | frequency=16180 | delta=1ms
raw=1263861049898 | formatted=19-Jan-2010 00:30:49:898 | frequency=16182 | delta=1ms
raw=1263861049899 | formatted=19-Jan-2010 00:30:49:899 | frequency=16184 | delta=1ms
raw=1263861049900 | formatted=19-Jan-2010 00:30:49:900 | frequency=16186 | delta=1ms
raw=1263861049901 | formatted=19-Jan-2010 00:30:49:901 | frequency=16188 | delta=1ms
raw=1263861049902 | formatted=19-Jan-2010 00:30:49:902 | frequency=16190 | delta=1ms
raw=1263861049903 | formatted=19-Jan-2010 00:30:49:903 | frequency=16192 | delta=1ms
raw=1263861049904 | formatted=19-Jan-2010 00:30:49:904 | frequency=16194 | delta=1ms
raw=1263861049905 | formatted=19-Jan-2010 00:30:49:905 | frequency=16196 | delta=1ms
raw=1263861049906 | formatted=19-Jan-2010 00:30:49:906 | frequency=16198 | delta=1ms
raw=1263861049907 | formatted=19-Jan-2010 00:30:49:907 | frequency=16200 | delta=1ms
raw=1263861049908 | formatted=19-Jan-2010 00:30:49:908 | frequency=16202 | delta=1ms
raw=1263861049909 | formatted=19-Jan-2010 00:30:49:909 | frequency=16204 | delta=1ms
raw=1263861049910 | formatted=19-Jan-2010 00:30:49:910 | frequency=16206 | delta=1ms

Minimum delta : 1ms
Maximum delta : 1ms
simonbrown:Desktop simon$
```

```
C:\WINDOWS\system32\cmd.exe
C:\>java -jar clock-accuracy.jar
raw=1263860967794 | formatted=19-Jan-2010 00:29:27:794 | frequency=13994 | delta=0ms
raw=1263860967810 | formatted=19-Jan-2010 00:29:27:810 | frequency=753345 | delta=16ms

Minimum delta : 16ms
Maximum delta : 0ms
C:\>_
```

*16 milliseconds on
Windows*

Clock granularity
can vary by platform

What's the
time?

Are you sure?

How much does
this all cost?

1-2 days

for a simple load testing script,
a few simple test runs and
some data analysis

1-2 weeks

for a realistic load testing script, some
representative test runs,
some data analysis and diagnostics

Let's wrap up

Few

software teams do any
sort of load testing

Even a little testing can increase

confidence,

particularly if done

early



simon.brown@codingthearchitecture.com

@simonbrown on Twitter

Thanks for listening.
Any questions?

coding
(the)
architecture



<http://www.codingthearchitecture.com>

Software architecture for developers