Malkawi *Human-centric Computing and Information Sciences* 2013, **3**:22
http://www.hcis-journal.com/content/3/1/22

Human-centric Computing and Information Sciences
a SpringerOpen Journal

RESEARCH Open Access

The art of software systems development: Reliability, Availability, Maintainability, Performance (RAMP)

Mohammad Isam Malkawi

Correspondence:
mmalkawi@aimws.com
Jordan University of Science and Technology, Irbid 21410, Jordan

Abstract

The production of software systems with specific demand on reliability, availability, maintenance, and performance (RAMP) is one of the greatest challenges facing software engineers at all levels of the development cycle. Most requirements specification tools are more suited for functional requirements than for non-functional RAMP requirements. RAMP requirements are left unspecified, specified at a later stage, or at best vaguely specified, which makes requirements specifications more of an art than a science. Furthermore, the cost of testing for RAMP requirements is quite often prohibitive. In many cases, it is difficult to test for some of the RAMP specifications such as maintainability, reliability, and high availability. Even the test for performance is quite often workload dependent and as such the performance numbers provided at test time or at system commissioning time may not be achievable during actual system workload. What makes the subject matter more difficult is the absence of a clear set of rules or practices, which, if followed closely, produce a system with acceptable RAMP specifications. As such, and until the design of RAMP software systems becomes a well understood theme, the development of such systems will be a fine art, where the tools and capabilities of developing such systems will depend on the particular system to be developed, the environment in which it will run, and the level of expertise and knowledge deployed. Just like no two pieces of art produced by the same artist are the same, no two software systems will have the same RAMP characteristics.
This paper will focus on the paradigms involved in the production of RAMP software systems through several case studies. The purpose is to promote the interest of researchers to develop more specific guidelines for the production of SW systems with well defined RAMP qualities.

Introduction

The production of software systems with specific demand on reliability, availability, maintenance, and performance (RAMP) is one of the greatest challenges facing software engineers at all levels of the development cycle. Most requirements specification tools, e.g., Accent, Nu Thena, SES, Rational, are more suited for functional requirements than for non-functional RAMP requirements [1-5]. RAMP requirements are left unspecified, specified at a later stage, or at best vaguely specified, which makes requirements specifications more of an art than a science [6-8]. Furthermore, the cost of testing for RAMP requirements is quite often prohibitive. In many cases, it is difficult to test for some of the RAMP specifications such as maintainability, reliability and high

availability. Even the test for performance is quite often workload dependent and as such the performance numbers provided at test time or at system commissioning time may not be achievable during actual system workload. What makes the subject matter more difficult is the absence of a clear set of rules or practices which, if followed closely, produce a system with acceptable RAMP specifications. As such, and until the design of RAMP software systems becomes a well understood theme, the development of such systems will be a fine art, where the tools and capabilities of developing such systems will depend on the particular system to be developed, the environment in which it will run, and the level of expertise and knowledge deployed [8]. Just like no two pieces of art produced by the same artist are the same, no two software systems will have the same RAMP characteristics. Software system design and development is by and large more complex than the programming phase of it, which was perceived as an art by Donald Knuth in his classic book "The Art of Computer Programming" [9].

There has been quite a bit of argument in the literature on what constitutes an art or a science in the software production cycle. This is evident in several arguments carrying a debate whether SW development is an art or engineering [10,11]. In a post on the internet titled "Software Development: Art or Science", Sammy Larbi from the University of Houston writes: "There is a seemingly never-ending debate (or perhaps unconnected conversation and misunderstandings) on whether or not the software profession is science or art, or specifically whether "doing software" is in fact an engineering discipline [12,13]".

In an article titled "The Art, Science, and Engineering of Software Development" [14], Steve McConnell argues that SW development is art, science, craft, and many other things.

Naveen Gunti [15] examines the benefits of using function point analysis in the context of the art of SW engineering. Robert Glass in his book "Software Conflict 2: The Art and Science of SW Development" agrees with earlier findings that SW design is a model that emerges in the human mind [16] similar to how a piece of art emerges in the mind of an artist. Jim Waldo, a distinguished engineer at SUN Microsystems [17] writes "Software engineering is a lot less like other kinds of engineering than most of us would like to think. There is an aspect of art to what we do, that is learned not in school but by finding a master and serving an apprenticeship".

The purpose of this paper is not to argue whether SW development is an art or science, or what is a science or an art in the cycle of SW development. Rather, this paper will focus on the paradigms involved in the production of RAMP software systems through several case studies. The purpose is to promote the interest of researchers to develop more specific guidelines for the production of SW systems with well defined RAMP qualities. The performance paradigm section, we will discuss the performance paradigm as one of the main pieces of software design. The reliability paradigm section will present the reliability and availability paradigms. Maintainability issues will be discussed in section Discussion. Section Conclusion will address the necessity for the development of guidelines and best practices for RAMP software system development, where a general framework for RAM is proposed. Concluding remarks are discussed in Discussion section.

The performance paradigm

My most recent encounter with a performance paradigm was related to the modeling of water flow in cases of flood, tsunami, cyclones, river floods and similar cases. Simulating 72 hours of water flow in some cases requires more than 12 hours of execution on a mid

size server. Is there a need to enhance the performance of the software? What should the performance requirements be? And how to achieve the required performance in a cost effective manner? Another case, involved a software tool used to mitigate intermodulation interference problems in telecommunication systems. In some cases, the SW tool would run for several hours, overflow the disk space with data, and cause the system to crash. Should the performance of the tool be improved? What are the performance requirements, how to achieve them in a cost effective manner? A more interesting question would target the limits of performance, which can be achieved on the system! Yet, one more example, involving the modeling and simulation of the evacuation of large facilities in case of natural or human inflected disasters. The simulation time can run for several hours for a given scenario. How much improvement is required, if any?

When performance requirements are analyzed independently from the concept of productivity [18], the above scenarios may not warrant a performance improvement. In the above examples, it is often required to repeat the study with different parameters and scenarios. An optimal solution in most cases is a must due to the safety and security nature of the studies. Repeating the experiments for several hours each time can be a frustrating practice for the engineer, which may lead to the introduction of approximation techniques or compromising optimality. Besides, the longer the process runs on the system, the more likely it will experience faults and errors during the execution process [19-21]. In the case of the interference analysis SW tool, one engineer explains: "If I can run the SW in less than one hour, I can generate more studies per day. I can make more money. Each study costs $2000.00".

Once we get to the point where a performance enhancement is needed, then we have to answer the following questions: "How much improvement are we looking for"? This (requirement) question has to be answered by the end user of the SW system. What is the limit of performance improvement? This (specification) question has to be answered by SW engineers, who should take into consideration the economics of the hardware settings. But the most difficult (architecture) question is: "How do you achieve the performance gains". Figure 1 shows a general framework for performance improvement. Details of the framework will be further discussed through the following study cases.

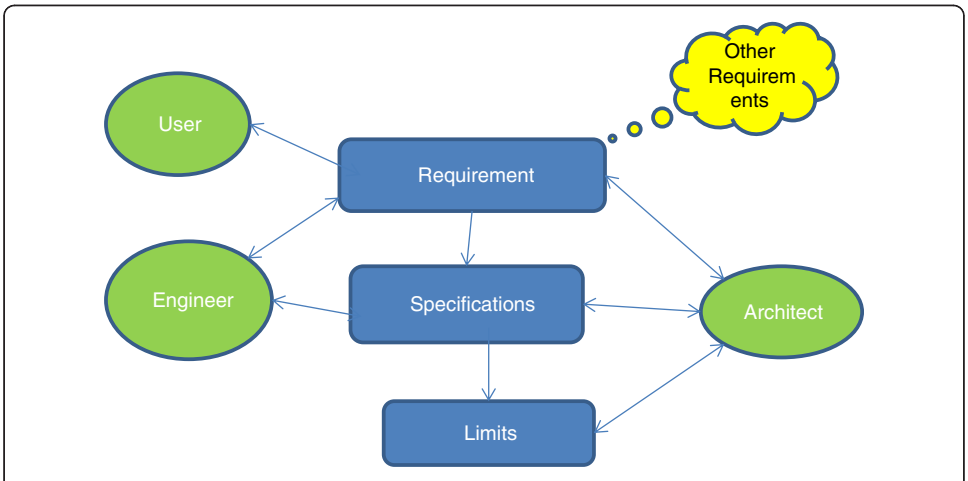

Figure 1 Performance improvement framework.

Case 1: initialization problem

Failure recovery is one of the important factors related to availability. Failure recovery involves the recovery mode, the time to recover and the recovery success rate. One of the recovery modes, after a complete system failure, is the restart of the failed system and/or the applications on the same system. In order to achieve a certain level of availability (5 NINES for example) [22-26], the system must be restarted (reboot, initialize, restore checkpoints) within a certain time constraint. The minimum acceptable recovery time is determined using technique such as Markov process analysis [27] or stochastic activity network simulation [28]. This is a case where the **performance requirement** is determined based on another higher level requirement, e.g., availability requirement.

In the course of analysis, the recovery time is further decomposed into subtasks based on the time consumed by each subtask. Performance budgeting is then used to estimate the potential enhancement of each subtask, if possible at all. Performance budget, in this context, defines the limits of performance improvement. Some of the subtasks that consume most of the budgeted time, in our example, include the boot image loading time, the kernel initialization time, and the payload components initialization time. It is essential to determine if any of the sub-tasks can be skipped in order to save time and speed up the process of initialization. For example, memory tests can be skipped at initialization time only to be performed later, when the system is not too busy. Also, the initialization of some payload components may be deferred until the system is completely recovered.

In our example, we draw the attention on the loading time of the boot image of the device; assuming that the boot image does not reside on the device, as is the case in wireless infrastructure devices. When evaluating the boot image load time using ftp protocols, it was noticed that the speed of ftp load depends on the file size as well as on the number of concurrent ftp load sessions. We evaluated two versions of the ftp protocol. We measured the load time for different file sizes and different parallel loads. Figure 2 shows that the load time is minimal when using version 2 of the ftp load protocol with 11.6 MB file size. The best results can be achieved when 4 download sessions are performed in parallel. We observed that increasing the number of open ftp sessions beyond 4 will increase the overall loading time. This is an example, where performance improvement requires careful selection of protocols and parameter setting.

The initialization example reveals several important points.

1. The dependence of performance requirement on other requirements (see Figure 1) such as availability and reliability. In this case, the recovery time (a performance parameter) depends on the recovery time, in the availability model.

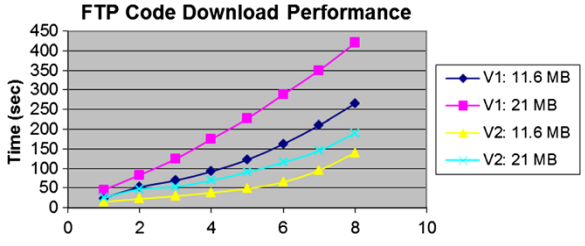

Figure 2 FTP code load performance.

2. The impact of parameter configuration, environment settings, and tuning on performance. The setting and configuration of parameters require expertise in experimentation, data collection, and analysis. The complexity of parameter setting and tuning can be very large; therefore, selection of the proper parameters and setting the proper values remain an act of art unless extensive experiments are used to validate the selected parameters and their values.
3. Setting a parameter may produce counter intuitive results. For example, increasing the number of parallel downloads results in longer download times rather than shorter one. Another example is priority inversion of threads, where a starvation or deadlock may occur when priority inversion is actually used to prevent starvation or deadlock.

Case 2: the interference sort and search problem

This case deals with the problem of intermodulation interference in telecommunications network systems. Intermodulation interference occurs due to the mixture of radio frequency (RF) signals in nonlinear devices. The resulting new RF combinations can be very large. For a total of (n) RF signals, the K^{th} order intermodulation produces a total of (n^k) combinations. Only a subset of the (n^k) combinations may interfere with other signals at receiving devices. In order to resolve or prevent interference, it is essential to search for those combinations which may cause interference. The complexity of the search grows exponentially with the growth of the number of original signals [29].

This is an example with several facets of art and design. Radio frequency engineers require a SW that can solve the interference problem in relatively short time. The SW should be robust enough not to overload the memory or the disk space. The SW should also be able to adapt easily to new sources of data with different data formats as well as to different technologies (e.g., CDMA, GSM, WiFi, WiMax, LTE).

Note that none of the requirements is defined in a clear quantitative manner. For example, RF engineers will not come clearly and say that we need a system that performs 2^{nd} or 3^{rd} order intermodulation for (N) signals in less than (X) seconds. Ironically, the telecomm industry until today states the requirement for completing an intermodulation interference analysis for a given site in days (typically 48 hours) rather than minutes or seconds. This is due to the complexity of the analysis which includes sorting and searching billions of elements in large files. It is also due to the lack of well defined strategy for defining the performance requirement of such system. Industry, however, acknowledges the need to address interference analysis in a timelier manner [30-32]. To illustrate the complexity of the issue, consider the following example.

Consider 2^{nd} order intermodulation with 5 signals (S_1, S_2, S_3, S_4, S_5). One potential list of intermodulation combinations would be the sum of any two signals (S_1+ S_2, S_1+ S_3, S_1+ S_4, S_1+ S_5, S_2+ S_3, S_2+ S_4, S_2+ S_5, $S_3 + S_4$, S_3+ S_5, S_4+ S_5). It is required to find all signals ($S_i + S_j$) that are larger than a given signal (S_g). When the number of combinations ($S_i + S_j$) is relatively small, we can sort this list (using a quick sort algorithm for example) and use a fast search algorithm, e.g., binary search, and locate the desired signals. This is a typical SW engineering practice well known by any engineer in the field. However, when the number of original signals is relatively large, say 10,000 and the intermodulation order is 3 instead of 2, then we have to deal with 10^{12} permutations. This is too large of a number to deal with, which could easily cause the paging

algorithm to thrash, and the time to sort and search to be in the order of hours rather than minutes.

This is a good example to illustrate how the art of SW engineering can help reduce the time complexity of the algorithms. Instead of looking at the combinations above as a linear array, let's view them as a lower half of a matrix as in Table 1 below. Only the original signals (Si) need to be sorted (Sort N signals instead of sorting N^2 signals).

Note that, if $(S2 + S3) > S_g$ in the third column, then all elements following (S2 + S3) in this column and in the following columns will also be larger than S_g. Similarly, if $(S1 + S4) > S_g$, then all the elements in the second column following S1 + S4 will be larger than S_g. Note that this representation of the data provides a sorted view of each of the columns. Search within each column can be as fast as a binary search. The solution provided for this problem is more of an art than simply SW engineering. In this case, it is the presentation layout of the data that leads to a productive solution. The complexity of algorithms developed using this representation is 2 orders of magnitude better than those with classical view of the data [30].

The intermodulation complexity problem reveals that the limit of performance improvement depends on the method of data presentation. This is especially true for big data processing. For each problem, the optimal data presentation must be identified by the system architect, who can then determine the limits of achievable performance.

Case 3: data mining and management

This case deals with large data manipulation and maintenance. Consider the following problem in telecomm data management systems. In a typical service provider network several switches control a certain segments of the network. Each switch controls several cell sites. One of the functions of a switch is to collect performance and radio frequency service measurement data related to the network segment (RFSM) as well as per call management data (PCMD) [33]. RFSM and PCMD Data are released by the switch at a certain time, e.g., on top of the hour. The amount of data released per hour can be very large depending on the number of cells in the territory of the switch and activity of the network. The data released per unit time can be in the order of gigabytes per hour. Moreover, the data released by the switch changes in format, type, and structure every time a new switch version is released. This change calls for a change in the code responsible for parsing and loading the data as well as in the schema of the database which hosts the RFSM and PCMD data. The switch experiences major release changes several times a year; in some cases it can be 4 releases per year. During the transition from one release to another and the subsequent code modification, the data management system could become unavailable for an unidentified time (depending on the success and duration of system upgrade). The cost of maintenance is non-

Table 1 Data representation for intermodulation problem

Signal	S1 + Sj (J = 2..5)	S2 + Sj (J = 3..5)	S3 + Sj (J = 4..5)	S4 + Sj (J = 5..5)
S1				
S2	S1 + S2			
S3	S1 + S3	S2 + S3		
S4	S1 + S4	S2 + S4	S3 + S4	
S5	S1 + S5	S2 + S5	S3 + S5	S4 + S5

trivial in terms of money and the period in which the system remains either unstable or unavailable.

Another issue related to performance is the time required to perform queries or to generate reports. Obviously, query and report generation performance depends on the size of the DB, the schema structure, and the organization/distribution of data within and across the tables. The well known rules of DB performance tuning can achieve limited performance improvement. The data is too large and diverse, which makes the process of data analysis difficult and time consuming. Finally, the reliability of the host servers has a direct impact on the reliability of the system at large. Hosting all the data in one DB and on one server is the least reliable and has the highest performance hit. Hosting each switch on its own separate DB and then on its own server is the most reliable, with best performance and highest cost.

What matters to the user at the end is how soon would data be available for him to query and browse, and how fast he can generate reports when they are requested by a manager?

In a complex system like the one described above, there is no single set of rules that can be specified, and if followed closely, the required performance will be achieved. And that is where the art of system architecture plays a great role. The proposed architecture of the system is shown in Figure 3. The figure appears as a piece of art whose components are squares, rectangles, circles, and arrows. Each of these components contributes to the overall system performance, reliability and availability.

The art begins with the selection of data transfer mechanism and application, e.g., ssh (secure shell file transfer), sshfs (secure shell to share files [34], ftp, ODBC (open database connectivity) or others. In the selection process, one needs to consider the tradeoffs of security, performance and reliability.

Another key performance parameter is the speed at which we can parse the raw data, prepare if for the loader, and load the data into DB tables. If this process fails to complete in less than (x) minutes, then the query and report performance will lag behind. Parsing and loading design requires the selection among languages, loading mechanisms, file structures, and inter-communication with the DB servers. For

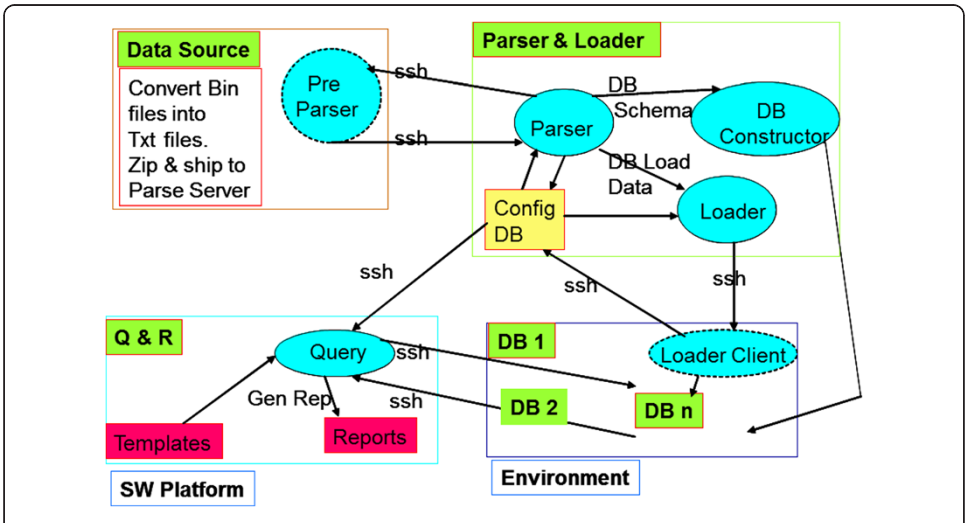

Figure 3 RFSM and PCMD system architecture and configuration.

example, Perl scripting language allows parsing very complicated text structures more efficiently than other languages such as C# or VB. However, loading the parsed and structured files into the DB is faster and more efficient using C#.

At the database level, we have to worry about how fast we load data into the DB and how fast we pull data out. The raw data as it comes from the source lends itself to large tables with millions of records and thousands of fields. Key to high performance is the ability to partition the tables in a meaningful manner. The irony is that there is no clear definition for what would be a meaningful manner. What we can say in general, is that larger number of tables allows for more parallel loads of tables into the DB. Also, smaller table sizes mean faster query and report generation. Too many tables, on the other hand, may slow down the queries and reports if they happen to scan large number of tables. A DB designer would like to store homogenous data (data commonly requested in a given report or query) in the same table; this design improves the data locality structure and leads to faster access of data, although it puts more burden on the parser and loader. In a relatively small DB system, this is easily done by investigating the semantics of the data. In systems like the one described in this example, this task is next to impossible.

Here comes the art part of the DB design. We can watch and monitor the access patterns of the DB, and overtime we learn which data fields are commonly retrieved in a given query or report. Based on access pattern recognition, we build intelligence into the mechanism responsible for partitioning the data. Table partitioning, and data migration between tables can be implemented in a mechanism, we call, the DB Constructor. Using this mechanism, the database is no longer a static repository structure. The DB schema changes over time, and the data migrates between tables. The more we access the DB, the better it performs. This phenomenon is exactly opposite to the well known SW aging phenomenon, where SW performance and reliability falls down with age.

The RFSM and PCMD case reveals the following important factors related to performance

1. Data partitioning has significant impact on performance. In large systems, it is very difficult to find an optimal partition. Adaptive learning algorithms can be used to find an optimal data partition.
2. Tools and languages must be carefully selected, observing that each subtask may require tools different from those used for other tasks.
3. Intercommunication between various modules is a major performance bottleneck. Proper intercommunications solutions allow for better performance optimization.

The reliability paradigm

SW reliability continues to be a major reliability bottleneck in large and complex systems. Compared to HW reliability, SW systems are more difficult to design for reliability, more difficult to test, and could constitute a safety and economic hazardous. A 2002 study commissioned by the National Institute of Standards and Technology found software bugs cost the U.S. economy about $59.5 billion annually [35]. More recently, a study conducted by Cambridge University researchers estimates the total annual cost at $312 worldwide [36]. SW failures have contributed to major system failures in the past

few years including the August 14, 2003 Blackout of Northeast Power Grid [37], Mars Climate Orbiter (September 23rd, 1999) [38], USS Yorktown (1998), Ariane 5 rocket explosion (1996) and many others [39]. Although, the SW industry has become more experienced in measuring and testing for reliability, we are still far behind in issues related to the design of highly reliable SW systems.

Problems like memory leak, memory corruption, memory overflow, deadlock and others have been known for quite a long time. Today, most SW systems still contain scores of bugs related to these problems. In a study, recently conducted on a large and complex SW system, where data was tracked for several years, it was observed that the system overall reliability does not improve over time. Figure 4 shows 3 releases of a system observed over 10 years time period. Note that the time it takes to stabilize the system (stability period) increases in the second and third releases. Also, the rate of defects per month goes up. It is true that the system also grows in size and complexity, but the expectation is that with time, the knowledge and expertise will also grow. This phenomenon is generally observed in many complex SW systems.

One of the main challenges in SW reliability is stress and accelerated life testing. For hardware, this is a well known procedure. For SW, there is no unified method on what would constitute a stress or accelerated test [40]. The choice of tools, methods, coverage, confidence levels, and ways of interpreting results remain an art for most of SW engineers.

In many cases, reliability depends on performance. For example, a fault not detected within (x seconds) may propagate and cause the system to be unstable. This was the case in the Arian 5 explosion (1995) [39], where the SW system failed to convert a 64-bit floating number into a 16-bit integer. The fault was detected, but only after the error has propagated and a command to shut down the system was issued; the rocket exploded shortly after.

Measuring SW reliability is by and large harder than that of hardware reliability. When a piece of HW is delivered out of the factory, the expected lifetime of that piece is known with a good degree of certainty. Stress testing and accelerated life testing methods have been used successfully for a long time. When it comes to SW reliability, the problem is much harder. The complexity of the problem stems sequence from the fact that the input data to a given SW is time and conditions variant. Different methods have been used to measure SW reliability. Among these methods are code coverage tools, number of code lines, and number of defects/bugs found in a given number of lines [41].

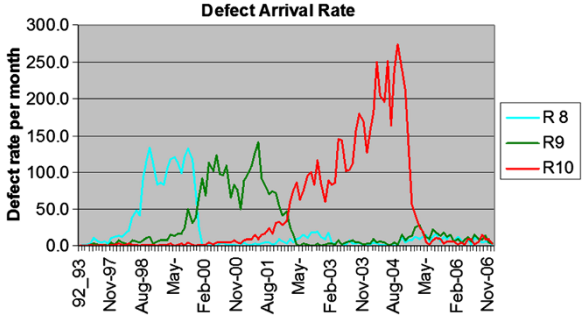

Figure 4 System reliability of three major releases measured over 10 years period.

Commenting on the number of code lines as a measure of reliability, Bill Gates says that "measuring programming progress by lines of code is like measuring aircraft building progress by weight". The number of defects/bugs found in a SW system during testing is a measure of how unreliable the system was before debugging and testing. The remaining bugs/defects in the system can turn to be a major cause of outage or safety hazard (e.g. the failure of patriot missile to track down an incoming scud missile due to arithmetic rounding errors [42]. Code coverage tools produce as good of a reliability measure as the covered code in a given test.

The artistic part of reliability measurement lies in the design of the various tests and procedures to stress the system and find out all possible errors, bugs and defects. For example, when a function erroneously deletes a pointer, resulting in a memory leak, and the function is never called at testing time, the leak will persist in the SW system. The lack of robust testing likely contributed to the radio system outage over the skies of parts of California, Nevada and Arizona. The system failed to failover to a backup server during a data overload failure scenario [43,44]. Testing did not cover this scenario.

Consider the following example which shows the limitation of stress accelerated testing. A truncation error in a 24-bit fixed point arithmetic can be as small as 0.000000000000000000000001 (decimal 0.00000009). If we were to represent 1/10 of a second using this arithmetic, then the truncation error can grow to 0.3 seconds after 100 hours of operation. Obviously, the error will not be detected if the accelerated test was run for less than 100 hours. Depending on what application the SW will be used in, the error can be either detected (after 100 hours of runtime) or remains hidden (if the application terminates in less than 100 hours).

The main problem which reliability engineers have to resolve is the design of the proper tests which reveal the majority of unreliable parts of the system they are designing. There is no set of well defined rules to follow. The variable space, which includes the input space, the failure modes, and applications, can be infinitely large.

In order to reduce the subjectivity of SW reliability testing and measurement, it is highly recommended to make use of well kept dictionaries and databases of failure modes and scenarios. One method, which can be used in this regard is the failure scenario analysis (FSA) [45]. FSA is recommended as a guide for continued reliability improvement. Failure modes are described both qualitatively and quantitatively. For example, for a given failure mode, there should be a description of the methods used to detect, isolate, contain and recover the failure. Also, there should be a specification of the time required to detect, isolate, contain and recover the failure. The times have to be carefully specified for each different application. The dictionary of the failure modes should be maintained during development, testing, and deployment.

Measuring the availability of a SW system is yet another challenge and piece of art. The availability of a system depends mostly on how fast it can be recovered after experiencing a failure mode as well as how frequently it fails. Most of the availability numbers used by various vendors are based on real-time measurement of availability of systems in the field. However, it is very difficult to provide an accurate measure of availability of a system when it is ready to be deployed. Different modeling and analysis techniques exist for availability measurement including analytical methods and simulation methods [27,28]. It is not uncommon to hear the phrase "there is no good

availability model; but there is a valid one". In my experience as a SW reliability engineer, I have not seen anything more art oriented than availability modeling.

As an example, consider the models in Figure 5. Figure 5a shows a model where the system failure is represented by one failure rate which will be the sum of the failure rate of all components. Figure 5b shows a more detailed failure and recovery behavior for each failure mode. The system fails at different rates, has different recovery mechanism for each failure mode and different recovery success rate (ρ). Figure 5b can also represent the case, where failure modes can be categorized into categories and each category represents a group of failures with similar failure and recovery behavior. All three models are correct representation of the system. Which is a better model, though, depends on how much details are available about the system and how close the system availability needs to be monitored.

Art of budgeting for availability

Availability is quite often measured in downtime minutes, outage duration and frequency. A FIVE nine availability system allows for 5.24 minutes downtime per year. The distribution of these 5.24 minutes among various system components is not always a straightforward matter. Quite often the distribution of downtime minutes needs to be negotiated among the owners of system components. More interesting even is how the system gets partitioned into components or subsystems. The broadest partitioning is the typical hardware/software partitions. Such partition makes it very difficult to design for the proper recovery when a failure occurs. More detailed partitioning, however, makes it very difficult to achieve the required availability or outage requirement. An

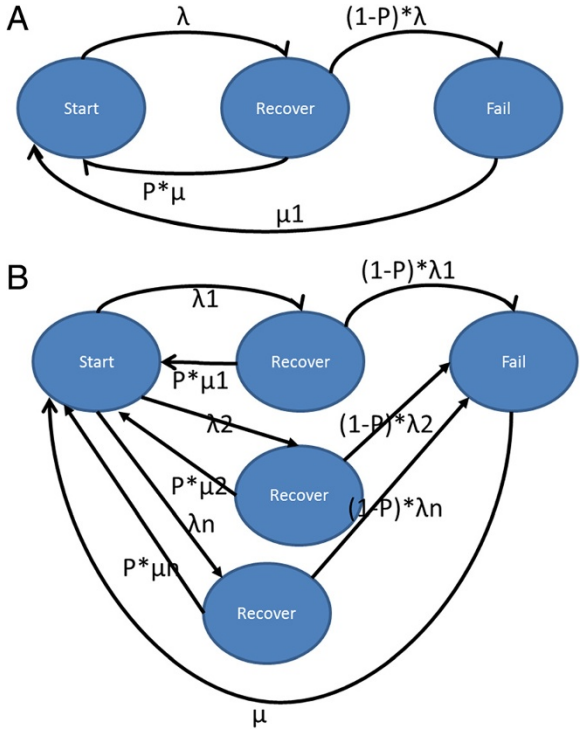

Figure 5 Availability models (A) condensed (B) detailed.

optimal distribution of outage time among the various hardware and software components is a fine art orchestrated by expert system architects.

Take for example, the system depicted in Figure 3. One way of distributing outage time is to evenly distribute it among the four subsystems. This is easy and straightforward. However, this distribution places a great burden on the DB environment where the number of transactions executed per unit time is very high. In this system, we have built N load sharing DB servers with auto failover, where a DB server can failover to the least loaded server. This implementation allowed more downtime to be allocated to the secure shell (ssh) communication, which turns out to be the least reliable in the system.

Art of design for reliability and availability

The greatest question of all remains "how to design and implement a reliable and highly available software system?" Is there a way to develop a system without memory leak, without memory corruption, without address space violation, without buffer overflow, without timing and synchronization errors, without data format translation errors, and the list goes on and on? Can we design a system where an error can be detected before it generates a serious failure and possibly a catastrophe like the explosion of a rocket or air flight control mishaps? Can we eliminate interface errors when two or SW modules are linked to form a more complex system? How much education, training, code inspection, debugging, and testing are needed before a SW can be certified for reliability and availability? The best answer to any of these questions is "we will try our best".

The most reliable SW development continues to be an art which involves several instruments. Such instruments include the selection of personnel skills (both development and management), the selection of development tools (including language, development environment), the selection of code coverage tools and code coverage strategies, the selection of code inspection tools and methods, the setting of the testing environment (including test suites, benchmarks, testing time), as well as the careful selection of third party SW components. The combination of selected instruments at a given SW production house dictates the level of SW reliability and availability.

In essence, reliability and availability is not a single task or product. Rather, it is a set of availability work products [23] as shown in Figure 6. Once implemented, the AWP can deliver a robust, reliable, and highly available system.

Discussion

The study cases discussed in this paper show how difficult it is to satisfy various RAMP requirements. Each and every software system has its own characteristics which are different from others. The process of achieving RAMP requirements remains an art that engineers and architects need to possess. The expertise of several engineers and architects may have to be integrated. In order for this art to be more effective and controllable, a peformability framework, which combines the four non-functional requirements (performance, reliability, availability, and maintainability), is proposed and shown in Table 2.

The reliability requirement, for example, has specific performance requirements such as failure rate (λ), fault detection time, fault isolation time, and fault containment time. These parameters must be defined in order to achieve a certain level of reliability. Similarly, availability can only be achieved if availability parameters meet

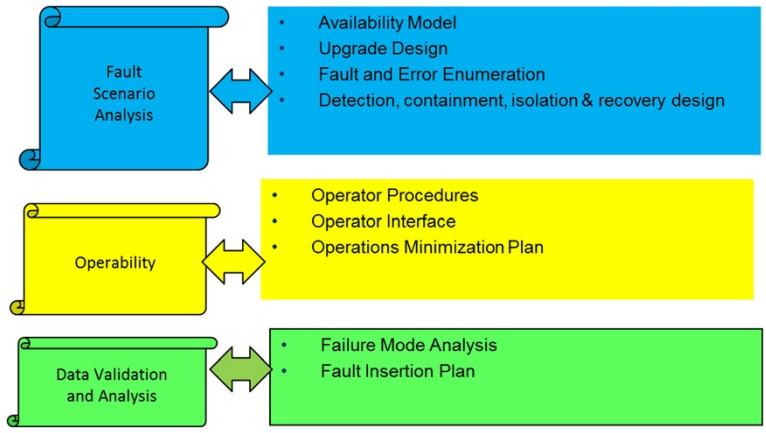

Figure 6 Availability Work Products (AWP).

certain performance requirements, e.g., the time to recover from a failure and the rate of successful recovery from failures (ρ). Also, specific performance requirements (throughput, speed/speedup, and bandwidth) need to be defined in terms of reliability, availability, and maintenance parameters. For example, the throughput of a wireless system can be obtained only under specific packet loss rate. The system speedup, when multiple units are used, can only be defined in terms of the redundancy mode; for example, M out of N load sharing mode sets the maximum speedup to M, although the system has $N > M$ units.

The performability framework also defines the interaction between various stakeholders of the system under consideration. The example of InterMod60 is a clear example where the restructuring of the intermodulation algorithms required the knowledge of RF and software engineers.

Another example is the use of asynchronous versus synchronous processes. In the example used in section case 3: data mining and management (Figure 3), the use of asynchronous processes was the proper solution for achieving the required availability. The use of time-synchronized processes would have been the preferred choice for achieving higher throughput. The skills and expertise of the architects had to be carefully deployed to decide which of the techniques is more useful. Same applies to the selection of the language and the run-time environment. In the example used in this study (Figure 3), it turns

Table 2 Performability framework

	Reliability	Availability	Maintenance
Performance	• Failure rate (λ)	• Recovery time	• Maintenance rate (μ)
	• Fault detection, isolation, containment time	• Recovery success rate (ρ)	• Maintenance success rate
		• Downtime	
Throughput	• Failure rate (λ)	• Downtime	• Maintenance rate (μ)
		• # NINES	• Maintenance method
Speed/speedup	• Redundancy mode	• Recovery mode	• Maintenance mode
Bandwidth	• Failure rate,	• Recovery success rate (ρ)	
	• Redundancy mode		
	• Detection		

out that the Linux environment is more suitable for the parser and loader processes. However, the report generation systems perform better in a MS windows environment. Thus it is worth solving the communication links between multiple environments than settling for a single environment where performance is compromised. Of course, the security and reliability of links between multiple environments need to be addressed. This was an example, where the diversity of tools was the solution for achieving both performance and reliability requirements.

The study also shows that proper performance metrics must be used. Where throughput is the main performance index for one system, response time can be the index for another system. The system engineers and architects must specify without ambiguity the main performance indexes to be optimized. In the initialization problem, discussed in Case 1: initialization problem section, the loading time of multiple boot images was the performance index. However, this index depends on another one, which comes from a higher level system, the availability of the network; in this example, the recovery time (an availability parameter) defined the boot image load time (a performance parameter). Hence, the relation between performability indexes of the various components of the system must be observed.

It should be noted that performance requirements may require modeling and simulation in order to set the proper performance values. Modeling and simulation are generally used to define the limits of performance, for example, the maximum throughout achievable under certain conditions. The limits of achievable performance should be well defined. This allows more realistic performance requirement setting. This in turns requires the selection of workloads and benchmarks. In the initialization problem discussed in section Case 1: initialization problem, the selection of the workload for testing the performance of ftp had to be carefully selected. As another example, the call model used in the evaluation of networking and telecomm systems has a direct impact on the performance requirements and measurements.

Testability is of equal importance. Any performability requirement must be measurable both in the lab (during development) and in the field. Requiring a transaction to be completed in the order of nanoseconds for example, where the lowest granularity of measurement devices is in microseconds is counterproductive.

The selection of the data representation model is of utmost significance as we illustrated in the example given in section case 2: the interference sort and search problem. Trends within large data sets might be better revealed when using one data representation model versus another. Therefore, the system engineers and architects must give enough consideration to the selection of the data representation model. In our example, the performance improvement would not have been possible without the proper selection of the data representation model.

The example of large data mining (discussed in section case 3: data mining and management) shows the challenge of organizing data in various tables. Database update and queries heavily depend on the data distribution among tables. The optimal distribution of data may not be easily attainable due to the complexity of the system measured by large number of tables and large number of attributes; (in our example, the number of tables exceeded 2000 tables, and the number of attributes in some tables exceeded 200). In this case, it is essential to develop adaptive algorithms to shuffle data across tables throughout the lifetime of the system, thus creating a dynamic database schema. In the case presented in section case 3: data mining and management section, performance was dramatically improved after deploying dynamic data migration among the tables of the database.

The selection of tools and languages to be used in the course of the SW system development is shown to directly affect the overall performance of the system. The selection of the tools and languages should be performance oriented rather than dictated by the available skills of the development team.

The performance requirements should not be set in isolation by only one part of the system development, whether the architect, the user, or the system engineers. Rather, this process must be integrated by all parts. The end user, the architect, the test and the system engineers all need to participate in defining the performance requirements of the system. The requirements of the complex system presented in section case 3: data mining and management section were established by all parties involved: the end user, the architects, the software engineers, and the testing engineers.

Reliability and availability requirements face more challenges than performance requirements. For example failure rates are very difficult to quantify for software systems unless accelerated software testing is performed. SW accelerated life testing has not matured enough in the SW industry and remains an open area of research and development.

Also, testing for reliability and availability is a challenge. Reliability and availability models are as good as the parameters used to drive the models, such as failure rates, recovery success rate, and recovery and maintenance time. The state space of SW systems can be very large such that the use of analytical availability models becomes prohibitive. Consequently, simulation models with significant approximations become the only means of measuring availability and reliability.

The use of failure scenario dictionaries and failure mode and effect analysis can be very useful in improving system reliability and availability. Keeping a history of defects, their means of detection, containment, isolation, and recovery will certainly help in mitigating future defects of the same type. We recommend the use of availability work products shown in Figure 6.

Proper budgeting for reliability and availability is essential for building reliable systems. For example, when the total down time of a system is set to a certain number (60 seconds per year for example), it is absolutely necessary to distribute the 60 seconds among the subcomponents of the system. The proper distribution of the budget is key to being able to achieve the requirement. The example given in section Case 1: initialization problem (the initialization problem) was based on availability budget allocation.

In summary, the process of building software systems with well-defined RAMP requirements is an art, where the engineers must choose and select among a very large number of parameters such as tools, languages, models, architectures, design methods, benchmarks and workloads, testing environment, performability indexes and more.

However, this art is not an open ended one. Rather, it is confined to methodologies and practices. The availability work products constitute a methodology by which the art engineer can use to build a robust high availability system. Benchmarking, work load characterization, performance metrics definition, and evaluation constitute a methodology by which the art engineers can build systems with well-defined performance requirements.

Conclusions

This paper presented the challenges of building systems with certain non-functional performability requirements (performance, reliability, availability, and maintainability).

Several case studies are presented to illustrate the performability paradigms, particularly the performance, reliability and availability, maintainability, and budgeting. The paper presents a framework, which shows that the art of software engineering for non-functional requirements can be engineered in a rather systematic manner. The author has used the presented framework to work out several cases, to achieve significant improvements in performance, reliability, availability and maintainability. The performance of intermodulation interference system (Intermode60) achieved an order of magnitude speed improvement. The availability of the data mining system achieved more than four NINES availability through the utilization of diverse languages and environment, and adaptive algorithms for dynamic maintenance of the massive database system. Finally, the paper shows that achieving the required performability parameters is human centric and depends on the integration of diverse skills of engineers, and the sense of art embodied by those engineers.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
MM investigated the impact of non-functional requirements such as reliability, availability, maintainability and performance (RAMP) on the overall system architecture. The author provided a framework to facilitate the implementation of RAMP requirements in a rather deterministic manner rather than a mere art of software development as is the current state of the art. MM read and approved the final manuscript.

Received: 16 May 2013 Accepted: 2 December 2013
Published: 21 December 2013

References

1. Eushiuan T (1999) Requirements and Specifications. Carnegie Mellon University, 18-849b Dependable Embedded Systems
2. Ascent® Logic website. http://www.alc.com
3. Nu Thena® Systems website. http://www.lynuxworks.com/
4. Rational® Software website. http://www-01.ibm.com/software/rational/
5. Scientific and Engineering Software® website. https://www.ece.cmu.edu/~koopman/des_s99/requirements_specs/
6. Eushiuan T (1999) Requirements & Specifications. Dependable Embedded Systems. Spring. http://www.ece.cmu.edu/~koopman/des_s99/requirements_specs/
7. Lattemann F, Lehmann E (1997) A methodological Approach to the Requirement Specification of Embedded Systems. In: Proceedings of the First IEEE International Conference on Formal Engineering Methods, ISBN 0-8186-8002-4; Nov. 12–14, 1997. Hiroshima, Japan, pp 183–191
8. Patridge D (1995) Where do Specifications Come From? In: Achievement and Assurance of Safety - Proceedings of the Third Safety-Critical Systems Symposium. , Brighton, United Kingdom, pp 302–310
9. Donald Ervin K (1974) The Art of Computer Programming, Volume I: Fundamental Algorithms, 3rd edition. Addison-Wesley International. ISBN 0201896834
10. John C (2012) Art and Science of Software Engineering. University of Washington Blogs. http://blogs.uw.edu/ajko/2012/08/22/john-carmack-discusses-the-art-and-science-of-software-engineering/
11. Ko AJ, et al. The State of the Art in End-User Software Engineering. ACM Surveys Vol. 43, No. 3, Article 21, April 2011; http://faculty.washington.edu/ajko/papers/Ko2011EndUserSoftwareEngineering.pdf
12. Loka RR (2007) Software Development: What Is the Problem? Computer 40(2):110–112
13. Victoria R (2011) Software Engineering: Art or Science. SD Times, Nov. 8, 2011; http://sdt.bz/content/article.aspx?ArticleID=36088&page=1
14. Steve MC (1998) The Art, Science, and Engineering of Software Development. IEEE Softw 15(1):118–120
15. Naveen G (2006) Art of Software Engineering, Function Point Analysis Examined. Avenue Razorfish. http://people.eecs.ku.edu/~saiedian/Teaching/Sp13/811/Papers/fun-point-analysis-explained.pdf
16. Glass R (2006) Software Conflict 2.0: The Art and Science of Software Development. Books International. ISBN 0977213307
17. Waldo J (2001) DSP Laboratory for Real-Time Systems Design and Implementation: Software Engineering and the Art of Design. Proceedings of the 2001 American Society for Engineering and Education Annual Conference; session 1526. http://www.artima.com/weblogs/viewpost.jsp?thread=7600
18. Votta L, et al. (2004) Measuring High Performance Computing Productivity. Int J High Perform Comput Appl 18 (4):459–473
19. Wilkins D (2002) The Bathtub Curve and Product Failure Behavior. The Reliability HotWire in Weibull.com. issue 21, November 2002; http://www.weibull.com/hotwire/issue21/hottopics21.htm
20. Wood A (2003) Software Reliability from the Customer View. IEEE Comp Soc 36(8):37–42
21. Iyer RK, Rossetti DJ (1985) Effect of System Workload on Operating System Reliability: A Study on IBM 3081. IEEE Trans Software Eng SE-11(12):1438–1448

22. Jun X, Zbigniew K, Ravishankar KI (1999) Networked Windows NT System Field Failure Data Analysis. In: Proceedings of IEEE Pacific Rim Intl' Symp. Dependable Computing (PRDC). , Hong Kong, China
23. Malkawi M, Votta L, Ignatius G, Moore B (2001) Availability Work Products – A Strategic Approach. IEEE Signal Processing Society 5th WSES International Conference, Crete
24. Malkawi M, et al. (2002) Analysis of Failure and Recovery Rates in a Wireless Telecommunications System. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN), pp 687–693
25. Malkawi M, Votta L (2000) Software Systems Availability Modeling and Analysis. Motorola Report and Motorola Symposium on Software Engineering. , Phoenix, AZ
26. Malkawi M (1999) High Availability Models for Common Platform BTS. Motorola Inc. Internal Report #R1999HAM01
27. Sahner RA, Trivedi KS, Antonio P (1996) Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package. Kluwer Academic Publishers, Netherlands. ISBN 0-7923-9650-2
28. Courtney T, Daly D, Derisavi S, Lam V, Sanders WH (2003) The Möbius Modeling Environment. In: Tools of the 2003 Illinois International Multiconference on Measurement, Modeling, and Evaluation of Computer-Communication Systems. Universität Dortmund Fachbereich Informatik, Germany, pp 34–37. research report no. 781/2003
29. Malkawi M, Malkawi A (2005) Spectrum Management and Rebanding. Mobile Radio Technology (MRT) J. June 2005
30. Malkawi M, Malkawi A (2002) A Comprehensive Analysis of External Interference. white paper published at http://www.glob-tel.com/index.html
31. Babcock WC (1953) Intermodulation Interference in Radio Systems. Bell Syst Tech J 32(1):63–73
32. Jacobsmeyer JM (2007) Solving Inermodulation Interference in Radio Systems. Mobile Radio Technology. (MRT) J; July 1, 2007
33. Lucent Technologies User manual Document 401-610-133 Issue 28- Flexnet/Autoplex Wirless Networks Executve Cellular Processor (ECP) Release 24.:4–125–4–127
34. Williams S Analysis of the SSH Key Exchange Protocol. Cryptology ePrint Archive, Report 2011/276. http://eprint.iacr.org/2011/276, 2011
35. Patrick T Buggy software costs users, vendors nearly $60B annually. Computerworld. June 25 2002; http://www.computerworld.com/s/article/72245/Study_Buggy_software_costs_users_vendors_nearly_60B_annually
36. Fiorenza B Cambridge University Study States Software Bugs Cost Economy $312 Billion Per Year. PRWeb Online Visibility from Focus, Cambridge Judge Business School. http://www.prweb.com/releases/2013/1/prweb10298185.htm
37. U.S.-Canada Power System Outage Task Force August 14th, 2003 Blackout: Causes and Recommendations. http://energy.gov/sites/prod/files/oeprod/DocumentsandMedia/BlackoutFinal-Web.pdf
38. Stephenson A, et al. (1999) Mars Climate Orbiter Mishap Investigation Board, Phase I Report on Project Management in NASA, pp 16–22. http://science.ksc.nasa.gov/mars/msp98/misc/MCO_MIB_Report.pdf
39. Lions JL Arian 5 flight 501 Failure, Report by the Inquiry Board. http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html
40. Matz S (2001) GoAhead Stress Test Definition; Motorola Internal Report; Rep. ##R2001ALT01
41. Jones TC (1978) Measuring Programming Quality and Productivity. IBM Syst J 17(1):39
42. Information Management and Technology Division, GAO/IMTEC-92-26 Patriot Missile Software Problem, B-247094, February 4, 1992. http://www.fas.org/spp/starwars/gao/im92026.htm
43. Douglas A (1992) Two disasters caused by computer arithmetic errors. Institute of Mathematical Applications, University of Minnesota. http://www.ima.umn.edu/~arnold/455.f96/disasters.html
44. Dominik GC, Pangan Oliver I (2004) Cultural Influences on Disaster Management: ACase Study of the Mt. Pinatubo Eruption. Int J Mass Emergencies Disasters 22(2):31–58
45. Dobrica L, Niemela E (2002) A survey on software architecture analysis methods. IEEE Trans Software Eng 28(7):638–654

doi:10.1186/2192-1962-3-22
Cite this article as: Malkawi: **The art of software systems development: Reliability, Availability, Maintainability, Performance (RAMP).** *Human-centric Computing and Information Sciences* 2013 **3**:22.