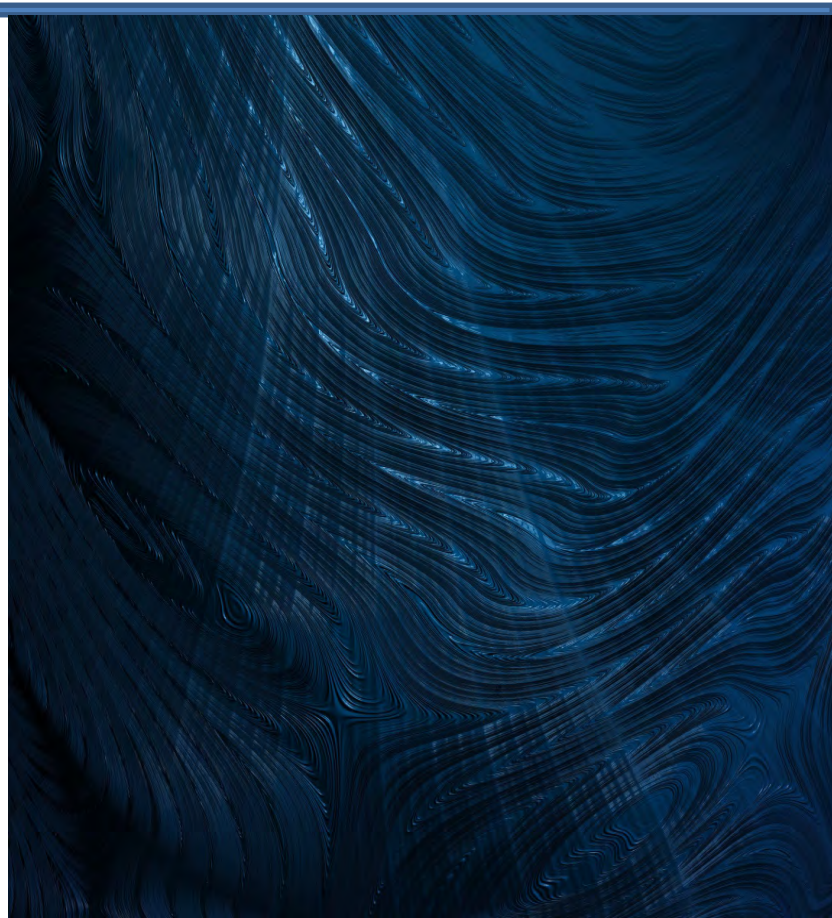
ASE Anti-Patterns

Joe Woodhouse
15 September 2021

PRIMA DONNA CONSULTING

Contents

- About the author & this presentation
- Patterns and anti-patterns
- Some Sybase-specific anti-patterns



About the author

- Sybase Australia 1996 – 2003
- Freelance consultant via Prima Donna Consulting for 18+ years
- Works exclusively with ASE, IQ, and Replication Server for 25+ years
- Based in London, UK and Melbourne, Australia
- International Sybase User Group Board of Directors since 2010
- UK Sybase User Group Board of Directors since 2019
- Not a lawyer – no charge for emails!

Why this presentation?

- I was taught many things that have not stood the test of time
  - May have made sense in the 1990s, but not in the 2020s
- As a consultant I see the same mistakes again and again
  - Not only being made by my peers
  - But also taught to the next generation
- It's not just DBAs and Database Engineers prone to this
  - Your storage, network, and sysadmin colleagues inherit their own anti-patterns
  - … many of which affect us at the database layer

Design Patterns (1 of 3)

- The idea originated in (bricks & mortar) architecture
- A design pattern is a reusable approach for recurring design problems

> Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

Christopher Alexander, *A Pattern Language*

Design Patterns (2 of 3)

- This does *not* mean templates, or even necessarily re-use of solutions
- First understand why a scenario or situation causes problems
  - Particularly when there is tension between competing goals
- A design pattern …
  - … doesn't give the answer
  - … gives the approach to find a good answer
  - … is based on values and constraints to find the "best" answer for a scenario
  - … reduces toil by preventing the need for repeated analysis from first principles

Design Patterns (3 of 3)

- Example: should my new building's floors be concrete, wood, carpeted?
- There is no one size fits all answer
- A design pattern is an approach to follow to arrive at a good answer
- Any good answer must at a minimum consider:
  - Is this a residential home? A commercial office? A farm house?
  - How many people and what kind of foot traffic?
  - Cost/budget constraints
  - Location and its climate
  - My values and aesthetics
  - Planning approvals and conditions

Anti-Patterns (1 of 6)

- Easy to say "opposite of a design pattern"
- But what does that even mean?
  - Increases toil
  - Does not understand tensions between competing goals and constraints
  - Imposes one size fits all
  - If it arrives at a good answer it is despite the pattern not because of it

Anti-Patterns (2 of 6)

- Anti-patterns were (maybe) a design pattern one day
  - And perhaps failed to change as the world changed
- More likely: they were never actually a design pattern
- Perhaps born from noble causes
  - Standardisation attempts: tame the chaos
  - Reusable solutions (rather than reusable approaches for finding a solution)
  - Runsheets for L1 call centre staff or L2 "factories"
- Sometimes it's just what we learned without questioning it

Anti-Patterns (3 of 6)

- Wait a minute
- Many of these anti-pattern concepts feature heavily in:
  - Software development methodologies and lifecycles
  - DevOps
  - Automation in general including CI/CD
  - SRE (Site Reliability Engineering)
- Do you see the problem?
- Many old approaches are being baked into the new

Anti-Patterns (4 of 6)

specific repeated practices in software architecture, software design and software project management that initially appear to be beneficial, but ultimately result in bad consequences that outweigh hoped-for advantages

Brown, William J.; Malveau, Raphael C.; McCormick, Hays W. "Skip"; Mowbray, Thomas J. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*

Anti-Patterns (5 of 6)

- Not every bad answer is an anti-pattern! An anti-pattern must
  - be commonly used, taught, and widely documented
  - be a structured process or action
  - initially appear appropriate and effective
  - ultimately have more negative than positive consequences
- Even then there must also be an alternative that is
  - documented
  - repeatable
  - proven to be actually effective

Anti-Patterns (6 of 6)

- A high-level example: IT project management death march
- A project known to be doomed to all but the decision makers
- Usually involve some or all of
  - Unrealistic or even deceptive claims for benefits; impossible promises
  - Overly optimistic estimates and scheduling
  - Feature/scope creep
  - Cuts to testing, training, and documentation
  - Throwing bodies at the problem combined with massive burnout
- And yet it could have succeeded with honest competent management

Bringing it back to Sybase (1 of 1)

- Time to get Sybase-specific
- Some of these issues have been in the Sybase world longer than I have
- No finger pointing
- We did the best we could at the time within the constraints:
  - Budget (money)
  - Budget (time)
  - Best knowledge available
  - Technology of the day

Things database people probably can't do much about

When do you do your changes? (1 of 3)

- What day and time do you schedule your changes?
  - Assume business hours are M-F, 9-5
- Friday 6pm? Saturday 6am?
- Deployment is when you *really* test your changes
  - See later slides re. the costs of UAT not being a clone of PROD
- But changes not fully exercised until placed under real PROD load
- … which is Monday morning when everyone starts their day
- 1+ day(s) to diagnose, 2+ days to fix
- Outage of three business days

When do you do your changes? (2 of 3)

Where's the anti-pattern?

- Intended to minimise PROD outages, instead increases PROD outages
- Optimising for the best case rather than the most likely case
- The decision most likely to actually cause the most outage days is rewarded as conservative risk-management

When do you do your changes? (3 of 3)

Joe's tips:

- Schedule your changes for 6pm *Thursday*
- You'll fully/properly test your change Friday morning at start of work
- When things go wrong you spend Friday diagnosing
- Now have the weekend available to fix
- Now 1+ day(s) to diagnose + 2+ days to fix = one business day outage

UAT vs. PROD (1 of 3)

- How many of us work in sites where UAT matches PROD?
- No, really - *identical*
  - VM vs. bare metal
  - Same purchasing/provisioning cycle
  - Identical hardware down to chip level
  - Identical firmware
  - Identical patches
  - Clustered vs. standalone; disk replication; DR/standby
- Seems rare these days
  - But this means nothing is *fully* tested until deployed into PROD

UAT vs. PROD (2 of 3)

Where's the anti-pattern?

- Something intended to prevent surprises in PROD now causes them
- Why have UAT at all?
  - Intended to save money (prevent PROD incidents & outages)
  - But is now compromised in the name of saving money
- The decision that causes PROD surprises and outages is probably rewarded as conservative risk-management

UAT vs. PROD (3 of 3)

Joe's tips

- Set the expectation that if UAT is not identical to PROD, then some issues will only be discovered once deployed into PROD
- Learn how to speak senior manager language:
  - VM UAT may seem like it is saving money compared to bare metal
  - But how much does even one hour of business hours outage cost?
  - How many issues were there in the last year because of this?
  - What did those therefore cost?
  - Cloning PROD for UAT pays for itself likely in 1-3 days of prevented outages

Storage (1 of 5)

- Storage architecture is now mostly centralised and one size fits all
- Not necessarily an anti-pattern, but too much is unquestioned
- File servers vs. databases need very different I/O tuning
- Some storage performance & tuning can't be done at all on a SAN

Storage (2 of 5)

- RAID level: cost vs. availability vs. performance: pick two
  - RAID 5 probably still the most common: cost and availability
    - In many ways the worst possible for database performance, penalty to writes
    - RAID 6 even worse!
- RAID striping size / block size
  - Too small = more I/O than needed; too big = losing benefits of striping
  - SAN used for databases should closely match database block size
  - This should be 2 x ASE page size, or some integer multiple of this number

Storage (3 of 5)

- SAN by definition is on the network
  - All disk I/O is therefore now also network I/O
  - SAN traffic competes with all other network traffic unless isolated
  - Has O/S been tuned for network performance?
- SAN replication common, but not needed for many ASE devices
  - Almost certainly not for tempdb
  - Probably not for any staging or scratch databases
  - Probably not for any archive/historical copies

Storage (4 of 5)

Where's the anti-pattern?

- Centralisation & consolidation lose the opportunity for bespoke tuning
- Have cost savings been analysed (today, not when decided)?
- Have costs of *not* fully tuning database storage for SLAs been analysed?
  - What would it be worth to achieve +10% batch performance on current h/w?
  - What would it be worth to delay next h/w refresh by one year?

Storage (5 of 5)

Joe's tips

- What would we do if designing storage for a new organisation?
- Take everything back to use cases and constraints
- Maybe SANs get their own network in parallel to the "real" network
- Maybe SANs for databases are segregated from SANs for file servers
- Maybe SAN replication is matched to SLAs *per database*
- Maybe RAID level & block size is matched to SLAs *per database*
- Maybe SANs only used when this solves problems that local disk can't

Data model (1 of 4)

- Unless developing in-house, almost impossible to change data models
  - Indexing, partitioning, datatypes
- Alas; it means we're hostage to another organisation's anti-patterns
- In-house doesn't mean we're immune to anti-patterns though
  - One Australian bank spent $80k to convert *datetime* to *smalldatetime*
  - Saved $5k in disk space
- Were you taught that every table should have a PK?
- Were you taught that every table should have a clustered index?
  - Or possibly that every table must not have clustered indexes?

Data model (2 of 4)

Where's the anti-pattern?

- Are data model decisions inherited from 10, 20, 30 years ago?
- Or are they the decisions we would make today on the merits?
- Are data model decisions proved to be beneficial?
- Is there data model maintenance to prove they are beneficial today?

Data model (3 of 4)

Joe's tips

- If using Replication Server (or ASE HADR), yes every table needs a PK
  - If not replicating, PK is not necessarily mandatory (e.g. history tables)
- Indexes should be based on solving specific problems
  - Adding indexes *should* improve response times for reads
  - Removing indexes *should* improve response times for writes
  - Either way we should prove that they helped
  - Regularly check for unused indexes (MDA, over time)
  - Don't be scared of dropping or creating indexes for just one task
  - Clustered indexes mostly useful now only for avoiding sorting

Data model (4 of 4)

Joe's tips

- Similar advice for partitions and datatypes
  - Use them to solve specific problems
  - Prove they helped
  - Ongoing checking to prove that they're still helping

Things database people probably can do something about

Patching strategy (1 of 4)

- Which is more conservative: to patch often or to patch seldom?
- Do your work policies and practices make it easy to patch? Or hard?
  - One Australian bank doesn't allow patches until proven they fix P1 incidents
  - One UK bank only applies ASE patches every 24 months
- Do you have all the OS patches required by ASE?
  - Maybe 1 in 20 sites do
  - Does this matter?
    - Usually not for stability or performance
    - It will slow down your SAP support cases if they notice you missed some

Patching strategy (2 of 4)

- How soon after release do you apply ASE service packs (SPs)?
  - These introduce new features and therefore are a risk for instabilities
  - Require significant application testing
- How soon after release do you apply ASE patches (PLs)?
  - These do not introduce new features (but may extend existing functionality)
  - Usually these are security, stability, and performance patches
  - Serious question: do these really require any application testing at all?

Patching strategy (3 of 4)

Where's the anti-pattern?

- Are patching decisions genuinely managing risks?
- Or are they instead managing patch-related inconveniences?
  - A manager might be congratulated for avoiding patch-related downtime
  - But are they tracked against P1 incidents due to unpatched ASEs?
- Even if bugs fixed aren't relevant, what are the implications for SAP Tech Support case turnaround time?

Patching strategy (4 of 4)

Joe's tips

- Set the expectation: patching CHG every month, even if not used
- Go check the manuals: do you have all the OS patches required?
- Patching strategy should be both planned and reactive
  - Planned for stability and performance patches, monthly as above
  - Reactive for security vulnerabilities, as required
- It *is* possible to achieve zero-downtime patches and upgrades
  - It requires $$$ + ability to run DR at same time as PROD + Replication Server
  - Easiest method: ASE HADR (premium license) + above

Raw partition vs. file (1 of 4)

- First we used raw partitions for guaranteed writes
- Then we used file systems + dsync for ease of administration
- Then we used file systems + directio for performance
- Vendor benchmarks prove that raw partitions is still the fastest
  - But raw is maybe only 2% faster today vs. well-tuned filesystem with cio
  - Third party f/s like Veritas Quick I/O expensive but perhaps best of both worlds

Raw partition vs. file (2 of 4)

- What is the problem we are trying to solve?
- How long does it take to get a sysadmin with root access?
- How many Sybase storage mistakes do they make?
  - "How many gigabytes in a terabyte?"
- But don't just say "performance is the goal"
  - We still probably don't want one consistent storage infrastructure for ASE
  - Even if everything else is on raw
    - Probably want tempdb on file with write cache and neither dsync nor directio
  - Even if everything else is on file with cio
    - Probably want disk dump areas on file without cio

Raw partition vs. file (3 of 4)

Where's the anti-pattern?

- What is the policy at your site?
- Which decade was it decided in?
- When was the last performance benchmark proving it was the best?
- Are we tuning for sysadmin/root inconvenience or for performance?
- Do we want DBAs to have root access?
- How many of these issues are solved by ample capacity and provisioning?

Raw partition vs. file (4 of 4)

Joe's tips:

- It's never too late to benchmark
- Ample capacity and pre-provisioned devices remove urgency for root
  - Do we care if it takes 1 week for new raw partitions if we have 3 weeks' spare?
- Treat all storage architecture like green fields
- What is the best we can build today?

UNIX mount point options (1 of 4)

- (Assuming file on UN*X) Do you know what your mount options are?
- Rarely controlled by the database people
- Usually controlled by the UNIX sysadmins
  - Who have their own anti-patterns in play
- Usually not difficult to persuade them re. queue depths & elevators
- Usually impossible to persuade them re. filesystem journaling

UNIX mount point options (2 of 4)

- UNIX file system journaling is the sysadmin version of raw vs. file
- They were taught in 1990 (AIX JFS) or 2001 (Linux ext3)
- What problem were they trying to solve? Was it a database problem?
- e.g. elevator algorithms (CFQ, deadline) try to reorder I/Os
  - ASE already reorders its I/Os (large I/Os, APFs)
  - The SAN also reorders its I/Os
  - We have three layers all doing this!
  - We can at least disable the middle layer
- e.g. Linux queue depth defaults more suited to apps than data

UNIX mount point options (3 of 4)

Where's the anti-pattern?

- ext3/ext4/xfs journaling keeps file metadata consistent with file data
- ASE fully preallocates file system devices
  - Their size will never change, file metadata is never at risk
- Proven beyond doubt that f/s journaling slows ASE writes 400%
  - The sysadmins congratulate themselves for conservative tuning for stability
  - I have literally never won this argument yet
  - Yes even with benchmarks and SAP vendor doco saying journaling isn't needed

UNIX mount point options (4 of 4)

Joe's tips

- Get a senior sponsor
- Get them to agree that tech decisions will be made on the merits
- Get them to agree to benchmark and to be bound by the results
- Provide SAP doco stating journaling is useless overhead and unneeded
- Tune Linux queue depths from 128 to 1024
- Tune Linux elevators from CFQ to noop (SAN) or deadline (local)
- Disable Linux f/s journaling!!!
  - 400% slowdown if you don't

Metadata cache (1 of 3)

- Metadata cache is routinely undersized, and it's SAP's fault
- sp_sysmon and sp_monitorconfig report all is well when it is not
- Should be sized according to sp_countmetadata
  - +10% for *open databases* and *open indexes*
  - +25% for *open objects*
  - *open partitions* should be equal to *open objects*
- Goal is to ensure we never run out, and this usually costs little
- Many downstream performance consequences if we start turning over

Metadata cache (2 of 3)

Where's the anti-pattern?

- Using the built-in diagnostics will result in possible under-sizing
- Doing the right thing results in the wrong thing

Metadata cache (3 of 3)

Joe's tips:

- Rely on sp_countmetadata
- Learn more about metadata cache internals

Monolithic code (1 of 3)

- Do you have some of these?
- A stored procedure or trigger that handles multiple disjoint code paths
  - Effectively each proc call exercises only one code path
  - if [these conditions] then [execute only this code]
- Almost always the result of incrementally extended code
- Almost always the result of multiple developers

Monolithic code (2 of 3)

Where's the anti-pattern?

- "We don't want thousands of procedures, this is cleaner"
  - "All our related code is in one place now"
- But it isn't cleaner – much harder to test and to maintain
- It's sensible to reduce unnecessary proliferation of objects
- This requires understanding what is and isn't necessary
- Very real implications for performance given procedure cache re-use
  - A query plan compiled for only one code path but will be used for another

Monolithic code (3 of 3)

Joe's tips:

- Don't do it!
- One code path per proc
- This isn't theoretical
  - One proc with four code paths = average response time 5+ minutes
  - Same code refactored into four sub-procs = average response time 1.2 seconds

ORDER BY (1 of 3)

- Anyone upgraded pre-ASE 11.5 to 11.5+? Or to 11.9.2+?
- Many assumptions about ORDER BY, usually "I don't need it ..."
  - "... because I will get my result sets in the order of the clustered index"
  - "... because I will get my result sets in the order of the GROUP BY"
- Strictly speaking these were bugs
  - ANSI standard for SQL says there is no reliable row order without ORDER BY
- Also remember DOL clustered indexes aren't really clustered any more
  - Like the Pirate Code (more a guideline than a rule)

ORDER BY (2 of 3)

Where's the anti-pattern?

- Official fix from Sybase: "always use ORDER BY"
  - Unstated: "... if you care about row order"
  - Often we don't actually care
- What happened? Many unnecessary ORDER BY clauses added
  - Not because of any genuine business requirement, but to avoid "regression"
    - Results were ordered pre-11.5 even when this wasn't needed
    - No longer sorting looks like a bug... even when sorting wasn't needed
- Sorting is expensive & costs many resources

ORDER BY (3 of 3)

Joe's tips:

- Use ORDER BY every time you need it, and never when you don't
- The goal is not always to prevent regressions
  - If original code was wrong then we want a "regression"
  - (A "progression"?)

Running out of paper (1 of 3)

- This is my name for trying to reduce number of lines of code
  - Also "rationing newline characters"
- The compiler doesn't care about whitespace
- The compiler *does* care about nested conditionals
- Which is faster?

```
if (a=10) and (b=20)
```

```
if (a=10)
    if (b=20)
```

Running out of paper (2 of 3)

- Where's the anti-pattern?
- Have you ever heard this?
  - "Clean code can be viewed on a single page/screen"
- Many code languages use "lazy AND" and "lazy OR"
  - Stop processing AND clauses as soon as you find the first FALSE
  - Stop processing OR clauses as soon as you find the first TRUE
- ASE T-SQL *does not do this!*
  - All compound conditionals fully evaluated, every time
  - Compound conditional is always slower than nested

Running out of paper (3 of 3)

Joe's tips:

- Always use nested conditionals, not compound
  - This emulates lazy-AND and lazy-OR
- Write the cheapest conditionals first
  - Goal is to only execute the more expensive conditions as a last resort

DBA housekeeping (1 of 4)

- We all (should) know the DBA housekeeping required
- Consultants like me have been urging you to run it better & more
- This has not always produced happy outcomes
  - Have your large housekeeping jobs ever exceeded the available window?
- Two main responses
  - Live with it (impact to PROD business hours)
  - Scale back housekeeping (impact to disk space and query performance)

DBA housekeeping (2 of 4)

Where's the anti-pattern?

- Insistence on running it can cause what it is designed to prevent
- Insistence on proving no regression vs. old methods
  - Do we care if reorg takes 2 hours longer if there is no impact on PROD?
- Neither of the two main responses are good solutions
  - Housekeeping needs to be done, it can't be ignored
  - And so much of it is avoidable these days in ASE 16.0.x

DBA housekeeping (3 of 4)

Joe's tips

- Run faster
  - Plenty of ASE and O/S tuning can be done
- Run smarter and leverage modern features in ASE
  - Concurrency
  - reorg rebuild … online
  - create index … online
  - create index … with statistics …

DBA housekeeping (4 of 4)

Joe's tips, continued

- Don't run it at all (in PROD)
  - Dump & load to offload some housekeeping from PROD
    - Dump PROD, load into DR or UAT, run update stats, optdiag stats out & then back in
  - Use replication or HADR
    - Never do housekeeping in PROD
    - Complete in PROD2
    - Switch PROD & PROD2 when done
    - Apps point to PROD2, housekeeping now in PROD

Spaghetti code (1 of 3)

- Were you taught to avoid spaghetti code?
- It's listed as a textbook code anti-pattern everywhere online
- But! Consider the GOTO statement
  - Fully documented and supported for ASE (it's in every Reference Manual)
  - Used wisely & appropriately it can save a lot of code
  - Exceptions and error handling
- Closely related yet seldom seen: the BREAK command

Spaghetti code (2 of 3)

Where's the anti-pattern?

- Attempt to avoid an anti-pattern can itself be an anti-pattern!
- Attempt to "make code more readable" can achieve the opposite
- These are usually good rules to follow
- They can be good rules to break

Spaghetti code (3 of 3)

Joe's tips

- Could your spaghetti code be improved with BREAK or GOTO?
- Are you following any other code rules from the 1990s?
- Are they still good rules today?
- Will they still be good rules tomorrow?

Other ideas, briefly (1 of 1)

- Row by row vs. set-based SQL
  - Set-based SQL always better… but maybe not for replication
- What is the real bottleneck
  - Response time vs. concurrency
  - Cache turnover vs. cache volatility
  - Are we tuning the right thing?
- Schema & code conventions
  - We don't have name 30-char name limits any more

Q&A, and thank you

Joe Woodhouse
joe.woodhouse@primadonnaconsulting.com

Joe is a freelance consultant available through Prima Donna Consulting and can be engaged full-time, part-time, or through split delivery.

Minimum engagement of one day; no premium for after hours, weekends, or public holidays.

Would you like this or another of Joe's papers presented to your workplace? Recent topics:

- ASE Tips & Tricks
- ASE Tempdb performance & tuning
- ASE Memscale – use cases & benchmarks

PRIMA DONNA CONSULTING