

Software Development in Antagonistic and Dynamic Operational Environments: Experiences and Best Practices for System Development Components in Reform and Modernization Projects¹

Ivan Krsul²
ivan@acis.ufl.edu
ACIS Laboratory
University of Florida³

Jaime Mendoza
jmendoza@aduana.gov.bo
PRISMA Project
Aduana Nacional de Bolivia

1 Abstract

Many software engineering models have been designed to reduce traditional software faults, increasing the likelihood of producing secure software. However, said models are still lacking in techniques for reducing faults resulting from dynamic and antagonistic environments. Of particular interest are environments where software needs to operate securely under stringent and changing judicial and political constraints, such as system development components of reform and modernization projects.

In this paper we refer to environments as *hostile* whenever the abstract business logic is dynamic and changes are antagonistic to the previous business logic, and posit an analysis of software faults resulting from said operational environments and the factors that contribute substantially to their introduction in production environments. During the analysis we identified four factors resulting from hostile environments that significantly increase the probability of operational software faults: Environmental Opacity, Environmental

¹ An abridged and preliminary version of this paper was submitted to the 2nd Symposium on Requirements Engineering for Information Security, October 16, 2002 in Raleigh, North Carolina. The paper presented is available on the author's homepage at <http://www.acis.ufl.edu/~ivan/articles/hostileSREIS2002.pdf>

² Ivan Krsul was the Director of the PRISMA (Proyecto de Reforma en la Implementación de Sistemas Informáticos Aduaneros) project at the Aduana Nacional de Bolivia from March 2000 to December 2001 and participated in the implementation of the ASYCUDA++ system for the Aduana Nacional de Bolivia since September 1999.

³ Author's address: P.O. Box 116200, 337 Larsen Hall, Gainesville, FL, 32611-6200, Phone:(352)392-9751

Variability, Environmental Complexity, and Environmental Bias, and conclude that new mechanisms that pay particular attention to dynamic and changing operational environments are needed.

In this paper we present a series of best practices oriented towards reducing dynamisms, reducing complexity, increasing transparency, and increasing overall system quality throughout the system development project. These practices are the result of our experiences with system development projects in reform and modernization programs and are exemplified the implementation of the ASYUDA++ system in the Bolivian National Customs Agency. The practices presented are light-weight in that no particular development framework needs to be chosen and can be readily applied without extensive training or highly qualified technical personnel. As a matter of fact, the practices documented are independent of the development framework or methodology chosen, if one is chosen at all.

2 Keywords

SOFTWARE ENGINEERING Requirements/Specifications, SOFTWARE ENGINEERING Tools and Techniques, SOFTWARE ENGINEERING Management

3 Introduction

Reform and modernization programs for governmental institutions or agencies are frequently accompanied by system development projects. By definition, these reform and modernization programs are established to change existing practices in favor of accepted best practices. Hence these programs operate in environments that are changing, chaotic, unpredictable and frequently resist the reform efforts. System development components of such programs are frequently expensive and time consuming. As predicted by the Capability Maturity Model [1], they are frequently late in delivering the desired products, require substantial resources beyond those initially budgeted, and produce results that leave much to be desired in regards to functionality and security assurances.

The field of software engineering, as shown in Figure 1, offers a wide variety of development frameworks or models that can substantially increase the likelihood of a successful implementation in the time specified, with the desired functionality and with the budget established. In our experience in reform and modernization

projects⁴, as the figure shows, system development components of modernization projects typically fail to incorporate these frameworks because of the high startup-cost of using these frameworks and because frequently the personnel involved in the systems development projects are not sufficiently knowledgeable. Hence, these projects typically operate with little assurances with regards to quality, timeliness and execution within established budgets⁵.

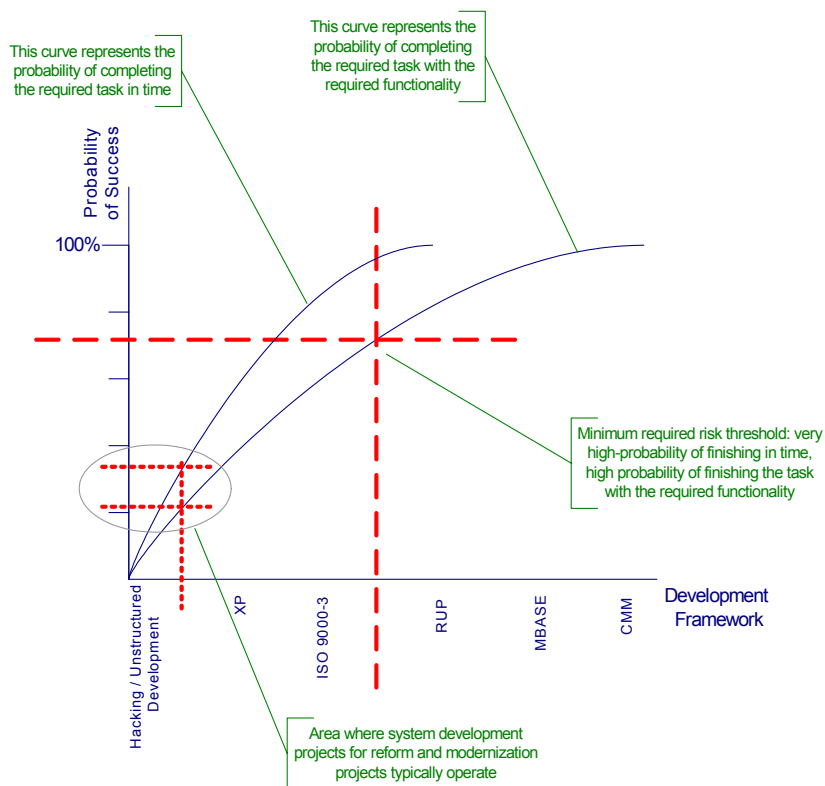


Figure 1

The use of formal development frameworks can have a dramatic impact regarding the probability of delivering products in time and with the desired functionality. Typically system components for reform and modernization projects operate with development frameworks that offer little or no guarantees.

⁴ Although the experiences documented in this document are limited to projects in Latin America, informal polls for similar development projects in Europe, Africa and the United States indicate that reform and modernization projects with similar characteristics have similar problems.

⁵ From a sample of 15 reform and modernization projects that the authors were a part of in Bolivia, Argentina, Guyana, Venezuela, Haiti, Honduras, and Salvador.

Take, for example the Capability Maturity Modeling for Software [1], a widely accepted conceptual ideal in terms of management practices for good software development. It is a model that achieves continuous software process improvement based on four large, evolutionary steps, but achieving each of these steps requires a lengthy and time consuming process. ISO 9000-3 also requires substantial amount of training before the process can be applied and consulting firms generally estimate anywhere from two to five years for certification⁶. Similarly, other relevant frameworks, including such techniques outlined in Rapid Development [14]; The Personal Software Process [8]; and Extreme Programming [7], have high start-up costs and lengthy initial training processes.

In system development projects within reform programs, as shown in Figure 2, the choice of development framework is influenced by three main variables: Ease of Use, Speed of Development and Quality Assurance. Unfortunately, with the current system development frameworks it is not possible to maximize all three variables. The projects described typically choose a development framework that maximizes speed and ease of use effectively minimizing the possibility of producing high-assurance systems.

⁶ Consulting firms estimate, in the experience of the authors, far below the actual execution times for these projects. In the case of ISO-9000 certification for the Peruvian Customs Agency, for example, certification was estimated at 36 months and achieved over a period that exceeded 8 years.

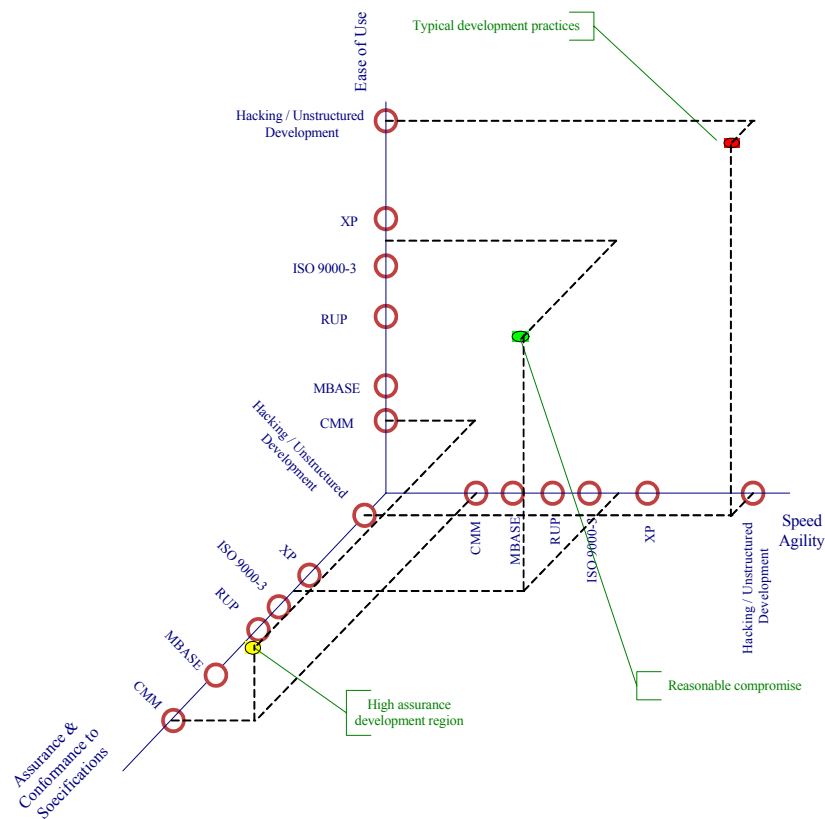


Figure 2
 The choice of development framework is influenced by the three variables shown: Ease of Use, Speed of Development and Quality Assurance. The projects described typically choose a development framework that maximizes speed and ease of use, effectively minimizing the possibility of producing high-assurance systems

These system development frameworks were designed under the assumption that the entire organization is committed to a single unambiguous goal and is working as a whole with the system development team to achieve it. In reform and modernization projects it is frequently the case that the systems development project is designed by multiple parties with possible conflicting interests. Involved during the design and implementation of a system are international donors, multinational agencies, regulating bodies and governmental agencies that can result in a wide range of disagreements on specific goals, increased complexity, significant design bias and vague or contradictory design specifications.

Figure 3 is a simplified representation of the spheres of influence from different organizations on a typical system development project within a reform program. It is a simplified depiction of some of the institutions that directly affected the design and deployment of the systems development project that implemented the ASYCUDA++ [12] system at the Bolivian Customs Agency during the period September 1999 through February 2001, and that we explore in more detail in section 5.

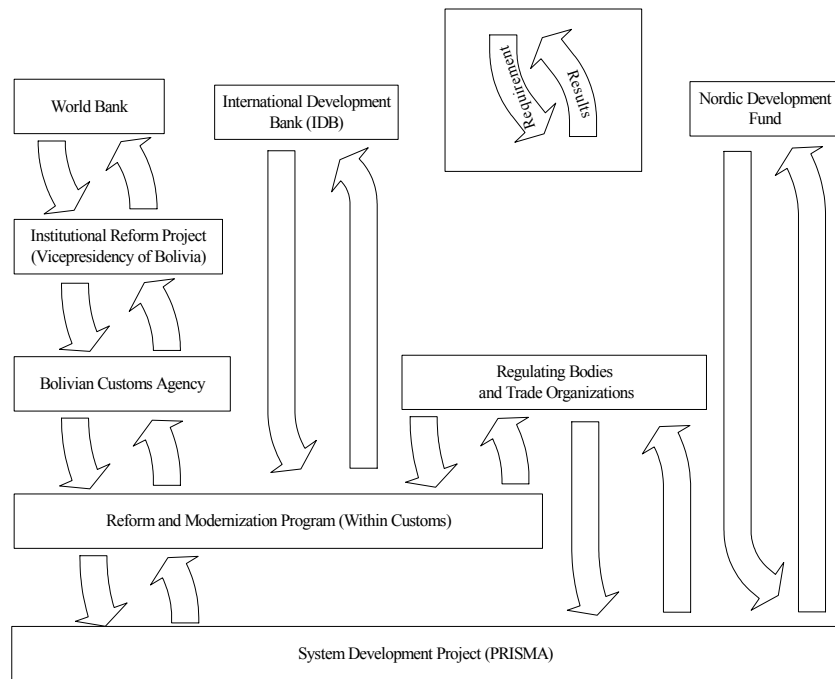


Figure 3
A simplified depiction of the spheres of influence in the design of the system project that implemented the ASYCUDA++ system at the Bolivian Customs Agency

In this case, the global goals of the system were defined by the customs agency through a Reform and Modernization Program –that worked closely with Customs Brokers and the Customs Agency– and presented a detailed project to international donors, who during the first year of the project had influence in the modification of the global goals to meet eligibility requirements shifting the global goals until early in the deployment stages. The Reform Program attempted to centralize the influence of regulating bodies and trade organizations with limited success, as donors, regulating bodies and trade organizations insisted on being involved directly with the development team.

In such projects, best practices addressing the issues of agreement on global goals, definition of requirements for the system development group and homogenization of reporting and follow-up mechanisms are virtually non-existent. Hence, the probability of success of the projects, given ambiguous requirements and poor visibility and follow-up, is drastically reduced. These projects are typically opaque (management cannot easily visualize the

internal state of development and cannot readily identify problems before they are catastrophic⁷), complex (have many components and many interrelationships among the components) and dynamic, hence decreasing our confidence in predicting outcomes [17].

3.1 Software Development in Reform and Modernization Projects

Computer Systems are instantiations of requirements derived, generally, from a logic that defines the expected results. In off-the-shelf software, the designer defines the logic a-priori and users must adapt to the system. In large organizations, requirements are derived from a *business logic*⁸ and the policies and regulations that are relevant. In governmental organizations, the business logic and the system requirements are derived from regulations (i.e. laws and decrees), the resulting procedures that interpret the regulations and document the necessary details, and the policies defined by the appropriate governing body. Once defined, regulations and procedures are unlikely to change drastically, but policies can change dramatically in short periods of time.

Software needs to be secure and security, in its most general definition, can only be provided when the business logic used for the requirements analysis matches closely that of the business logic in place by the time the software is deployed. Given that the business logic of the organization encapsulates high-level security requirements, including access control, authentication, authorization, availability, privacy, etc., if the business logic in place by the time the software is deployed is different from the one used to define the security requirements; it is unlikely that the system will implement the correct security model.

⁷ Total transparency (i.e. seeing every detail at all times) is not always desirable, as too much detail can be overwhelming, but rather systems should be designed so that administrators and engineers can visualize the necessary information whenever it is required. See the [17] for more details.

⁸ The business logic of an organization is the set of high-level objectives defined for the organization and hence implicitly for the systems that must operate in that organization. The business logic is equivalent to the development goals described in [10].

Software engineering life cycles divide the development process into a number of discrete steps that range from conception, through requirement analysis, development to release and operation. The importance of a correct design has long been recognized and there are many techniques for assuring that the software indeed meets end-user expectations, from the use of critics in the design environment [6] to effective risk management [14].

This is particularly relevant for software that is built in environments that are subject to constant change. In [13] Finkelstein and Kramer lists *Change* as a current research challenge: “How can we cope with requirements change? How can we build systems that are more resilient or adaptive under change? How can we predict the effects of such changes?” To this list we add: What effects does change have on the security of software? How can we build systems that are secure in spite of change?

Regardless of the software engineering life cycle used for a systems project, at each step in the development process engineers make a number of assumptions regarding the environment where the software will finally operate. At each step, each incorrect environmental assumption introduces a small error that has a cumulative effect on subsequent stages in the process. The error introduced is not the product of an incorrect design, a misunderstanding of the specifications, or the product of software faults. The design is correct and, in the absence of other software faults, the implementation is correct for a system that was designed and built to operate in an environment differs from the environment in which it will actually operate.

Developers that have made faulty assumptions regarding the environment in which the system will operate, and have not introduced any traditional software faults, will build a perfectly correct system that *would* operate as designed in the environment presumed. As such, the system is devoid of software faults. There are no design errors, no mistakes in the development process, no coding flaws, yet the system will fail to operate as expected, and sometimes the failures can be catastrophic.

If the abstract requirements for the system change during the development and such changes are antagonistic to the business logic in place at the time requirements were specified, the effect is equivalent to that of the developer making faulty assumptions regarding the operational environment.

When a system is built in the presence of dynamic and antagonistic requirements, we may postulate that it will operate in a “Hostile Environment”. Hostile not in the sense of the environment is purposely trying to damage the system, but nevertheless managing to do so. We perceive a natural resistance towards the system from the environment that manifests itself as defects in the design or coding of the system.

Software that operates in hostile environments fails to operate, as pointed out before, not because it was badly designed, or badly coded, but simply because it was designed for an environment that is no longer present by the time the system is operational.

4 Categorization of emergent problems

In this section we present a categorization of software faults for emergent or environmental faults, where these manifest themselves, mainly due to a hostile environment, during operation of the software.

“The context of software system development is changing. Systems are rarely developed from scratch; most system development involves extension of preexisting systems and integration with ‘legacy’ infrastructure. These systems are embedded in complex, highly dynamic, decentralised organisations; they are required to support business and industrial processes which are continually reorganised to meet changing consumer demands.” [13]

This is the case of system development projects being undertaken in developing countries⁹, and the implementation of the ASYCUDA++ system at the ANB is no exception, since this customs organization experienced drastic changes during implementation, creating enormous environmental uncertainty.

In particular, we found it necessary to extend the classification proposed by [3] to describe the conditions that contribute significantly to the development of software systems that are correctly designed and implemented, as required by existing software engineering practices, yet produce significant faults during operation.

⁹ Based on the MARIA system (Argentina), the Lucía system (Uruguay), the implementations of ASYCUDA in Venezuela and Bolivia, the implementation of the SIF system in Honduras, The Red Social Project (<http://www.redsocial.org>), the SIGADE projects, etc.

As such, we traced the causes for the operational problems identified during the deployment of the ASYCUDA++ system and identified the following factors as contributing substantially to the creation of hostile environments:

Environmental Opacity refers to the degree of opacity in the operational environment, during the design, coding or deployment. The environment is opaque if developers are unable to “see” the characteristics of the operational environment [17].

Environmental Variability refers to the “*rate and volume of changes in the environmental factors. Rapid and large volume of changes could decrease confidence in predicting outcomes....*” [3].

Environmental Complexity “*refers to the diversity and interdependence of environmental factors that organizations have to contend with...*” [3].

Environmental Bias refers to the degree of bias – wishful or prejudiced thinking – in the design of the systems with respect to the actual operational environment.

Hence, we postulate that hostile environments are opaque, dynamic, complex and biased. In particular, by their very nature it is difficult to determine the characteristics of the environment as many are hidden or not readily evident, the characteristics of the environment change much faster than software engineers can adapt to the changes.

5 A Case Study

In this paper we present as case study the Bolivian National Customs Agency (Aduana Nacional de Bolivia – ANB), which for two and a half years, starting September 1999, pursued an aggressive development schedule for the adaptation of the ASYCUDA [12] system to the existing regulations and policies defined by the ANB. ASYCUDA is a computerized customs management system which covers most foreign trade procedures and has been successfully implemented in over 80 countries. The organization used domain experts with more than 10 years of field experience in the design of the software requirements and standard software engineering models and practices to reduce faults, yet posterior analysis of 178 problems reported in the development process shows that the majority of problems experienced were not due to inadequate requirement analysis, poor development

practices or flawed software, but to rapidly changing regulations and policies, which in effect were creating an operational environment where it seemed that there were software faults, when in reality the abstract business logic had changed and the software systems had not followed these changes. Additionally, the changes in regulations and policy observed were frequently antagonistic to those previously in place

Developed in Geneva by UNCTAD (United Nations Conference on Trade and Development) [11], the ASYCUDA system is very configurable and traditionally installed without modifications to the source code, and the implementation efforts are limited to the adaptation of the local customs procedures to conform to the Kyoto convention, and the harmonization of documents and codes. The implementation of the ASYCUDA software (version 1.16, also known as ASYCUDA++) at the ANB differed considerably from the traditional implementation model used in the countries where this system was deployed¹⁰, as extensive modifications were required to the source code of the system to conform to stringent regulatory and political requirements imposed by the Bolivian government. A reform and modernization project called PRISMA¹¹ was created with a team of 18 people for the effect.

This team used domain experts with more than 10 years of field experience in the design of the software requirements and standard software engineering models and practices to reduce faults, including:

- CMM for software model [1, 2] (in this particular implementation the PRISMA team reached level 3¹²).
- Techniques recommended for rapid development [14].
- Standard Risk Analysis and Requirement Management practices.
- Practices recommended for ISO 9000-3 certification. [15]

¹⁰ Bolivia is the first country where the source code was released and all previous implementations were off-the-shelf. The software could be configured and regulations and policies had to be adapted to fit the system.

¹¹ PRISMA: Proyecto de Reforma en la Implementación de SisteMas informáticos Aduaneros.

¹² No formal certification mechanisms were used. The implementation team self-assessed the level in three month intervals.

- Standard requirement management practices, including goal management and follow-up.
 - Quality control practices conforming to ISO 9000-3, tightly integrated into the production environment.
- [15]
- Standard software testing techniques, including testing, code walk-through, operational simulations, etc.

Said models and practices successfully reduced traditional software faults in the development stages, yet the ASYCUDA++ exhibited a significant number of operational problems that had mean-repair times in excess of 12 hours¹³. Most of these operational problems resulted in violations of explicit or implicit security policies, as security requirements were embedded into the system architecture.

5.1 Data Collection

During the implementation of the ASYCUDA++ system at the ANB the PRISMA project kept a detailed record of 178 different problems considered *hard to fix* by the development team – problems that were not solved within 12 hours of being reported. 60% of these problems were identified in a workshop with users of the system four months after initial deployment, and the members of the development team reported the remaining 40% six months after the initial deployment.

For each of the problems reported, we measured keywords using the following criteria:

Importance: Describes the importance of solving the identified problem. Define importance as a function of the severity of the effects of not solving the problem. If the effect is that the system can cease to function, we classify the problem as *very important*. If the effect is that the system malfunctions, but still produces meaningful results, we classify the problem as *important*. If the effect is that the system still functions correctly, we classify the problem as *unimportant*.

¹³ In this paper we limit our analysis to the problems reported during the implementation of the export control phase of the project and cover the period March 2000 to October 2001.

Urgency: Describes the perceived urgency of solving the problem by the development team. If the problem needs to be fixed within a month, we classify the problem as *very urgent*. If the problem needs to be fixed within three months, we classify the problem as *urgent*. If the problem can be solved after three months, we classify the problem as *not urgent*.

Impact: Measures the dispersion of the effects of not solving a problem. A problem has *global impact* if all users of the system will feel the effects of not solving the problem, *institutional impact* if all users of the institution that developed the system will feel the effects of not solving the problem, and *local impact* if only developers would feel the effects of not solving the problem.

In addition, various yes/no features were measured for each problem as follows:

Type of Problem	Description
Normative	These are problems that result from incorrect or incomplete norms or procedures defined <i>a-priori</i> . These norms or procedures define the desired functionality of the systems built.
Systems	These are problems that result from incorrect system design or coding.
Infrastructure	These are problems that result from deficient or non-existent infrastructure that is required for the system to operate.
Administrative	These are problems that result from deficient administration.
Support	These are problems that result from poor customer support.
Training	These are problems that result from poor training.

Table 1 – Features measured: type of problem

Entity or group that can solve the problem	Description
Internal	The systems group/developers can solve the problem.
Institution	Any other group in the institution (i.e. not the developers) can solve the problem.

External	People outside the institution can solve the problem.
----------	---

Table 2 – Features measured: group that can solve the problem

Type of solution identified	Description
Policies	Solving the problem requires the creation of policies – these tend to be administrative, but can also be legal, technical or normative.
System / Technical	Solving the problem is a matter of applying existing technical resources – generally fixing bugs, system configuration, network configuration, etc.
Normative	Solving the problem requires the development of norms or procedures.
Coordination	Solving the problem requires that two or more groups –independently administered– coordinate efforts.
Communication	Solving the problem requires communication – includes emailing people, distributing documentation, publishing notices, etc.
Planning	Solving the problem requires a significant amount of further planning – generally because existing methods or techniques do not apply.

Table 3 – Features measured: type of solution identified

Cause of Emergent Fault	Description
Environmental Bias	The developers identified that the problem was caused in part because at some stage during the implementation of the system, somebody made a significant decision that was biased (i.e. it was done based on wishful thinking).
Environmental Opacity	The developers identified that the problem was caused in part because it was not possible to clearly identify the characteristics of the actual operational environment.
Environmental Variability	The developers identified that the problem was caused in part because the operational environment varied significantly between design/coding and operation.

Environmental Complexity	The developers identified that the problem was caused in part because the operational environment had a significant amount of complexity.
--------------------------	---

Table 4 – Features measured: cause of emergent fault

Security Policy Violated	Description
Integrity	Problem resulted in a violation of implicit/explicit integrity policy
Accountability	Problem resulted in a violation of implicit/explicit accountability policy
Access Control	Problem resulted in a violation of implicit/explicit access control policy
Availability	Problem resulted in a violation of implicit/explicit availability policy
Authentication	Problem resulted in a violation of implicit/explicit authentication policy
Privacy	Problem resulted in a violation of implicit/explicit privacy policy

Table 5 – Features measured: security policy violated

The following table summarizes the findings:

Result	% of Problems
Problems classified as "Not Urgent"	40%
Problems classified as "Urgent"	23%
Problems classified as "Very Urgent"	37%
Problems classified as "Not Important"	40%
Problems classified as "Important"	42%
Problems classified as "Very Important"	18%
Problems that were caused by one or more of Environmental Variability, Environmental Complexity, Environmental Bias or Environmental Opacity.	65%
Problems that were caused by Environmental Variability	30%

Problems that were caused by Environmental Complexity	37%
Problems that were caused by Environmental Bias	21%
Problems that were caused by Environmental Opacity	24%
System problems of technical nature that could be solved by the development team and where the solution was technical.	11%
Problems that could not be solved by the development team.	58%
Problems that did not have a systems technical solution.	86%
Normative problems	24%
System Problems	12%
Infrastructure Problems	15%
Administrative problems	53%
Support Problems	7%
Training Problems	20%
Problems whose solution required the generation of policies.	26%
Problems whose solution included technical solutions.	14%
Problems whose solution included the development of further norms, procedures, laws, etc.	22%
Problems whose solution requires significant amounts of inter-group coordination.	49%
Problems whose solution requires communication.	29%
Problems whose solutions required significant amounts of further planning.	47%
Integrity	25%
Accountability	16%
Access Control	9%

Availability	28%
Authentication	11%
Privacy	4%

Table 6 – Summary of finding for 178 problems collected

5.2 Co-word Analysis of Operational Software Faults

In this section we present the result of applying co-word analysis to the 178 problems mentioned previously for the project implementing the ASYCUDA++ software for the ANB. As mentioned before, this project used standard software engineering models to reduce operational faults, and successfully reduced traditional software faults in the development stages, yet exhibited a significant number of problems that had mean times to repair in excess of 12 hours.

The data collected in section 5 can be easily converted into a database where each problem is described as a set of related keywords and visualizations techniques can be applied to discover the sources of the problems or strengthen our confidence in the a-priori classification developed in section 4.

Co-word analysis is a content analysis technique that is effective in mapping the strength of association between keywords in textual data. Co-word analysis reduces a space of descriptors (or keywords) to a set of network graphs that effectively illustrate the strongest associations between descriptors [20, 21].

Co-word analysis is an example of a graphical modeling technique that applies some of the ideas of association analysis [18, 19]. Graphical modeling is a variant of statistical modeling that uses graphs to display models. “In contrast to most other types of statistical graphics, the graphs do not display *data*, but rather an interpretation of the data, in the form of a *model*... Graphs have long been used informally... to visualize relations between variables.” [18].

This technique illustrates associations between keywords by constructing multiple networks that highlight associations between keywords, and where associations between networks are possible.

For co-word analysis, every field in a database can be used as a keyword, or can be transformed to a series of keywords by applying the following rules:

1. If a field in the database can have the values yes, no, ?, and NA, then a keyword with the name of the field is generated if the value of the field is yes.
2. If the field in the database can have a single value from a list then a keyword with the name of the field followed by the value of the field is generated.
3. If the field in the database can have a list of values from a well-defined set, then for every value in the field we generate a keyword with the name of the field followed by the value of the field.

We applied these rules to the features measured for the 178 problems identified during the deployment of the ASYCUDA++ system as mentioned in section 5, and used the co-word tool originally developed for [16] at Purdue University and described in [9], and obtained from our data the networks shown in Figures 4, 5, 6 and 7 networks for the parameters *max pass 1 links* = 5, *max links* = 10, *min cword* = 25.

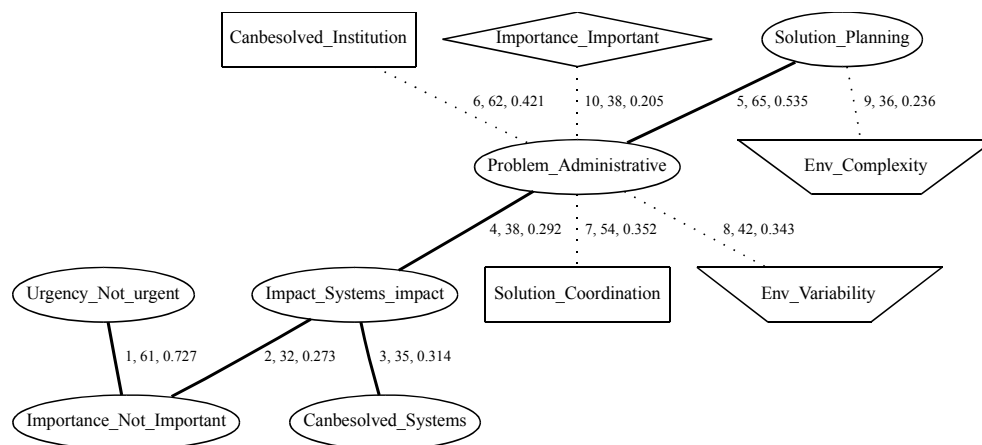


Figure 4: Graph No. 1. Density = 0.5, Centrality = 0.72

This graph shows that the problems identified were frequently administrative problems that had a systems impact, required significant amounts of further planning for the solution, were problems that were not urgent or important, and the problems had technical solutions. These problems were generally the result of policy changes that resulted in changes to the desired functionality during the development stages.

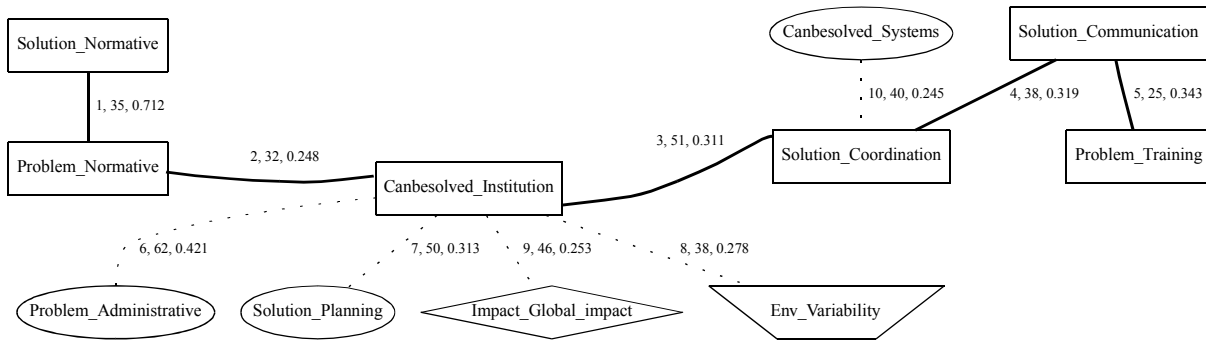


Figure 5: Graph No. 2. Density = 0.48, Centrality = 0.69

This graph shows a group of problems that were normative in nature (i.e. the regulatory framework changed or was re-interpreted during implementation), could not have been solved by the system team, required significant amounts of coordination and communication and required training or re-training personnel on the use of the system, regulations, etc. These problems are typically the result of major changes in regulation or policy, or the result of drastic changes in the interpretation of the regulations or policies during development.

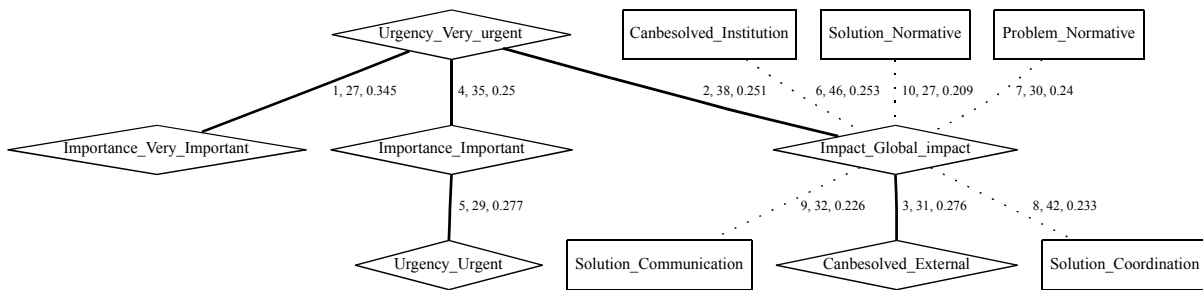


Figure 6: Graph No. 3. Density = 0.3, Centrality = 0.52

This graph shows that a group of problems identified affected the development team, the institution and the end-users (global impact), could be solved by parties external to the institution, and were very urgent and very important. These problems were typically the result of resistance by the end-users to the regulations and policies defined by the ANB (and instantiated by the system). The problems typically required political solutions that significantly affected the development team.

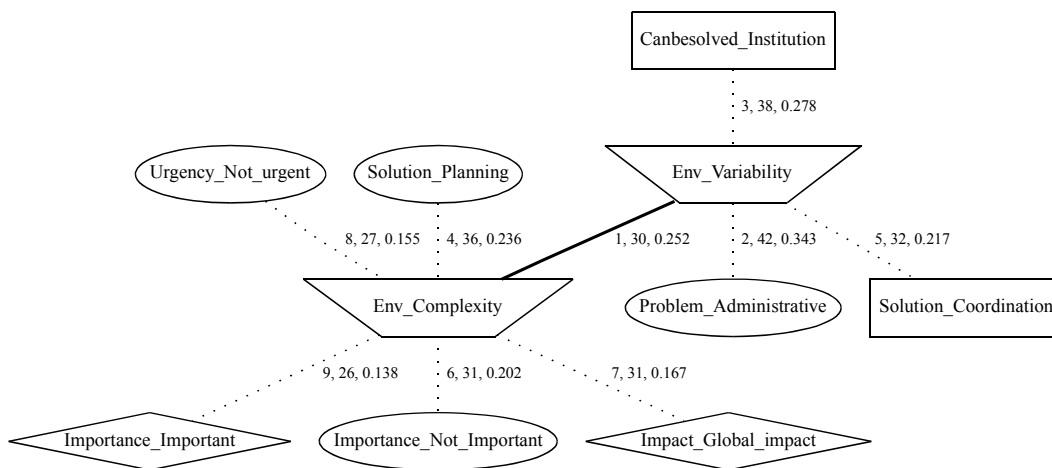


Figure 7: Graph No. 4. Density = 0.25, Centrality = 0.64

This graph shows frequent associations between Environmental Variability and Environmental Complexity. This is to be expected, as complex system are more likely to have problems when the environment changes.

In these networks, solid lines connecting nodes represent pass-1 links and dotted lines represent pass-2 nodes. Each link is labeled with a triple $\langle \textit{number}, \textit{co-occurrence}, \textit{strength} \rangle$ that indicates the order in which the links were added (1 corresponds to the first link), the co-occurrence for the key pair, and the strength of the link.

The networks generated by the co-word analysis tool show the distribution of their centrality and density values. The resulting networks that have the highest centrality and density values represent the more predominant relationships among the keywords in the data. Isolated networks identify relationships with keywords that have low centrality values and hence the keywords are infrequently used in other networks. Isolated networks with high-density values indicate strong relationships among keywords in isolated groups, and point to dominant features of the database.

5.3 Interpretation of Results

The data collected, as summarized in Table 6, shows that aggressive use of standard software engineering methodologies is successful in reducing the incidence of traditional software faults, as 89% of the problems recorded were not systems problems that could be solved by the development team using technical tools at their disposal.

However, the total error was far larger than expected, as 86% of the problems did not have a technical solution that could be applied by the development team.

Particularly interesting is that the solution to 49% of the problems identified required significant amounts of coordination and the solution to 47% of the problems required significant amounts of further planning. This is an indication that the development team probably did not prepare adequately for the types of problems encountered. Since 65% of the problems were caused by one or more of Environmental Complexity, Environmental Bias, Environmental Opacity or Environmental Variability, it is reasonable to assume that a significant fraction of the problems encountered require either more coordination efforts or further planning.

Figures 3 and 4 show how most problems encountered required coordination and planning as a significant component of the solution, with 24% of problems being normative (relating to norms, procedures or law) and 53% of the problems being administrative (relating to general systems and institutional administration). Such

problems frequently arose because of either unexpected changes or unexpected complexity in the operational environment.

Figure 6 shows the strong correlation between Environmental Complexity and Environmental Variability, and shows that the problems identified typically required significant amounts of coordination and planning and that the development team alone could not have solved said problems.

During the final stages of development, or early in the deployment of the system, the business logic (equivalent to the development goals described in [10]) had changed without due consideration to the effects on system development. Rather than the exception, such changes tend to be the norm in environments that operate under evolving political and/or judicial constraints. It is noteworthy that some of the institutional goals (as defined in [10]) were hidden from the development team and/or domain experts because of political reasons¹⁴.

Finally, it is interesting to note that 25% and 28% of the reported problems resulted in a violation of implicit or explicit integrity or availability security policies, while only 9% and 4% of the problems violated an access control or privacy policy.

6 Best Practices

During the implementation of the ASYCUDA++ system in the Bolivian National Customs Agency (Aduana Nacional de Bolivia – ANB), we found that current software development practices, though technologically sound *per se*, are not tightly integrated with effective mechanisms for dealing with hostile environments. Through the application of traditional software engineering methods, the PRISMA projects was successful in reducing traditional software faults, but was largely unable to prevent a significant number of problems that resulted from operational environments that were either rapidly changing (variable), complex, opaque or not as expected during the design (biased).

¹⁴ The mere fact that we may have hidden goals makes the requirement analysis challenging, but this is beyond the scope of this paper.

Systems such as the ASYCUDA++ customs software typically require high levels of security and current software engineering models are unable to deal with stringent and changing judicial and political operational environments, particularly those where change cannot be predicted a-priori.

To avoid this type of problems, new mechanisms are needed that pay particular attention to dynamic and changing operational environments [13], including but not limited to:

- Qualitative observations of the decision making process used by legislative and political bodies.
- Computer security models for dynamic operational environments.
- Tools that provide better support for the development of software in changing environments.
- Tools and techniques for predicting hostile environments.
- Best practices specifically tailored for system development in hostile environments.

We argue that software development mechanisms for hostile environments, as shown in figure 8, should provide mechanisms for adapting the software development cycle to shifts in global goals, particularly when the shift in goals result in conflicts with the actual implementation¹⁵.

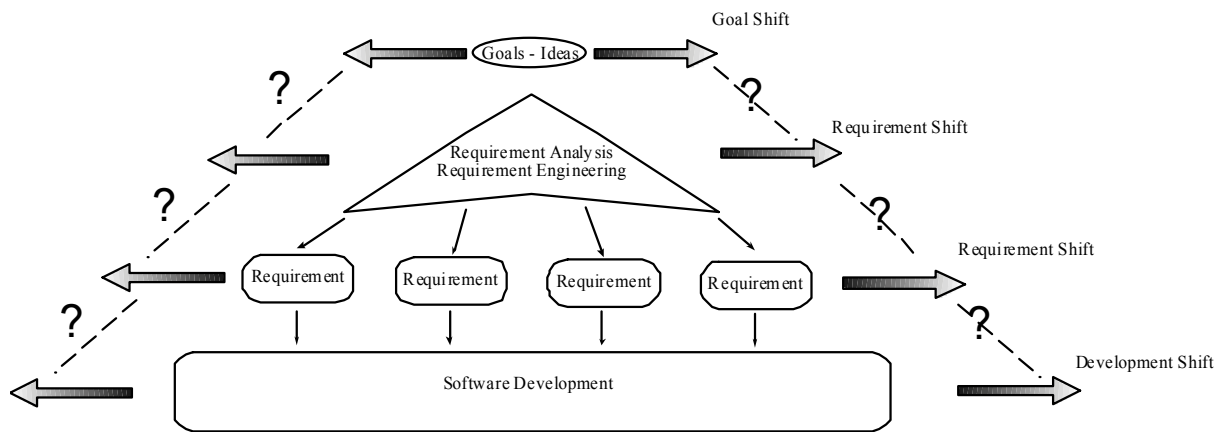


Figure 8: Goal Shifts Must Result in Implementation Shifts

¹⁵ Note that these are typical of the applications in the domain examined and we cannot generalize to application domains where there can be no goal shift (for example, in the design of a laser operating device)

Alternatively, mechanisms should exist that prevent the shift in global goals when the software is unable to follow suit, by providing effective feedback to policy makers regarding the cost and effort required to adapt an evolving system to changing global goals.

In this paper we present a set of best practices that resulted from our experience in the implementation of the ASYCUDA++ system at the Bolivian Customs Agency during the periods of December 1999 through February of 2002 and that addressed the following issues:

1. Project visibility and project status reporting [VISIBILITY].
2. Reduced shifts and changes in goals and requirements [DYNAMISM].
3. Reduced complexity [COMPLEXITY].
4. Reduced bias [BIAS].
5. Focus on quality [QUALITY].

Some of these practices were weaved into the fabric of the project from the onset and substantially improved the likelihood of success of the project, in spite of constant pressure from the Customs Agency, the Reform Program, international donors and trade organizations to change the global goals and under constant financial uncertainty. Other practices were adopted as the project matured and some are the result of posterior analysis on the implementation efforts.

Every system development project, regardless of the development framework chosen, must complete, formally or informally, the steps shown in figure 9 and with the application of the basic best practices we present in this paper, and that we believe do not require extensive training or lengthy start-up processes, we increase the chances of success of projects within acceptable parameters as shown in Figure 1.

The best practices we present are oriented towards reducing dynamisms, reducing complexity, increasing transparency, and increasing overall system quality throughout the system development project as general guidelines and in cases in specific stages of development. These practices are light-weight in that no particular development framework needs to be chosen, and can be readily applied without extensive training or highly

qualified technical personnel. As a matter of fact, the practices documented are independent of the development framework or methodology chosen, if one is chosen at all.

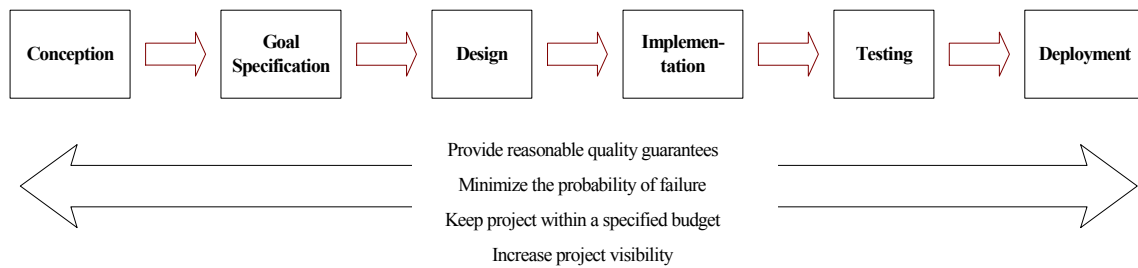
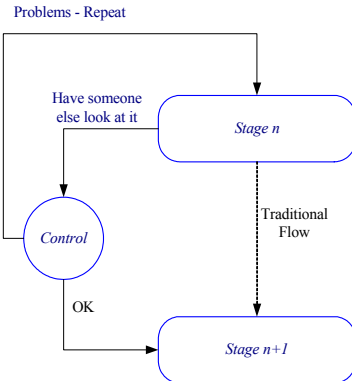


Figure 9
 Every system development project, regardless of the development framework chosen, must complete, formally or informally, the steps shown. The ideal practices are those that can guarantee the constraints shown.

6.1 Overall best practices

Practice	Effect of applying the practice
<p><i>Plan, schedule and budget using worst-case rather than best-case or average-case scenarios.</i></p> <p>Reform and modernization projects typically operate under adverse conditions precisely because the institutions where these systems will operate are in need of reform. Hence, even average-case scenarios are rarely if ever possible.</p>	Reduce BIAS
<p>Include quality control as an integral part of the project. In our experience, the easiest mechanism for implementing quality control is illustrated in Figure 10. Project management establishes a system for assigning tasks to a particular person or group and only considers the task complete when a third person, not involved in the execution of the task certifies the task as correct. This system can easily be deployed by creating a database of tasks assigned to groups or project members and meticulously keeping track of what tasks were certified and by whom.</p> <p>In a particularly successful implementation of this system implemented in the deployment of the ASYCUDA++ system at the Bolivian Customs Office, a simple Web-based database was implemented for documenting the tasks assigned to project members and the database kept track of what tasks were completed and who vouched for the task execution. The project would not consider a task complete until the quality control field was filled.</p>	Increase QUALITY

Practice	Effect of applying the practice
<div style="text-align: center;">  <p>Figure 10</p> <p>Quality control can be achieved easily in projects that have limited or no experience with the subject by requiring that any task performed be certified as correct by a project member that was not involved in the development of the task.</p> </div>	
<p>Digitize everything into a structured database and publish the documentation as widely as possible. Reform and modernization projects in general, and system development projects in this context in particular, operate in hostile environments and the practice of digitizing every letter, response, bidding document, purchasing document, receipt, etc., significantly reduces mistakes and contributes to overall project transparency. Also, auditing processes with funding agencies is simplified enormously when the information is available electronically.</p>	<p>Increase VISIBILITY, increase QUALITY</p>
<p>If possible, make all documents public. Goals are less likely to change if they have been widely disseminated and policy makers will be more careful in outlining desired deliverables if they are aware that the documents produced will be widely available.</p>	<p>Increase VISIBILITY, reduce DYNAMISMS, increase QUALITY</p>

6.2 Best practices for the Conception stage

Practice	Effect of applying the practice
<p>Document high-level deliverables in such a way that evaluation is possible. Avoid vague and blurry concepts and try to include in the documentation the mechanisms that will be used to measure the degree of success for each conceptual deliverable.</p>	<p>Reduce BIAS</p>
<p>Keep this document up-to-date.</p>	<p>Reduce BIAS, reduce</p>

Practice	Effect of applying the practice
	COMPLEXITY
Have this document signed and approved by high-level management and funding agencies. Typically high-level management and funding agencies sign on to vague deliverable statements. Having them sign a detailed and measurable set of deliverables will reduce substantially the dynamisms of the project.	Reduce BIAS, increase VISIBILITY, reduce DYNAMISMS
Document what constitutes qualified human resources for the project. Have this document signed and approved by high-level management and funding agencies.	Reduce BIAS, increase VISIBILITY, reduce DYNAMISMS
Determine the hiring practices for the project. If the project must use existing human resources, establish the correspondence between the desired and existing resources. If necessary, explicitly plan to train existing resources.	Reduce BIAS, increase QUALITY.

6.3 Best practices for the Goal specification stage

Practice	Effect of applying the practice
Document the goals and deliverables with a much detail as possible.	Reduce BIAS, increase VISIBILITY, reduce DYNAMISM
Have these goals and deliverables signed and approved by high-level management and funding agencies.	Reduce DYNAMISM, increase VISIBILITY

6.4 Best practices for the Design stage

Practice	Effect of applying the practice
Institute a formal mechanism for submitting system requirements, approving or rejecting	Reduce BIAS, reduce

Practice	Effect of applying the practice
them and publish this information widely. In this particular case, a bureaucratic process is desirable because it will naturally filter the requests for change that are not essential for the satisfaction of the formally specified goals.	DYNAMISM

6.5 Best practices for the Implementation Stage

Practice	Effect of applying the practice
Institute a formal mechanism for documenting the development state for formal requirements.	Reduce BIAS, increase VISIBILITY, increase QUALITY
Do not implement anything that is not formally required, including testing and experiments.	Reduce DYNAMISM, increase QUALITY.
Avoid research.	Reduce COMPLEXITY.
Hire human resources that are qualified as defined by the project documentation.	Reduce BIAS, increase QUALITY.

7 Conclusion

Software engineering are lacking in techniques for reducing faults resulting from dynamic and antagonistic environments, particularly for *hostile environments*, where the abstract business logic is dynamic and changes are antagonistic to the previous business logic. Observing 15 software development projects for institutional reform projects in different countries, we identified four factors that significantly increase the probability of operational software faults: Environmental Opacity, Environmental Variability, Environmental Complexity, and Environmental Bias. We presented, as a paradigm of such software projects, the software development project that implemented the ASYCUDA++ system in the Aduana Nacional de Bolivia.

We argue that the software engineering field should develop mechanisms for adapting the software development cycle to shifts in global goals, particularly when the shift in goals result in conflicts with the actual implementation, and in absence of these mechanisms we present a series of best practices oriented towards reducing dynamisms, reducing complexity, increasing transparency, and increasing overall system quality throughout the system development project. The practices presented are light-weight in that no particular development framework needs to be chosen and can be readily applied without extensive training or highly qualified technical personnel. As a matter of fact, the practices documented are independent of the development framework or methodology chosen, if one is chosen at all.

8 Acknowledgments

The authors would like to acknowledge the Aduana Nacional de Bolivia, and in particular to the members of the PRISMA project, for the data collection that made this analysis possible.

9 Bibliography

- [1] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis and Charles V. Weber, *Capability Maturity Model SM for Software, Version 1.1*, Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University, May 1993.
- [2] Mark C. Paulk, *Using the Software CMM in Small Organizations*, Available by request from the Software Engineering Institute, Carnegie Mellon University, 1998.
- [3] Choon-Ling Sia, Hock-Hai Teo, Bernard C. Y. Tan and Kwok-Kee Wei, *Examining environmental influences on organizational perceptions and predisposition toward distributed work arrangements: a path model*, Proceedings of the international conference on Information systems, 1998.
- [4] Taimur Aslam, Ivan Krsul and Eugene Spafford, *Use of A Taxonomy of Security Faults*, Technical Report TR-96-051, September 1996, Purdue University.
- [5] Simson Garfinkel and Gene Spafford, *Practical UNIX and Internet Security*, O'Reilly & Associates, Inc, 1996, Second Edition.

- [6] Gerhard Fischer, Kumiyo Nakakoji, Jonathan Ostwald, Gerry Stahl and Tamara Sumner, *Embedding Critics in Design Environments*, The Knowledge Engineering Review, Vol. 8, pp. 285-307, 1993.
- [7] Extreme Programming in Practice by James W. Newkirk, Robert C. Martin, Addison-Wesley Pub Co., 1st edition, June 2001.
- [8] The Personal Software Process, Watts S. Humphrey, Carnegie Mellon Software Engineering Institute, Technical Report CMU/SEI-2000-TR-022, ESC-TR-2000-022, November 2000.
- [9] CoWord analysis tool. COAST Laboratory (Purdue University), ACIS Laboratory (University of Florida), <http://www.acis.ufl.edu/~ivan/coword>
- [10] Annie I. Antón, *Goal-Based Requirements Analysis*, Second IEEE International Conference on Requirements Engineering (ICRE `96), Colorado Springs, Colorado, pp. 136-144, 15-18 April 1996.
- [11] United Nations Conference on Trade and Development. <http://www.unctad.org>.
- [12] ASYCUDA - Automated SYstem for CUstoms DAta. <http://www.asycuda.org>.
- [13] Anthony Finkelstein and Jeff Kramer, *Software engineering: a roadmap*, Proceedings of the conference on The future of Software engineering, Limerick, Ireland, Pages 3 - 22, 2000.
- [14] Steve McConnell, *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, 1996.
- [15] James L. Lamprecht, *ISO 9000: Implementation for Small Business*, ASQC Quality Press, 1996.
- [16] Ivan V. Krsul, Software Vulnerability Analysis, Ph.D. Thesis, Purdue University, May 1998.
- [17] Dietrich Dorner, *The logic of failure: why things go wrong and what we can do to make them right*, Metropolitan Books, 1996.
- [18] D. Edwards, Recent Advances in Descriptive Multivariate Analysis, Royal Statistical Society Lecture Note Series, Chapter 7, *Graphical Modelling*, pages 135-156, Clarendon Press, 1995.
- [19] Leonard Kaufman and Peter J. Rousseeuw, *Finding Groups in Data*, John Wiley & Sons, Inc., Wiley Series in Probability and Mathematical Statistics, 1990.

- [20] Neal Coulter, Ira Monarch and Suresh Konda, *Software Engineering as Seen Through Its Research Literature: A Study in Co-Word Analysis*, Journal of the American Society for Information Science (JASIS), Volume 49, Number 13, Pages 1206-1223, November 1998.
- [21] J. Whittaker, *Creativity and Conformity in Science: Titles, Keywords, and Co-Word Analysis*, Social Science in Science, Volume 19, Pages 473-496, 1989.