# Towards a Performance Pattern Language

## Results of the EuroPLoP 2001 Design Fest

Klaus Marquardt, Dorothea-Erxleben-Straße 78, 23562 Lübeck, Germany
email: pattern@kmarquardt.de


## Motivation

Every software engineer has used techniques to increase the execution speed of the system under development. Some of these techniques are purely technical, i.e. using a more efficient algorithm, others focus on correct distribution of tasks among different system parts, on management and development process, or on feature avoidance. Engineers working in different areas such as databases, embedded systems, or distributed systems, are used to apply specific techniques that are apparently different to techniques used in other domains. Nevertheless, we have found that these techniques parallel each other to a high degree.

The purpose of the Design Fest was to work towards a Performance Pattern Language.

## Participants

Alexander Horoshilov, Christian von Mueffling, Ansgar Radermacher, Amir Raveh, Andreas Rüping, Dietmar Schütz, Oliver Vogel, Makrus Völter, Sherif Yacoub

## Results

We collected a number of pattern names, pinned them down, asked the contributor for clarification (problem and solution), and grouped the pattern names. We managed to cover a wide range of performance aspects, from embedded systems to distributed, database oriented and other technical systems, and to design policies and management practices. While trying to sort the patterns, we found a number of common forces. We grouped those pattern names together that deal with a common topic. This topic made a headline of the group. The groups are still not independent on each other, a lot of patterns seem to belong to a number of them. More thoughts are necessary to find a meaningful classification. This is a first indication for the original claim of the Design Fest, that the patterns cross different domains. Future workgroups will gather more knowledge and substance here.

**Forces**

- Simplicity
- Local versus global optimum
- Resources - files, handles, bandwidth - all resources are finite
- Not all resources are equal
- Speed zones - autobahn/ inter-city/ urban
- Different paces of change/ maturity (infrastructure, environment, subsystems)
- Resource management: plan <-> tune

**Ordering criteria**

- Availability
- Reliability
- Portability
- Testability
- Time
- Designing
- Coding
- Process
- Scope
- Domain specific
- Common among domains

The appended list contains all pattern names, sorted by their preliminary group, plus a brief description of problem and solution. Where known, a contributor is given for further reference.

## Acknowledgements

I would like to thank James Noble and Charles Weir for their work on Patterns for Managing Limited Memory. As a workshop participant on EuroPLoP 1998, their contribution motivated me to collect performance patterns and initiate this Design Fest.

Thanks to the authors whose patterns we cite, and apologies to everybody whose patterns we have missed.

## References

*GM1996* Gerard Meszaros: A Pattern Language for Improving the Capacity of Reactive Systems. In: PlopD-2

*KM1998* Klaus Marquardt: Patterns for Software Installation and Activation. In: Proceedings of EuroPLoP 1998

*NW1998* James Noble, Charles Weir: Proceedings of the Memory Preservation Society. In: Proceedings of EuroPLoP 1998

*NWB2000* James Noble, Charles Weir, Duane Bibby: Small Memory Software: Patterns for Systems with Limited Memory (Software Patterns Series). Addison-Wesley

| Name | Problem - Solution | Remarks |
|---|---|---|
| **Development Process** | | |
| BUDGETS | Assign a performance budget to individual tasks or functions. Monitor these budgets.<br>Your architecture sets up budgets of CPU load, IPC, ISR, etc. Each developed component must stick to this budget. Violation of budgets give you early escalation options.<br>And maybe developers could even trade their budgets among their peers… No I have not seen this, but some people believe it should work this way with natural resources. | [*NW1998*]. Relates to PERFORMANCE EVALUATION |
| THINK SMALL | Change your development habits. Make yourself think that the system is even smaller and less responsive than it actually may be. This prevents failure from false assumptions. | [*NW1998*] |
| DECOMPOSITION OF PERFORMANCE | | |
| VALIDATION OF PERFORMANCE | How can you measure time performance?<br>When assessing a software system, time performance is often considered to be an important criterion. However, equally often, it is unclear how fast exactly the software must be to be acceptable.<br>Therefore, define performance quantifiable criteria, that is, criteria which can be measured in numbers, so it's clear whether they're met or not. For example, the response time of an interactive system is often supposed to be under 1 second. | |
| FINE-TUNING | How can the time performance be increased at the end of the development process?<br>Performance needs to be cared for from the very early stages on. However, fine-tuning can only be done once the implementations details are clear: the database model, access functions, the frequency in which these functions are called, etc.<br>Therefore, plan for a fine-tuning stage at the end of development cycle. Typical fine-tuning activities at the end of a project include the definition of elaborate database indexes. | |

| Name | Problem - Solution | | Remarks |
|---|---|---|---|
| THE RESOURCE / BUDGET MISER | We know all resources - files, handles, bandwidth, CPU power, currrent consumption, etc. - are finite.<br>It is very easy for developers to assume they are not, or design a sub-system that can hog resources that are needed for other sub-systems.<br>How can you protect the system from a sociopath, desperate or the ignorant? | Assign a person to hold the resource budget in his pocketbook. Make sure the person has Moliere's L'Avare as the role model. Any allocation from the resource budget must be approved by the RESOURCE/BUDGET MISER. Who is more likely to wrestle it from someone else's budget, rather than assign some of the RESOURCE SPARE.<br>Sample resources are: MIPS, CPU utilization, current consumption, real-time clock quanta, memory footstamp, message queues.<br>Provide a balance for ALL RESOURCES ARE FINITE.<br>Also known as THE TIME/MIPS KEEPER | The organisational role related to BUDGETS |
| **Requirements** | | | |
| FIGURING THE 80% TYPICAL CASES IN REQUIREMENTS ANALYSIS | How can the time performance in the average case be increased?<br>Requirement analysis often states that a software system must meet certain time performance criteria. However, though often unspoken, it is fine when these criteria are met only in the most typical use cases.<br>Therefore, make sure that the software is fast in enough for the typical use cases, typically in 80 percent of all cases (the 80:20 rule).<br>In relational databases, de-normalisation is often the method of choice to achieve this goal. For instance, think of a 1:n relation in a data model, pointing from an insurance contract to all insured persons. Given the contract number, you need to know the insured persons quickly. Typically, you store a pointer to the first ten insured persons with the contract, and accept the fact that the system slows down when it has to deal with contracts for families with more than 10 people. | | |

| Name | Problem - Solution | Remarks |
|---|---|---|
| PERFORMANCE AS A FUNCTIONAL REQUIREMENT | Treat performance as a functional requirement. Define acceptance criteria and test cases. | Relates to BUDGETS |
| ANALYSE INTERACTION / NECESSARY RESPONSE TIME | | |
| **Management** | | |
| DELAY DELIVERY | Delay the product delivery when the performance is insufficient, and take the time for technical measures to improve performance. | Relates to PERFORMANCE AS A FUNCTIONAL REQUIREMENT |
| TRADE-OFF PRIORITY DRIVEN | Trade performance against other criteria for your project, like system extensibility development cost, development time, or hardware cost. Base this trade on explicitly given priorities.<br>Responsiveness is a highly visible quality; Resources are limited, both time and cost<br>Invest a certain amount of other resources for the benefit of the quality responsiveness. Apply one or more of the more specific trade-off techniques. Evaluate your CAPACITY BOTTLENECK [*GM1996*] to identify where other resources are well invested.<br>To keep the development focused, project owners must give clear priorities which aspects of the project are important. An ordered list is a great help, if it is agreed upon all project stake holders.<br>Your system becomes more responsive, but it will be more expensive, or be delivered to market delayed. Alternatively, other qualities like portability or maintainability might suffer. | Principle to other concrete patterns. CAPACITY BOTTLENECK describes how to deal with limited resources, and weight between memory, processing capacity, and messaging capacity. |

| Name | Problem - Solution | | Remarks |
|---|---|---|---|
| PHASED DELIVERY | Customers want your product to do everything, and they want it now.<br>Managers on the other hand want you to use as few development resources for the shortest duration possible.<br>Everyone wants your product to meet schedule, quality and performance goals.<br>Oh, and they all want it yesterday…<br>How can you satisfy all these opposing constraints? | Not all functionality needs to be delivered on the same date. In most cases, it is more feasible to divide product delivery into phases, each phase providing additional product functionality.<br>Examples are providing a growing set of features, adding more interfaces in each phase, growing in complexity of interfaces incrementally.<br>A side benefit is that everyone has better understanding of "what it *really* takes to make it work" as you progress through delivery phases. | |
| ADD HARDWARE | Instead of squeezing the software for processor cycles, choose a faster hardware. In distributed systems, assign physical hardware to more of the logical layers.<br>This is an implementation of TRADE-OFF PRIORITY DRIVEN | | |
| ST. FLORIAN'S PRINCIPLE | Performance commitments are a risky business – once you commit to performance, you have a measurable goal to meet, while your ability to meet that goal is usually very "fuzzy"…<br>How do you reduce the performance risks in your project? | Also known as S.E.P. - SOMEONE ELSE'S PROBLEM<br>Define you projects' scope in a way that performance is not a problem of your duties. Take care that all performance issues occur in a different (sub-)project, which is by far more critical or visible with respect to performance.<br>Be aware that your boss may appreciate that, but your peers might not want to cooperate in the future. | Seen that one too often. BUDGETS can even help you to establish the S.E.P. |
| DO IT YOURSELF | Do not use operating systems, databases, or libraries. Develop a minimal set of functions that you need, with less functionality and a narrow focus on performance. | | |
| IGNORE & CONCEAL | Define you projects' scope in a way that performance is no problem. Leave all performance issues out of the contract, and stay far away from real users. | | |

| Name | Problem - Solution | Remarks |
|---|---|---|
| **Prototype** | | |
| PERFORMANCE EVALUATION | Develop the product, and check the performance on a regular base. Decide about performance related measures on there results.<br>Develop first, then care for performance. In the unlikely case that the performance turns out to be insufficient, you still have time for counter measures – the time that you have saved while not caring for performance issues. Now you start profiling and optimizing a hopefully well designed and understandable code. | [*NW1998*] |
| PROTOTYPING | How can you can make sure time performance is addressed early enough in the development process?<br>Based on a mere design it is terribly hard to make performance estimations. However, it is relatively early that you need to know about the time performance you can expect from the system you're building — so early that you are still able to make corrections to the software architecture in order to increase the time performance.<br>Therefore, use a prototype for measuring the time performance. Be aware that the prototype won't give you precise details about the time performance of the system you are going to build. But the prototype will give you a good idea of the order of the time performance you can expect. | |
| **Interfaces** | | |
| DATA TRANSFER OBJECT | Instead of calling several remote operations with one parameter or result, call one operation passing a serializable DATA TRANSFER OBJECT, which contains the original arguments as attributes. Works also for return values (which BULK INTERFACES do not!) | Relates to BULK INTERFACES. Also known as VALUE OBJECT, but that name is already occupied by another pattern (Riehle) |

| Name | Problem - Solution | Remarks |
|---|---|---|
| BULK INTERFACES | In distributed Systems, instead of calling 10 operations with one argument, call 1 operation with 10 arguments. Related to DATA TRANSFER OBJECT. | |
| AVOID NETWORK TRAFFIC | Reduce the necessary network traffic. Here are a number of implementation patterns lurking:<br>APPROPRIATE NETWORK - You could choose an appropriate network; sometimes ethernet is not the optimal choice for your system.<br>COMBINE PACKETS – You can adopt your message size to your network's preferred packet size, and combine multiple messages into a single packet.<br>COMPRESSED TRANSFER - You could pack your message for transportation<br>USE ENUMERATIONS – In most cases, your own protocols do not need to transfer strings, and you can avoid semantical overhead.<br>CACHING – see DATA TRANSFER<br>FIXED TIME TRANSFER – You can reduce the traffic when you delay the transfer of available data or events until their sending time has come.<br>CODED CONVENTION - You can reduce the amount of sent data by convention, when both transmission partners rely on the same code and do not need to synchronize about their basic objects. | CACHING is a global principle for a large number of patterns |
| DELIVER / TRANSFER BULK DATA | | Similar to DATA TRANSFER OBJECT? |
| REDUNDANCY | | Global principle |
| MINIMIZE INTERACTION | | |

| Name | Problem - Solution | Remarks |
|---|---|---|
| **Visible Behaviour** | | |
| INTERRUPTIBLE COMMAND, KILL | Give the user the opportunity to interrupt a process that he once initiated, when its execution exceeds a defined or perceived time, or when system performance degrades on other processes. | [*NW1998*] |
| SUICIDE | A system supervisor process kills tasks that violate their performance budget.<br>Optionally, you may want to inform the system user.<br>Known in embedded systems, and on web servers. | Relates to BUDGET, is a non-interactive version of KILL |
| SACRIFICE THE DEFENCELESS | Sacrifice that part of your system that seems to have the least influence on perceived performance. The main criteria are user events and real time scheduling. If there criteria are absent, the process may be as well.<br>This is actually what some schedulers do unintentionally. Processes that do not register for system events but make some long term calculations, tend to starve in highly reactive environments.<br>Can be implemented as a variant to SUICIDE, where another less relevant process is sacrificed first. | Related to SUICIDE and GRACEFUL DEGRADATION |
| MAKE THE USER WORRY | Tell the user what is wrong. He may decide to kill you, sacrifice someone else, or shed the load. Or he may just accept the current state. | [*NW1998*] |
| ILLUSION OF FEEDBACK | Did you ever see an hourglass on your computer? Or a line of dots? | |
| GRACEFUL DEGRADATION | Also known as PARTIAL FAILURE [*NW1998*]. Measure your system load. When performance degrades, unload some expensive pieces of code and omit some calculations that serve mere convenience reasons. You could also MAKE THE USER WORRY [*NW1998*] whether to degrade | [*NW1998*] |

| Name | Problem - Solution | Remarks |
|---|---|---|
| **Deployment Structure** | | |
| DISTRIBUTE RESPONSIBILITIES (N-TIER ARCHITECTURE) | | |
| FAT CLIENTS | Fat Clients take the load of selecting display items and preparing the clients screen display away from the server. | |
| THIN CLIENTS | Thin Clients take the load of data selection and display preparation away from the client. Also known as TERMINAL | |
| SELECTABLE SCALABILITY | | |
| SHED LOAD | When your local system can not provide you with the resources you need, maybe some other system / node / processor can. Establish a load balancing protocol and shed load among peer processors.<br>This seems to require a THREAD PER TASK model, the same code on all nodes, and an additional administrative protocol. | [*GM1996*] |
| LOAD BALANCING | | |
| BUS (BROKER) | | |
| **( Ungrouped )** | | |
| SKIP A LAYER | In Layered Systems, you may intentionally skip a layer to avoid the overhead and improve performance. As a last resort only – usually there are many much better options. | |
| PROACTOR REACTOR | | |
| THREAD PER TASK | Each functional task, like a client connection, is executed within its own context (thread). This is the preferred model in highly event driven systems. | |

| Name | Problem - Solution | | Remarks |
|---|---|---|---|
| SCHEDULING | Different scheduling mechanisms, depending on the thread priorities, serve different needs. Among them are THROUGHPUT SCHEDULING; REALTIME SCHEDULING; STEADY BEAT SCHEDULING; REMAINING TIME SCHEDULING | | |
| TRUSTED ~~COMPUTING~~ BASE | Your system has reliability goals.<br>Your code contains an unknown number of bugs, yet to be found. And it keeps changing…<br>How reliable can your system be? | To provide a reliable system, the foundations used must be very stable in their reliability level. These foundations may include the platform's operating system, the database engine used, the messaging libraries and code base chosen, but may also cover hardware components.<br>The number and scope of such foundations are kept as a balance between the opposing forces of "as many as possible, covering as much of what we need to do" and "any change introduced in foundations must be thoroughly tested before it can introduced into MY product". | |
| BUILD CLUSTERS | | | |
| **Custom Resource Management** | | | |
| INTERFACE IS NOT WITHOUT OVERHEAD | Many people view partitioning a system into subsystems, with well defined interfaces between them as a solution to making development and testing manageable.<br>Where do you draw the line? | Any interface added to a system has a price tag associated with it – including the design effort, through coding, testing, customer meetings, processing overhead, resource utilization, and product maintenance. | |
| CACHING | Store some already retrieved data in a fast access place. Subsequent read accesses can be faster than the initial retrieval – provided the source did not change in the meantime. Thus, caches must be refreshed frequently or according to another policy. A "transparent" cache implementation requires significant development effort, all calls must be redirected. | | CACHING is a basic principle for several patterns, based on REDUNDANCY |

| Name | Problem - Solution | Remarks |
|---|---|---|
| INSTANCE POOL | After you have allocated memory, channels, whatever (…) once, do not release them after use but pool them for later usage – at which the allocation time vanishes. Trade-off against development effort and a higher memory load.<br>Also known as REUSE INSTANCES, AVOID ALLOCATION, POOLING. | Implementation of CACHING |
| VIRTUAL INSTANCE: POOLING, PASSIVATION | Let clients communicate with a virtual instance represented by a Proxy – instantiate the real object only when needed, and use POOLING or PASSIVATION to get rid of temporarily unneeded instances.<br>This is most effective for resources that are expensive to acquire, so that the additional indirection and administration effort pays off. | |
| OS MECHANICS | Employ the facilities of your specific operating system like shared memory, queues, messages, page locks, etc. Do not try to develop an OS independent solution when your OS provides unique or superior mechanisms with respect to efficiency.<br>This is an explicit trade off against other qualities such as portability. | Implementation of TRADE-OFF PRIORITY DRIVEN |
| MEMORY MANAGEMENT | | |
| LOCK PAGES | When your operating system supports paging, you may prevent undesired paging by locking specific pages where a particular functionality must be very responsive. | Implementation of OS MECHANICS |
| MANAGED RESOURCE, PREALLOC | You can reduce the time to acquire a resource (such as a database connection) by preacquiring them, and storing them in a pool. When a client needs one, it's not acquired by merely taken from the pool – this should be much faster. | Relates to ADVENT and POOLING. |
| **Compression** | | |
| USE COMPRESSION | Reduce storage requirements or save bandwidth/reduce transmission time if data needs to be retrieved via a network or bus. The price to be paid is the additional overhead for compression and decompression code as well as the required processing power to perform one of these actions. The former (compression) might not be necessary if data could be compressed in advance, relates to ADVENT. | |

| Name | Problem - Solution | Remarks |
|---|---|---|
| AVOID COMPRESSION | When the compression of your data is more time consuming than the additional infrastructure overhead that larger data causes, avoid compressing your data. This is more likely for very small data items. | |
| **ISR** | | |
| SHORT ISR | Keep interrupt service routines very small. If necessary, transfer a state marker to the normal flow of execution and let a more complex evaluation or calculation follow there. | |
| TRAMPOLINE | Interrupt service routines need to balance the need to respond in a timely manner to stimulus, and the time it takes to process the delivered data.<br>Spend too much time in processing, and you hog the CPU, sacrificing both other tasks' response time and the ISR's ability to respond to the next stimulus (resulting in potential loss of the next accompanying data).<br>How can you achieve a balance?<br>Quickly service an ISR by collecting whatever data/event it offers, and defer any processing to a processing queue by a more delay tolerant mechanism.<br>This way data overruns are kept minimal. | |
| **Database** | | |
| DE-NORMALISATION | Your initial database schema most likely adheres to some normal form, i.e. you keep distinct items in different tables, and avoid redundancy. This can lead to very small rows and to complex joins. When you explicitly violate the normal form where joins are most expensive, the retrieval time can be improved by orders of magnitude. | |
| SMALL TRANSACTIONS | Keep your transactions limited to a few objects / records. When only few instances are locked at the same time, the throughput can be increased especially in concurrency situation.<br>The implementation depends heavily on your LOAD PROFILE and your workflow scenarios. | |

| Name | Problem - Solution | Remarks |
|---|---|---|
| PHYSICAL DB DESIGN | The physical database design determines where which parts of the logical database are physically located. A sophisticated allocation of physical resources can improve system performance significantly.<br>There are a number of patterns waiting to be discovered… like KEEP UNRELATED OBJECTS ON DIFFERENT PAGES. | |
| USE DB MEANS | | |
| DATABASE INDEXES | | |
| **Change Resources Dynamically** | | |
| TRANSFER / COPY ON BOOT | When your system starts, you copy frequently accessed code or data into a memory area that has faster access than your mass storage.<br>For example, RAM has faster access cycles than ROM, so all items that are stored in the ROM can be loaded into RAM during initialization.<br>This is an application pattern of the ADVENT pattern.<br>Variant: DECOMPRESS ON BOOT, application of USE COMPRESSION | Implementation of the REDUNDANCY principle and the CACHING pattern.<br>Variant by Ansgar. |

| Name | Problem - Solution | Remarks |
|---|---|---|
| ADVENT | Weigh the different times when a necessary task can be done: during installation, during loading, initialization, or execution. Implementation patterns are:<br>• PRE-INITIALISE DATA - Can be done during loading or initialization. Relates to COPY-ON-BOOT, and MANAGED RESOURCE, PREALLOC<br>• PRE-POPULATE DATABASE - Most applications need some records in their databases to operate smoothly, like Null Objects [*BW1998*], default configurations etc.<br>• PRE-LOAD CODE – load dynamic libraries during idle times. Modern CPU's use their cache in a similar way.<br>• EAGER CREATION - create objects, connections etc. in advance to avoid latency on usage<br>• LAZY CREATION – create objects, connections etc. no earlier than necessary<br>• LATE LOADING (AUTOLOADING [*NW1998*]) – load dynamic code parts only when really needed | [*KM1998*] |
| LAZY CREATION | Construct objects, connections etc, not earlier than necessary. You might not need them in the end, and in that case your application is more responsive.<br>Do not apply LAZY CREATION in systems that can not deal with a somewhat arbitrary load during run time, like real time embedded systems.<br>This is an application pattern of the ADVENT pattern. | |
| LOAD PROFILES | Determine your system load in relation to the tasks or requests that the systems handles in specific situation. Start with basic scenarios of use, find out how frequently they occur, and what kind of load the functionality causes.<br>A knowledge of your LOAD PROFILES helps you in PERFORMANCE EVALUATION and in determining which branch of the ADVENT pattern you apply in which situation. | Relates to LOAD PROFILES, PERFORMANCE EVALUATION, and ADVENT |
| MEASURE CUSTOMER INTERACTION | | |
| VIRTUAL PROXY | | |

| Name | Problem - Solution | Remarks |
|---|---|---|
| VARIABLE INIT | | |
| SHADES OF PERFORMANCE | Complex system have different performance requirements in different system parts. Where a delay on the display may be merely annoying, a delay in machine control can become costly or even deathly.<br>Determine which parts of your system needs which shade of performance. Helpful categories are<br>• Hard realtime (e.g. machine control, typically milliseconds, costly or deathly consequences)<br>• Soft realtime (e.g. machine display, typically milliseconds, annoyance as consequence)<br>• Interactive (e.g. display, typically seconds, annoyance as consequence)<br>• Batch (e.g. nightly batch, typically minutes, annoyance as consequence) | |
| PREPARE OFFLINE, RUN ONLINE | | Part of ADVENT. ANNOTATIONS are an example. |
| **Installation** | | |
| ANNOTATIONS | Instead of coding performance-critical aspects manually into each application in a family, leave these things out of the code, specify them in annotations declaratively, and use optimized frameworks or code generation to implement the "annotated" things. | |
| GLUECODELAYER | Instead of solving specific problems generically or by using dynamic reflective features, generate a specifically adapted layer of code that handles the issues more efficiently. | |

| Name | Problem - Solution | | Remarks |
|---|---|---|---|
| **Run time resource measuring** | | | |
| RESOURCE GUARD | When your system needs a reliable performance, you may need to apply SUICIDE, or SACRIFICE THE DEFENCELESS. How do you know what to do when?<br>Define BUDGETS for performance relevant resources, and create an instance that monitors your system for budget adherence. This RESOURCE GUARD knows the other tasks, their budgets, and the policy of reactions on a budget violation. It needs to be independent of the monitored tasks, so that their failure does not affect the RESOURCE GUARD by side effect. In critical systems, the RESOURCE GUARD may trigger a HARDWARE WATCHDOG that resets the system when the RESOURCE GUARD fails. | | Relates to SUICIDE, SACRIFICE THE DEFENCELESS, and BUDGETS |
| **Measure during development** | | | |
| PROFILING TOOLS | You have source code for the application. How can you get an estimate of the execution time of application modules? | Use a performance analysis tool (sometimes called PROFILING TOOLS). The profiling process helps the analyst determine which functions or code snippets take the longest time to execute and which functions are called most often. For example, the *prof* and *gprof* tools on the Solaris platform, the *DProf* for Perl, and several Java profiling tools. | |
| PLATFORM / MODULE MEASUREMENT | You do not have the source code of the application. How can you get an estimate of the execution time of an application? | Use a platform performance measurement tool that will report statistics about the utilization of the system resources (such as CPU, memory, etc.) by an application. As an example, use the Win2K performance-monitoring tool. | |
| ALT (ACCELERATED LIFETIME TEST) | | | |

| Name | Problem - Solution | | Remarks |
|---|---|---|---|
| **Probing** | | | |
| PEEPHOLES & TESTPOINTS | Your product is ready, but has some performance problems<br>You want to measure performance inside the sub-systems, and find where the bottlenecks are.<br>You want to verify interim processing is handled correctly in each of the sub-systems.<br>But adding code now to help you do so changes the behavior of the product, introduces bugs, and requires regression testing. | Plan in advance and place inside the product tools to measure and evaluate performance and ease troubleshooting.<br>Peepholes are junctions in your system where you can check interim results. They are the software equivalent of the small glass covered peepholes place in potential failure points in machinery, to make visible what might break.<br>Testpoints allow you to inject known input into various points in the processing flow, in order to observe if they are processed correctly further downstream, at another testpoint. In electronic circuit design, this is a standard design procedure. | |
| BUILT-IN PERFORMANCE SENSOR | During the development of the component, you want to ensure that you will be able measure its performance later at runtime. | Develop "performance-enabled" components, which provide capabilities to turn on and off performance collection mechanisms inside the component. | |
| PERFORMANCE SENSOR WRAPPER | You need to measure the performance of some components at runtime. Components are acquired off the shelf. | Add a wrapper around the component that provides a performance sensor behavior. Performance sensor wrappers will log several parameters including: invocation timestamps, end of execution timestamps, length of the request's queue, etc. | |
| LOG & PARSE | You would like to get performance data (execution times) from an actual runtime environment. How do you get runtime measurements? | During an application run, log events of interest such as time stamps for invocation and returns. Collect logs. Run a performance analysis engine to get execution time statistics from the log files after the run is finished. | |

| Name | Problem - Solution | | Remarks |
|---|---|---|---|
| WATCHDOGS & SENTINELS | Your system runs in an unattended mode. Some of the components are unreliable and could hang/crash without returning the execution control token to the rest of the application. How do you ensure the application detects failures and acts accordingly? | Add a watchdog for each unreliable component whose execution could halt the system operation. | |
| PLEASE TRY AGAIN (REMOVE AND RESTORE) | One or more processing error or exception has occurred.<br>It seems that in spite all integrity checks and fail-safe mechanisms designed and tested, the system is now in an unpredictable failure mode.<br>How do you recover from such a severe failure? | You rely on reverting to a known state, and assume that this brings back the system (or sub-system) to a known initial state.<br>Examples – you do this each time you make the three-finger-salute – Ctrl-Alt-Del, don't you? | |