Error Handling by Design

Andy Longshaw
Blue Skyline Ltd.
andy@blueskyline.com

Some important questions

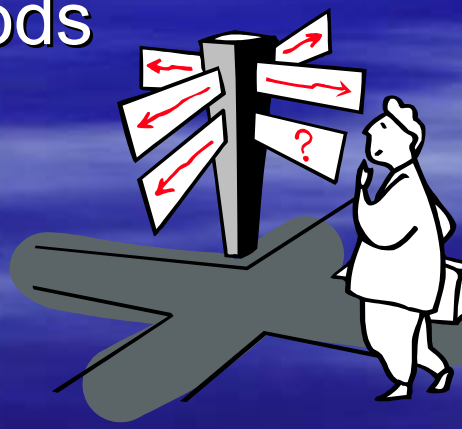
- Who am I?
- Who are you?
- Pre-requisites
  - An understanding of distributed systems
- Objectives
  - Examine issues for reporting of errors in distributed systems
  - Some approaches to dealing with these issues

The forgotten stakeholders

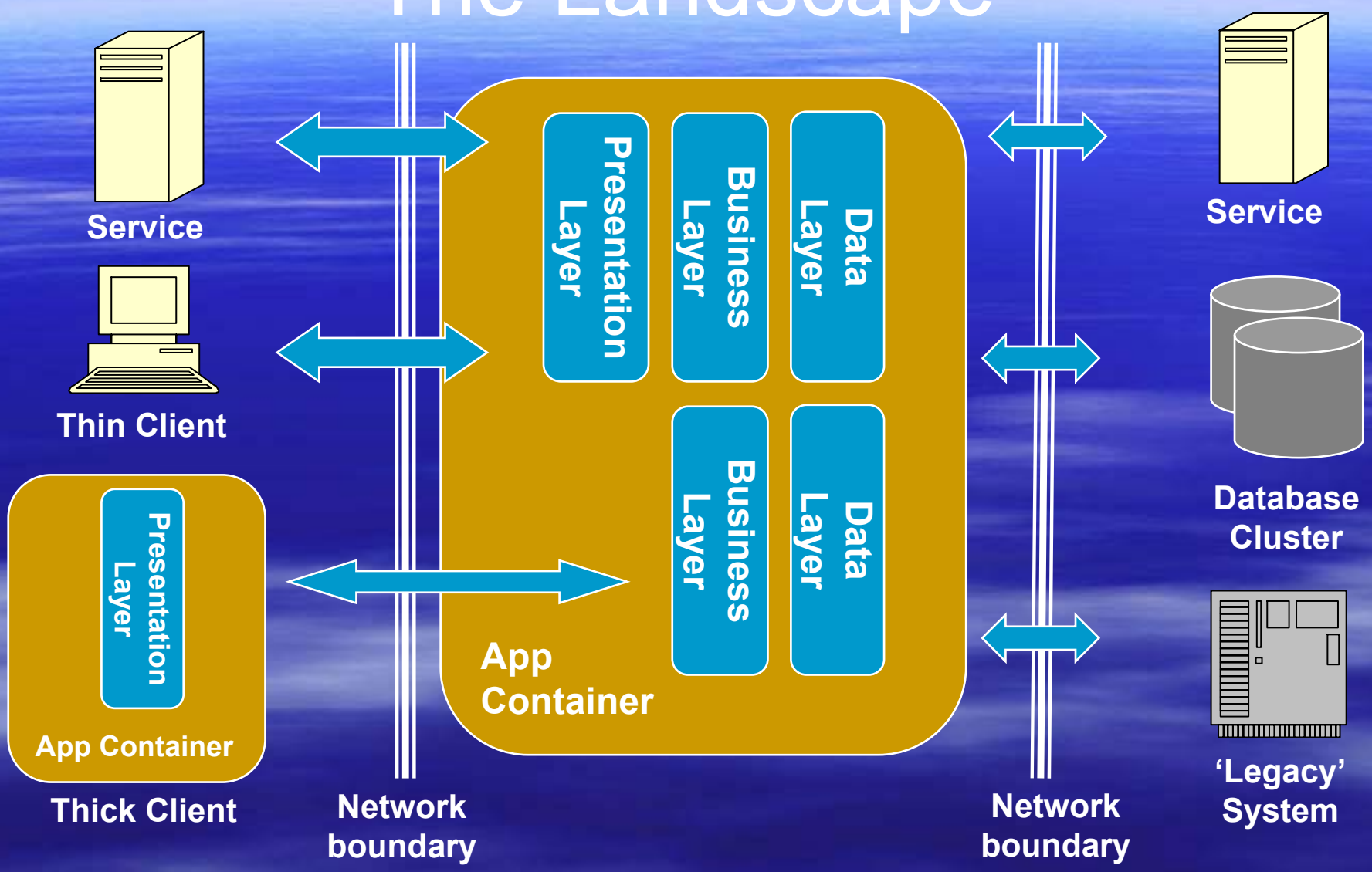
- Recent recognition of multiple stakeholders
  - Principal ones: Users/customers, business sponsors, developers
- Often the needs of other stakeholders are only considered briefly
  - Testers, deployment team, support staff
  - “Why should we go out of our way to make their jobs easier?”
- Success of a system is as much about operation and support as about features and functionality

What are we trying to do?

- Overall context:
  - Errors in distributed business systems
- Mining a coherent set of patterns
  - How did we get here?
  - From a conversation with Eoin Woods
  - No obvious source of guidance/advice
  - Some overlap with other resources



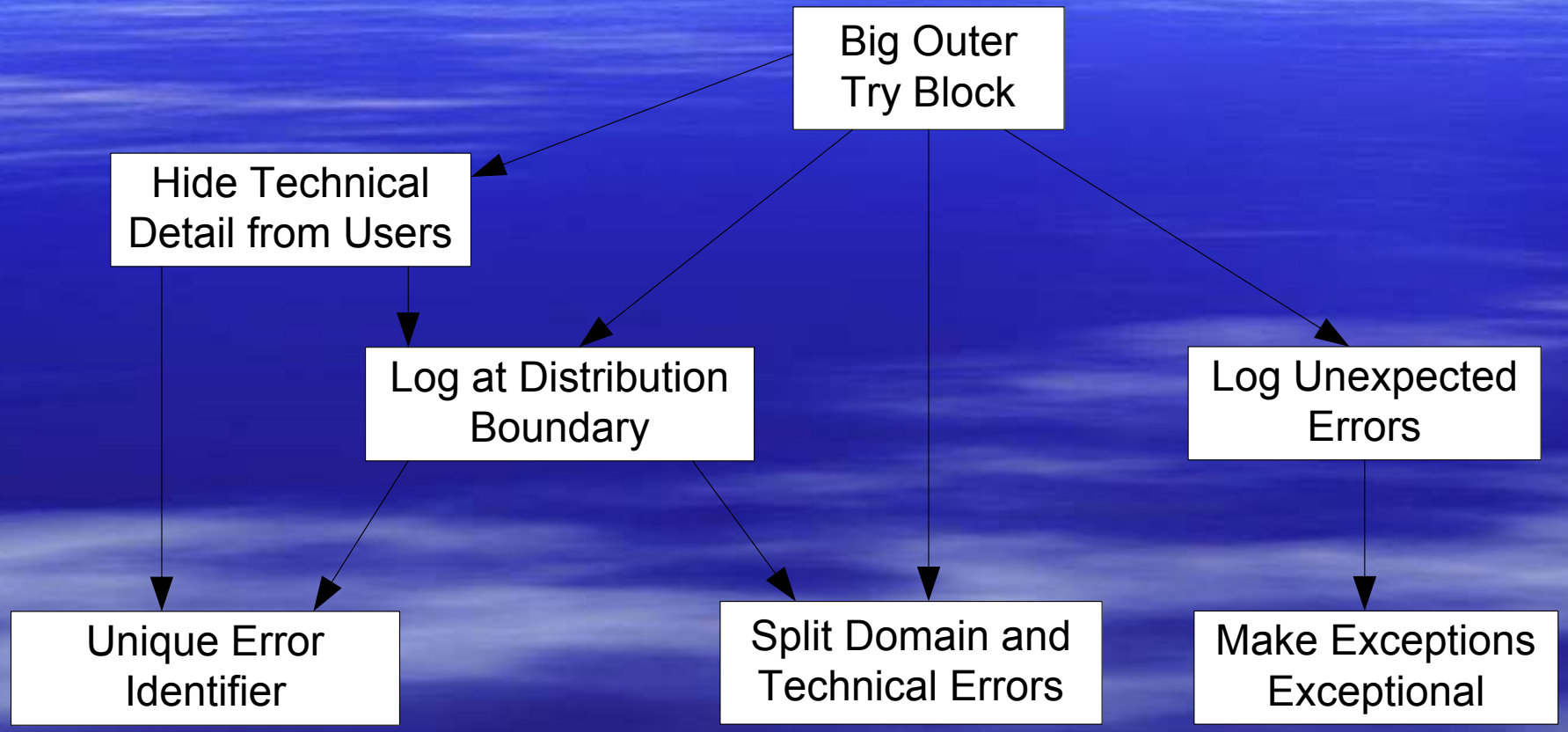
The Landscape



Some initial thoughts

- Need information to diagnose and fix problems
- Information must be correct and appropriate
- Too much error information is as much of a problem as too little
- Error handling must be applied in a consistent and disciplined way
- Error community are the forgotten stakeholders, not the end user

Overview



Split domain and technical - 1

Handling technical errors in domain code makes this code more obscure and difficult to maintain.

- Domain and technical errors form different "areas of concern"
  - Technical errors "rise up“ from the infrastructure
  - Business errors from an incorrect business action
- Complex distributed applications => much more infrastructure to go wrong!

Split domain and technical - 2

- Technical error handling complicates domain code
- Handling needs to be different
  - May retry technical errors
  - Handled in different places (e.g. façade)
  - Interesting to different stakeholders

Split domain and technical error handling. Create separate exception/error hierarchies and handle at different points and in different ways as appropriate.

Split domain and technical – 3

- Technical errors should not cause domain errors
  - Don't "cross the beams"
- Different approach at technical boundary
  - Distribution or edge of application
  - Domain errors should pass 'seamlessly'
  - Technical errors should be logged/handled
- Business code can ignore technical errors

Log at Distribution Boundary - 1

- Multi-tier systems, particularly those that use different technologies in different tiers
- Error information is bulky
- Access to remote error logs is more difficult
- Error location information is of little use remotely

Propagating technical errors between system tiers results in error details ending up in locations (such as end-user PCs) where they are difficult to access and in a context far removed from that of the original error.

Log at Distribution Boundary – 2

When technical errors occur, log them on the system where they occur passing a simpler generic SystemError back to the caller for reporting at the end-user interface.

- Include with other boundary processing
  - Marshaling
  - Security and audit
- Log in a way appropriate to platform
- Logged with other potentially related platform errors, but…
- … error trail across multiple machines

Unique Error Identifier - 1

- You are applying *Log at Distribution Boundary*
- Cross-system forensics takes a lot of effort
  - Load balancing between tiers
  - Clock skew can confuse timestamps
  - Bursts of errors are hard to disentangle

If an error on one tier in a distributed system causes knock-on errors on other tiers you get a distorted view of the number of errors in the system and their origin.

Unique Error Identifier – 2

Generate a Unique Identifier when the original error occurs and propagate this back to the caller. Always include the Unique Identifier with any error log information so that multiple log entries from the same cause can be associated and the underlying error can be correctly identified.

- Must be unique across whole system
- GUID/UUID is a good start
- May need to pass as a string to maintain integrity
- Include whenever the error is logged

Big Outer Try Block - 1

Unexpected errors can occur in any system, no matter how well it is tested… are unlikely to be handled…will propagate right to the edge of the system and will appear to "crash" the application if not handled at that point.

- Lots of potential information lost
- Users tend not to be very technical
  - Reporting of error context + content is inaccurate
  - Will ignore and work around if possible
- Trying to avoid error handling in multiple layers

Big Outer Try Block - 2

Implement a Big Outer Try Block at the "edge" of the system to catch and handle errors…report errors in a consistent way at a level of detail appropriate to the user constituency.

- Typically a try/catch in 'main' application
  - Exe/class, JSP/ASP, service/daemon
- Tell user communities about it
  - Appropriate to their needs
  - See *Hide Technical Error Detail From Users*

Big Outer Try Block – 3

- Consistent and understandable application behaviour when an unexpected error is encountered
- Reduces the need for error handling elsewhere
- Error information is not lost

Hide Technical Detail - 1

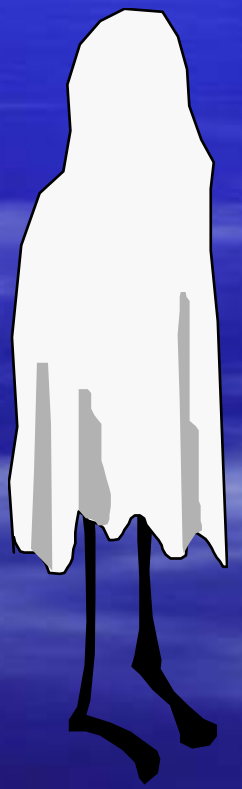
The technical details of errors that occur are typically of no interest to the end-users of a system.
If exposed to such users, this error information may cause unnecessary concern and support overhead.

- Largely non-technical user community
- Detailed error information is
  - Not useful to them and…
  - … scary
- End up with bad reputation for application

Hide Technical Detail - 2

Implement a standard mechanism for reporting unexpected technical errors to end-users…in a consistent way at a level of detail appropriate ....

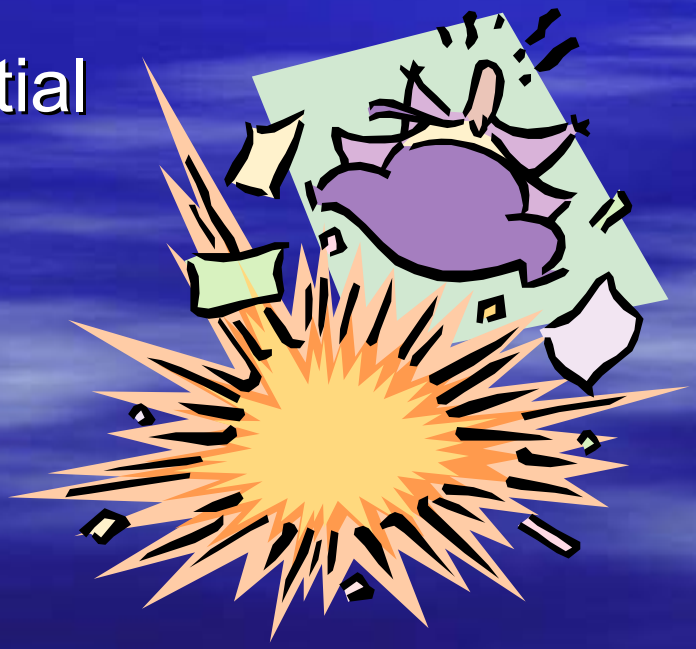
- Provide simple API to encourage consistent use
- Log technical detail for support staff
- Provide meaningful message to user
  - Make it clear that it is not their fault
- Automate error reporting where possible



Log Unexpected Errors - 1

Much domain code includes handling of exceptional [business] conditions …If such routine error conditions are logged, this makes real errors requiring operator intervention difficult to spot.

- Domain processing with potential for business failure conditions
  - E.g. product search failure
- If business errors are logged then logs fill up quickly
- More logging code reduces maintainability



Log Unexpected Errors - 2

Implement separate error handling mechanisms for expected and unexpected errors. Error conditions that are expected to arise in the course of normal domain processing should not be logged but handled in the code or by the user.

- Log unexpected errors applying other patterns
- Error logs are smaller and clearer
- Logged errors are worth investigating by support team
- May still record business ‘errors’ for different stakeholders
  - Business analysts and development team (usability)
  - Security department (e.g. logon failures)

Make Exceptions Exceptional - 1

…languages include exception handling facilities …However, if exceptions are used to indicate expected error conditions …calling code becomes much more difficult to understand.

- Domain errors are expected but unexpected errors will always occur
  - E.g. from bad configuration
- Large number of exceptions is a problem
  - Especially checked exceptions
- Don't warp code to handle expected errors as exceptions

Make Exceptions Exceptional - 2

Indicate expected domain errors by means of return codes. Only use exceptions to indicate runtime problems such as underlying platform errors or configuration/data errors.

- Reserved return value for standard error conditions
  - E.g. empty list
  - Handle in code flow
- Raise exceptions for unexpected conditions
  - E.g. platform, configuration or data problems
  - Handle with specific processing
- Semantics/expectations of method call are important here
  - FindXXX vs RetrieveXXX

Summary

- Aspects of distributed business systems bring their own issues for error handling
- Need coherent set of solutions

- Paper reviewed at EuroPLoP 2004
- http://www.blueskyline.com/ErrorPatterns
- Also Focus group “Design for Maintenance - Maintenance by Design”