

**Final Report**  
**Assessment of UNICEF-Supported HMIS Software programs in health sector in**  
**Iran.**

**By Reza Alemi,M.D.**  
**May 2002**

**Abstract:**

The present project was carried out to assess the UNICEF-Supported software in health sector in Iran, and to perform a SWOT environmental analysis, so that a plan of action can be compiled for the future strategies in software development and maintenance.

Using WHO and academic guidelines (See References), an assessment tool was developed in the form of a questionnaire, and a conference was held in which key people involved in five different software projects from different parts of the health sector were invited. These included high ranking managers, software engineers and the end users of this software. This group discussed the most prominent problems, most pressing needs, and the most practical ways of solving these problems, and a summation of their views was incorporated to the final plan of action.

There are various strengths and opportunities in the current environment. Managers and policy makers respect the value of information management and are ready to invest in reasonable projects. Software technology and industry can be easily imported and incorporated in the new projects. There is minimum client resistance to change and adaptation to the new strategies; in fact, the end users are the most eager group asking for changes and updates, because they see how a good software solution can help them with their responsibilities.

Whoever, there are also weaknesses and threats. The managers fear having to depend on one programmer or group of programmers to maintain and upgrade the software once it is developed. The programmers complain that there are no defined standards and processes to define what is and what is not their duty, and they have difficulty receiving their payments since there is no official place for a software programmer in the health system. The end users are confused as to what they can and can't expect the programmers to do for them. They don't know which upgrade they could ask for. There is no official way to report a bug on a project.

A plan of action is compiled to address the findings of this investigation. The most pressing need is to construct standard documentations for the existing software. This will eliminate the dependency on the original programmer and facilitate the delegation of maintenance and upgrade to third party software teams. A proposal to conjure a committee for review and selection of information standards is compiled to be handed to the Ministry of health. Necessary steps are outlined to convert the existing software to component based, fully modeled solutions.

**Introduction**

Everywhere a software program is in use, concern is rising that how it is going to be maintained and upgraded. This is neither a problem specific to the health sector, nor is it confined to Iran. In fact, the problem has long been recognized in the industrialized world, and frequently referred to as 'The Software Crisis'.

The primitive way of software development, i.e. the 'Code and Fix' process, which consists of a programmer interviewing a manager to see what he wants, then producing

some code to do the job, and then modifying or debugging the code to suit the new or unforeseen needs, is doomed to end up in a tangled piece of software that is impossible to maintain and upgrade, and also depends entirely on the original programmer, who in many instances doesn't even himself remember the meaning of the parts of his own program.

That is why in the software industry there are emerging emphasis on the development process, insisting on protocols such as object oriented programming, component based software development, software modeling, multi-tier application development, and standard documentation.

The present paper tries to establish the fact that the crisis has reached the health sector of Iran, and to present a selection of solutions to deal with it.

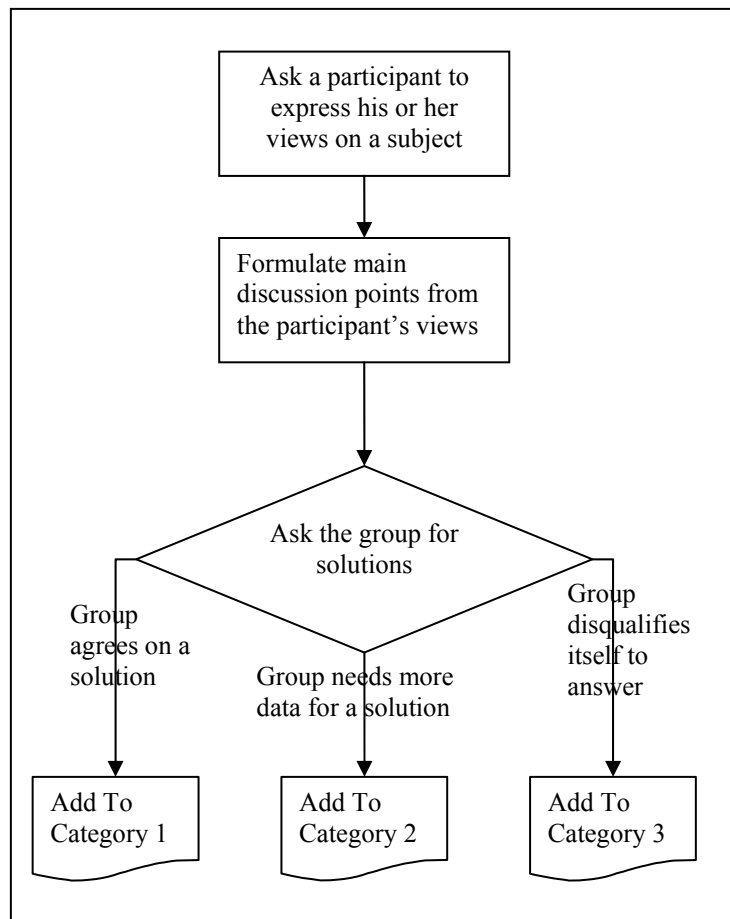
### Materials and Methods

Following the guidelines of chapter 3, pages 73-86, of the *Design and implementation of health information systems* book, published by WHO, 2001, ISBN: 9241561998; a questionnaire was compiled in 3 sections. The first section addressed the high managers, asking them their expectations from the program, their idea of fair expenses for maintenance, development and employment of these programs, and their vision about the future of the program. The second section addressed the programmers, asking the standards they used in their programs, the level of documentation and user guides they provide, and the recommended hardware and equipment they need for their software. The last section addressed the end

users, to find answers on who easy they find the program to learn, share data with other programs, speedup their routine work, give them new tools and looks to cope with their problems and to respond and act on their comments and suggestions.

Synonymous questions were produced in different formats to minimize the misunderstandings and find if the questionnaire was filled intentionally, or randomly.

A meeting was setup by the office of health network development of the ministry of health and medical education, and managers, software developers and end users of five software programs where invited. Those who didn't attend the meeting were interviewed separately or by mail. These projects were the



DTARH project, and The Shemiranat Project, which were supported by the UNICEF, the Malaria project as an example of software developed for surveillance systems, the NISTA project as a research subsystem software, and the Personnel database as an example of administrative software. A couple of other projects, such as the death registry and the physician registry were also evaluated as a pilot assessment.

The meeting was conducted as a focus group discussion. Questions were presented in turn to the group and answers were gathered and utilized to form the basis of further questions. This way, the issues were divided into 3 categories. The first category consists of items on which the whole group agrees. The second category consists of controversial subjects, on which the majority of the group agrees, but they prefer to withhold judgment until further information is at hand, mainly when the issues on the first category is solved. The third category is made of issues that the group disqualifies itself from answering, and suggests other people to address.

A literature review was conducted on the results of the discussion, to see if the stated problems and objectives were addressed anywhere in the current sources. Solutions were gathered and ranked by the level of feasibility, and compiled into a plan of action to address the first category issues. The questionnaires, which were given to the group a week before the meeting, were gathered at the meeting. The answers were summarized in the results section. Those forms in which more than 3 pairs of synonymous questions were given different answers were disqualified as unreliable.

As was stated in the ethical issues of the original proposal, no source code was requested from the programmers. The validity of their answers was determined by comparing their answers for different questions. If, for example, a programmer declares his software to run on a WAN and at the same time says his software shares data with other instances only by disk, the validity of WAN-compatibility claim is waived.

## Results

A summary of the answers provided in the questionnaire are presented below:

**Software Strategies:** while all of the programs were developed to increase the performance of an existing system, to manage high volumes of data, to generate reports at different levels, to increase the quality of data gathering and analysis, and to act as a decision support system, few also aimed to monitor and evaluate the health programs, and to de-centralize data entry. None try to model and simulate a situation, decentralize data analysis, educate health personnel, or use the Internet for data communications and publication. The computerization is done for Registering health facilities and infrastructure, Registering health personnel and monitoring their activities, Monitoring health problems, key interventions, and critical resources, Managing logistics and stock, and Disseminating and archiving data. All programs were based on an existing paper system, some surpassing it by 10-30%, some still not covering all of it (80-95% of the paper system).

**Levels of Action:** The **data entry** is mostly at the district level, although the SHEMIRANAT project has set the level to the lowest possible (i.e. the health house). Most **data analysis** and reports are made at the district, province, and central levels, and again the SHEMIRANAT project has set this for all levels. As for the **transmission of**

**data between levels**, all of the programs have critical data that should be transmitted from the lower to the upper level in a matter of days, although there is also non-critical data that can tolerate longer delay. Only the SHEMIRANAT project needs the data to be accessible to users on different levels at the same time (e.g. through a WAN); other projects think it enough if a network of about 10 people on the same LAN can have access to the data simultaneously. **Feedback** generation, when present, is mostly done as routine indices; although some programs have capabilities of generating on demand feedback. **Analysis** is performed as simple calculations (sum and average) in most of the instances, while there are a number of indices that need modest training to master, and a few who need statistical experts.

**Quality control:** All participants appreciate the possibility of data entry errors and deliberate manipulation of data in their systems, however, none of the programs seem to have implemented a mechanism to deal with the latter (e.g. log files, authenticated access, etc.) none of the participants accepted the possibility of measurement and sampling errors. Most participants stated that their software uses double entry or data analysis techniques to find possible errors. The SHEMIRANAT project, while admitting to the possibility of deliberate manipulation of data, believes the entry to 'have a high confidence coefficient' and doesn't mention any kind of quality control

**Workload:** most programs are designed to accept the entry of 10-200 input forms per month in at any level. Exception is again the SHEMIRANAT project, which expects 200-1000 forms a month to be filled. Nevertheless, all this work is to be done as a part of daily duties of the health care personnel, and no extra payment is believed necessary for this activity. There are different strategies for maintenance and upgrade of the software, including short term contracts with the private sector when need arises, employing new programmers, use of the available programmers, and long term contracts with private sector.

**Publication and distribution:** Most users expect their data to be published on paper, or distributed on the electronic media for certain people who have the privilege to access it. No one plans to distribute data on the Internet for public access or sell the data to interested parties.

**GIS:** While almost every participant believes in the value of GIS, this functionality is not implemented in any of the evaluated software. Some have the ability to export their data to external GIS programs.

**The End users** interviewed were responsible for data entry, analysis, report and feedback or a combination of these activities. They had all worked more than 6 months with their respective programs. Most had started with no knowledge of computers, although a few has experience with DOS and few with Windows operating systems. Most had learned the programs by themselves using trial and error methods, and has enjoyed help from someone who already knew how to use the program. None of them had found and read a book or article about the program to learn it. The learning process had taken them less than 6 months, some had learned their programs in less than a month.

**To educate new users** for working with the programs, most end users stated that they could teach one with a background in DOS in less than a month; although some required no background and others asked for familiarity with Windows and statistical methods and indicated that the learning would take 1-6 months.

**To find out how the end users use the program** a series of questions were asked with yes, no, and not applicable answers. All users stated that they don't need to memorize a code list for working with their programs, and that the program is capable of finding entry-time errors and warning them. All agreed that the program they use can backup the data easily, and is better than the paper based system, and seldom stops responding or needs a re-boot. Most said they could recruit other colleagues to help them with simultaneous data entry if the need arises, and could search the data for something they needed.

Most end users seemed not familiar with possibilities such as spell checking, work load assessment, interface customization, scripting, and help system. Most were satisfied with the process of bug report and speed of execution, were in contact with other users of the program, could easily install the program on a new computer, could import and export data, and stated that the program is enough for their work for the next few years.

More than half the interviewed users said they had had a data loss experience with their programs. None indicated that they enter data frequently as free text, but only half said that they used lists in the interface to select the values of data fields. All said their programs offered easy analysis of data, but none felt that they did as much analysis with their programs as data entry. More than half stated that they don't have to dedicate a computer to the program, but most needed a re-boot to switch to the operating system they use most for their other needs.

Less than half of the users felt the data they are entering should have been entered at the lower levels of the system. Also the believers that the program interface is an exact replica of a paper form were less than half.

Few, if any, of the users could take his or her work to home. Almost no one could ask a third party to add functionality to the program. No one could customize the reports of the program they used, and the people satisfied with the process of implementing new features were also in the minority.

For data analysis and reporting, most users employ the program's built-in functions. When this is not enough, some use hand-produced reports and few export the data to another application such as Microsoft Excel to accomplish the task.

**The Technical aspects** of the programs were to be assessed by a number of questions, too. However, the nature of this investigation was that the participants were not obliged to answer any question if they didn't want to, and the only projects in which the programmers chose to answer the questions were the DTARH and the Malaria projects. Regrettable as it was that other programmers chose not to fill the questionnaire; it doesn't

have a considerable impact on the overall outcome of the investigation. The goals and the expectations were to be extracted from the first and third parts of the questionnaire, and the second part was only there to select the best policy of maintenance and development and to recommend it to the programmers.

The DTARH project is a Microsoft Foxpro application developed with the code and fix procedure over a relational database. There is no data dictionary, and interaction with data is done using Foxpro Rushmore technology. The Persian dates are stored as free-text or 3-field collections, and the Persian text is stored using Sayeh standard of Sinasoft company. It has built-in functionality for query, report and analysis of data.

In DTARH, the coding system used to fill the data values is proprietary. There is user level security, backup is done on disk and with occasional schedule. The instances of DTARH can export and import data as Text formatted or DBF files, and DTARH is composed of more than two independent parts.

DTARH supports Networking with IPX/SPX protocol, has table locking, an average data traffic of more than 10 MBp/day. It can use 10-base T networks. Third party development for DTARH needs access to source code, which is partially documented. Technical support for DTARH is during office hours and through phone line.

The Malaria project is an IFA developed using EPI-info software from CDC. It is a small program to help in generating secondary reports from primary data. It stores date as 3-field collection, and Persian text in IranSystem format. It too was developed with code and fix process and has an uncommented source code, with no data dictionary. Interaction with the non-relational database is through Epi-info's proprietary techniques, it doesn't support open standards such as SQL. Coding system comes from DTARH, and additional codes were developed proprietary.

Maintenance and development of the Malaria project are done through changing the source code. There is no backup system, no network support, and no security. Analysis, query and reports are produced with the EPI-Info built-in functionality. EPI-Info has the capability to export data to other programs such as Microsoft Excel and Access, for further processing.

## **Discussion**

The main concerns expressed by managers during the meeting were:

- **Dependency on the original programmer:** Once software is developed, the managers don't want to depend on the original programmer to maintain and upgrade it. If the programmer leaves the system for any reason, the manager needs insurance that others can follow his or her work, so that the project is not lost. As the use of the software increases, so does the demands on the programmer for debugging and adding new features, and a task which used to be done by a single programmer now needs a team. It is unwise to employ software that doesn't meet these criteria as a critical nation wide application.

- **Dependency on a particular hardware or operating system:** as the software industry evolves, new solutions and possibilities emerge. The managers would like to make use of the new advances and don't want to get stuck with the old technology just because their software can't support the new one. The copyright issues for most software should be considered if the country decides to enforce the international copyright acts.
- **The need to know everything before hand:** Most programmers start their software design with a series of interviews. They then compile a proposal and start producing the software. The managers are often faced with a change in strategy or a new or an unforeseen requirement which the programmers refuse to implement because it was not stated in the original proposal.
- **The parallel tasks:** lack of an official responsive for the development of health information systems leads to every office developing its own software, most of them doing the same job, incompatible to share data amongst themselves, and there is no defined standard on which these software can be evaluated and selected.
- **The endless planning process:** when a manager has a problem for which he or she needs a software solution, and then finds out that another office is developing it, he or she has no idea or assurance that the software will actually reach the operational phase. In fact, most managers recall wasting precious time waiting for another office to develop a program, and at last starting on their own.
- **The financial support:** the budget for software projects comes from independent proposals. There is no continuous financial support for maintenance and upgrade of software. After a few years, managers can't find a funding source to continue supporting their applications. The available funding is too low to attract professional firms, and the manager must resort to less experienced, less expensive programmer who might as well turn out to have made mistakes in his design and development because of his lack of knowledge, provides lowest possible maintenance and customer support, and the final product has the weakest compatibility with existing software and standards.
- **The restart form scratch problem:** The health system managers have not received computer software development training. They often come face to face with a situation when a "new" expert announces that the old software was developed in error, and the whole process should be re-written from scratch. There is no guarantee that another expert will not show up to say that the second one was also wrong. Each re-design takes time, costs money, and must overcome issues such as client and programmer resistance.

The main concerns and problems of the programmers follow:

- **Obscure goals and needs:** most programmers complain that while the managers are keen for software solutions, they don't know what exactly they want. They tend to frequently change their views and requirements, so that it is impossible to reach a final point for the project. Most of the time, the programmer is invited too early, before enough planning is done by the managers.
- **Unavailable 'big picture':** while all programmers know they have to write their code to support a 'big picture' in which different programs can interact with each other, they find out that there is no such thing in existence. There is no standard

on which the ministry agrees and requires the programmers to follow. In fact, the managers mostly don't even know what such a standard is. As a result, the programmer has to select a standard, only to find out later that others have chosen other standards that are incompatible with his.

- **Insecure payment methods:** the overwhelming bureaucracy that exists in imbursements results in the fact that the programmer is not sure whether he will actually receive the payment he is promised. There have been countless examples of programmers failing to get their payment from the ministry. As a result, programmers are not ready to invest their time and energy fully in a ministry project.
- **Inconsistent management strategies:** the unavailability of a consistent strategic plan for departments results in abrupt changes in strategy when the manager of the department is changed, which occurs quite frequently. Often the new manager disagrees completely with his predecessor and would not follow his promises on payments and support, resulting in the project left half finished and the programmers unpaid.
- **Uncertain 'end points':** when a program is finished, the new possibilities that emerge result in immediate surfacing of new needs. This in turn ends in the manager asking for new things before the last one is finished. The process can result in a half finished project in which the programmer thinks he has done more than he should, while the manager believes he has not got his money's worth.
- **No official place in the health system:** there is no official place for a software programmer in most departments, so they have to employ the programmer in other posts, which is very difficult is at all possible. Even when this is done, the salary of such posts can't compete with the salary of a programmer in the private sector even remotely.

The main problems of the end users consist of:

- **Their opinion is seldom incorporated in the software design:** most software is developed by interviews between the programmers and the managers. The end users, who are the ministry employees, are seldom if ever asked what they need most. This ends in software that leaves the most time consuming issues out and focuses on simple tasks, which are often easier if done on paper.
- **Their extra effort is not appreciated:** learning and mastering new software, many of which don't include a manual or a help system, is a time consuming and tiring process. However, there is often no benefit involved for a ministry employee to put him or herself through this trouble. Soon they find out that by learning new software they just increase the expectations of their superiors, while by pretending not to know how to operate a computer they would be left alone.
- **They have to do the same task over and over:** the unavailability of a reference point to organize and set the priorities of software development has resulted in many parallel activities, and most employees find themselves re-entering the same data into multiple incompatible programs. There are instances, even when the paper output of one program has to be typed back in as the input to another one.
- **They don't know what they can ask and how to ask for it:** when a user encounters a problem, needs a feature added, or a bug fixed, there is seldom any standard path foreseen for him. Most use their personal friendship with key

people to get the job done. There is no mechanism for ensuring of customer satisfaction, the answers a user gets in this situation, mainly “not possible”, “not feasible”, and “under construction” are not verified or followed.

The prominent technical problems of software in use

- **The Persian Text:** There have been a large number of standards for Persian language since the start of computer use in Iran. No official place has developed an open, well supported standard that would address all the specific properties of the Persian language. As a result, each programmer would use a way most achievable to him. Some of these standards include: IranSystem, the simplest one with the most public domain utilities, used in many systems where a fast and simple solution was needed; Sina, a commercial standard that became popular because of its success in changing MS windows to support Persian, Sayeh, which is used by DTARH and has a hardware lock, and many others, composing a list of about 30 standards. The direct result of this heterogeneity is that many of these programs, while similar in interface and appearance are not able to share data directly because of the different languages they use. The technical differences between the standards are too detailed to be discussed here, and just the fact that no one has yet been able to produce a complete converter for these standards is mentioned to show that it is not always easy to convert a standard to another one.
- **The Persian Date:** another source of incompatibility between software is the Persian date, a scientific calendar founded by Omar Khayam, the ancient Iranian scientist. While the database engines are designed to store the date as Julian (seconds past a certain date) or Gregorian, the Iranian nation uses this Shamsi calendar. Many programs go down in flames when it comes to managing Persian date. Some just resort to the worst, entering 1981/2/31 (which is invalid) for 1381/2/31. Some store the month, day and year in different fields, and some do a simple conversion to store the date as Julian and show it as Persian. Any sharing of data should consider this issue carefully.
- **The database design:** flaws in the database design are commonly the latest to show themselves and the most damaging in a program. While the user interface is readily apparent in a program and could be evaluated at the time of the commissioning of the new software, the database design errors only surface after the program is put under load, which means that it has been in use for some time, has consumed some amount of energy and resources, and is holding valuable data. Most managers and end users surf around the interface when facing new software, unaware about the technical qualifications a database should have. After considerable amount of time and energy is invested in the program, they get stuck in the limitations posed by non-standard database design. At such a stage, exit barriers for leaving the program and rewriting another one are so high that most decide just to accept the shortages and put up with what they have.
- **Communication and networking:** communication between various programs acting in different sectors of a system is an important need once a program becomes popular. Unfortunately, there are few standards worldwide that are accepted as general purpose solutions for data transfer between different programs. These tend to be quite new and somewhat complex, and expertise in their application is rather scarce. Engines to translate data to these standards and

back to the form that the program would understand are needed for such functionality, but need to be developed or at least modified specially for a particular program. As for networking, apart from simple file sharing provided by the operating system, most programmers rely on the built-in capabilities of the software they use for issues such as collision detection and multi-user environment setup, which results in even more inconsistency across the programs.

- **Maintenance and upgrade:** the direct result of hiring programmers on a project basis is that when they are finished, they will move on to other projects and can not be found when after a while there is need for maintenance and upgrade of the program. Since the programmer is not hired by the institution, and since there is no standard document for the programs, another programmer can't continue the work of the first. Even when the original programmer is available, the non-standard software development process and the 'code and fix' protocol commonly employed in producing such programs inevitably leads to a patchy, unstructured code that is even impossible to maintain by the original developer.

The technical solutions

- **The Unicode standards:** special properties and rules of the writing in Persian, which are similar to Arabic and Hebrew, are addressed by Unicode UTF-8 and Windows-1256 standards. Converting text in existing databases to these codes and redesigning the interfaces to utilize them is feasible since it results in consistency of text management and can benefit from the existing tools present commercially or in public domain. Moreover, it is a sure step toward ensuring that the program can be managed by third parties.
- **The Date conversion:** Procedures converting Shamsi date to Georgian and back, accounting for the 5<sup>th</sup> leap phenomenon which occurred in 1375 (1996) in shamsi date and 2000 in Georgian date, and would repeat every 30 years, can be developed and distributed as objects which support common OOP component models, such as ActiveX and JavaBeans, for present and future programmers to incorporate in their projects. The components can even be placed on the Internet for easy upgrade and distribution.
- **The Data Dictionary:** every database should have a data dictionary accompanying it, which is a standard document that describes the rule and logic of every row in the tables of the database, along with the views, triggers, stored procedures, and other objects; has scripts to build them, save and restore them, and delete them; documents the relations between tables, and describes the roles and categories of users. The presence of such a document ensures that the database design can be reviewed and edited early in the implementation process, the program can be maintained by other programmers, and third party components can be developed to use the data of the program.
- **The object oriented programming:** The accepted solution to the "software crisis" is the object oriented programming, or OOP. This philosophy consists of breaking down the software into small 'objects' that poses three fundamental properties, i.e. encapsulation, inheritance, and polymorphism. While details of the technology are out of the scope of this report, it can be said that these small pieces of software are debugged and tested separately, and can be replaced or reused if needed. Maintenance, upgrades, changes and debug of OOP based software is far

easier than the software based on older technologies, since the various parts don't assume anything about each other except the properties and behavior they have declared public. Also, they don't concern themselves about how a task is done, just what is done. Different people can be employed to complete or maintain a project, and when an object is malfunctioning and is beyond repair, it can be replaced without affecting the other parts of the system.

- **The multi-tier application development:** The newer technique of application development, which has evolved from the client/server technology, is to divide the software to multiple "tiers"; for example the database, the application logic, and the user interface. Each tier then communicates with other tiers independently. In contrast to the old approach, where these tiers were developed simultaneously in a single compartment, the new method has the advantage that each layer of the program can be managed or changed separately. For example, the database can be placed on an advanced machine and the user interface on ordinary office computers, thereby decreasing the cost and resources needed for maintenance activities. Also, if a new and better database engine is shipped to the market, it can be deployed in the database tier without affecting the other layers of the program. This approach greatly facilitates networking and communication, since it contains these properties inherently.
- **The component based standards:** the next step to the OOP technology is developing objects that respect an agreed-upon standard, so that other objects can communicate with them without knowing their internal structure. ActiveX and JavaBeans are examples of these standards. Contrast to simple objects in OOP languages, two components need not be written in the same language, and the source of a component is not needed for a programmer who wants to use it in his own project. For example, a programmer can develop an ActiveX in C++ language and then give it to another programmer to use it in a Visual Basic application. ActiveX objects can also be used in popular applications such as MS Excel or even MS Internet Explorer. This way, when a feature is needed in a program, it can be commissioned to be implemented as an ActiveX and be added readily to the application. Newer versions of ActiveX or JavaBeans components can be deployed on the web to upgrade applications seamlessly, therefore cutting the costs of upgrade. Imagine, for example, that a program is deployed in all the health centers of the country. If a newer version is produced, it should be sent to every node and installed there, which costs a lot of time and money. If, however, the software is developed using a component based technology, the only requirement would be to deploy the new version on the central website and each instance would upgrade itself automatically once it connects to the website.
- **The software development process:** as a software program evolves and becomes more essential in management and decision making, it becomes critical to have a thorough understanding of the processes and the solutions implemented inside it. In order to facilitate analysis and plan the future development of the software, the philosophy of software development has gained increasing acceptance all over the world. One of the techniques of this philosophy is the Unified Modeling Language (UML), which consists of making a 'blueprint' of the software as it is developed, this blueprint can then be fed to a number of tools that would analyze

the workload of each object and determine the efficacy of breaking a busy component to multiple components or of merging small components to a large one. The changes can then use reverse engineering to convert this blueprint to code and vice versa. For a nationwide program, which has to be consistently checked for optimum performance and efficacy, such a blueprint is an absolute necessity.

## **Conclusion**

### **The SWOT analysis**

- **Strengths**
  - The required commitment exists upon the health managers
  - The required technology and knowledge exists in the country
  - The users are acquainted with the nature of computer applications
  - The old ways have proven ineffective and the new ways are welcome
  - Widely accepted standards exist for common problems (e.g. text, date)
  - Proven solutions exist for most pressing problems (e.g. OOP, UML)
  - Proven paper systems exist that can act as templates for programs
- **Weaknesses**
  - No generalized strategic schema showing what must be done by whom
  - No recognized body exists for setting standards that all will follow
  - No organized protocol for software commissioning and contract
  - No official place for software programmers in the system
  - No standard documentation protocol for programs
  - No verification and auditing reference point for quality control
  - No plan or process for performance monitoring
  - No official user support system
- **Opportunities**
  - Least client resistance is anticipated
  - The original programmers would cooperate for changes
  - Sufficient funding can be provided for the project
  - Most of the data can be salvaged from the old programs
  - Sufficient time is at hand for correcting the current programs; they can remain operational until the final solution is ready.
  - Capacity can be built into the system for future development using the current experience.
  - The private sector is available for help provided that clear rules are set to govern the cooperation
- **Threats**
  - No clear definition of general terms (such as MIS)
  - No guarantee that the existing software can tolerate increased workload
  - No communication standard between existing programs
  - No one but the original programmer can document, maintain, and upgrade the existing programs
  - The existing software is bound by the old hardware and technology and can't utilize the new advances.
  - Deployment of the new version of software takes time and costs high.

- End user opinion is not valued enough, which both discourages the end user and deprives the system of the chance to see its weaknesses and act on them.

### **The categorized issues**

- **Category 1, issues on which the whole group agrees:**
  - The existing software has cost considerable amount of time, money, energy and resources and should be saved or salvaged to the maximum extend possible
  - A high priority is to ensure that the system would not depend on one person, group of people, or company for maintenance and upgrade of critical software
  - A fast and efficient user support system should be established to receive and act on bug reports and feature requests, and to act as a forum for all the end users and managers to exchange ideas about the information system.
  - A protocol for reviewing and evaluating new advances in software and hardware technology is needed to introduce new possibilities and functionalities to the managers and end users
  - Any effort to upgrade existing software must be proceeded with a thorough feasibility study documenting the expected gain of such upgrade and checking for cost-effectiveness of the operation
  - Protocols for realistic maintenance and upgrade should be incorporated in all existing software since it is inevitable that change would become necessary in the future
  - Official and consistent methods of payment and employment should be sought to act as incentives for the programmers
  - Existing software should be documented as soon as possible to aid in staging required next moves and to facilitate employment of third party programmers to help maintain and upgrade them
  - A review board is needed to evaluate and comment on the existing and new technologies, to help users and managers select the appropriate solutions for their problems away from the commercial advertisement hoax.
  - A strategic plan is needed to outline the priorities and information needs of each office in the ministry of health, so that these plans can be compared to each other and form the basic structure of the MIS schema for the ministry.
  - Every effort should be made to avoid monopoly in the health software development, distributing fair chance for competition across all who wish to contribute.
  - Effective presentation of capabilities of produced software prevents parallel works.
- **Category 2, issues for which additional information is needed:**
  - The feasibility of upgrading the existing systems
  - The priorities requiring new software development activities

- The data items of each level of the health system
- The workload, or traffic, of each node in the system
- The optimum way of educating new users of the software
- **Category 3, issues that the group disqualifies itself from answering:**
  - The best software development process
  - The detailed list of needs and requirements of health information system
  - The best software or hardware technologies to employ
  - The best solution for payment and financial problems

#### **The Problem, cause, solution, and prevention**

- **Problem: Software depends on the original program for development and maintenance**
  - **Cause:** No documentation or design exists for the program
  - **Solution:** ask the programmer to document the program and produce a data dictionary and at least sequence diagrams or algorithms so that other programmers can use it for maintenance and upgrade.
  - **Prevention:** prepare a guideline for the managers to know how they shall order, commission and accept a program to ensure there is enough documentation and capability of later delegation of maintenance and upgrade to other programmers
- **Problem: Software depends on a particular hardware or operating system**
  - **Cause:** dependency on vendor specific solutions
  - **Solution:** use the documentation to produce software blueprints and use reverse engineering to rid the program from dependency on hardware or operating system.
  - **Prevention:** instruct the programmers to use open standards and use software development technologies that allow reverse engineering (e.g. UML).
- **Problem: All of the requirements are not known before hand, and new needs might arise that the program won't be able to address.**
  - **Cause:** the program is developed as a single entity, not making allowance for future changes or reusability
  - **Solution:** use the documentation to make the program object-oriented, so that future changes would be possible with minimum infliction on other parts of the program
  - **Prevention:** instruct the programmers to construct the software using object-oriented technologies, so that adding new features or changing the logical rules would require minimum energy and resources in the future
- **Problem: The financial resources are low and experienced firms can't be employed for developing the software, resulting in low quality software**
  - **Cause:** Management is trying to "buy the solution off the shelf"
  - **Solution:** after using the documentation to make the program object oriented, turn the objects to components (ActiveX or JavaBeans), so that future development can be accomplished with small steps. In other words, once the program is component based, the required functionality is broken down to multiple components, which are produced by a series of short time, low budget contracts.

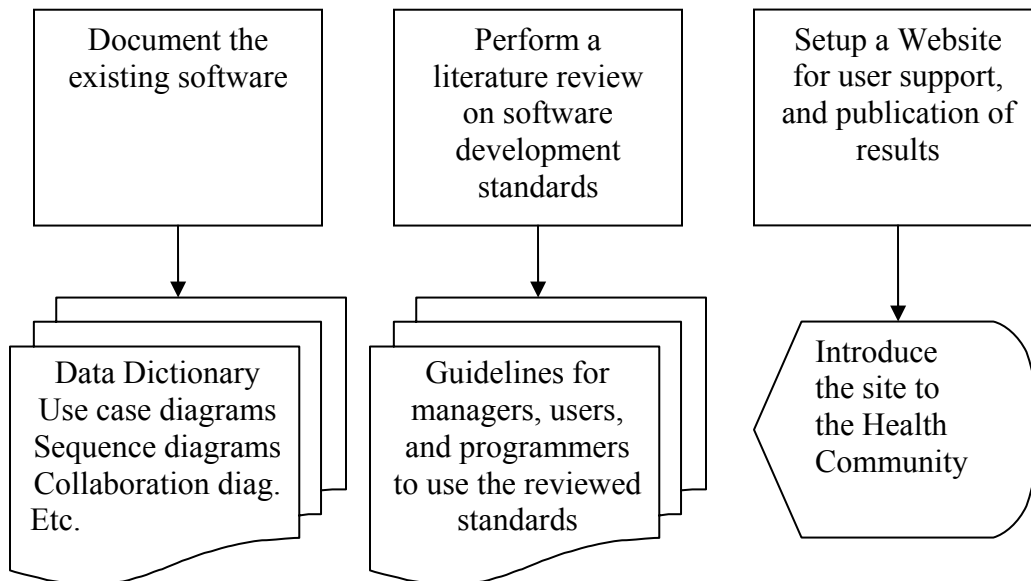
- **Prevention:** instruct the managers to avoid off the shelf software contracts. Advise at least three separate phases for a software project: design, implementation and deployment, and maintenance and upgrade. The programmer should produce the software with the proviso that another group should be able to undertake the next step.
- **Problem: The format of text and date are inconsistent between applications, making data sharing and third party component development difficult**
  - **Cause:** use of proprietary or old standards for text and date management
  - **Solution:** ask the programmer to develop a converter to change the text format to UTF-8 or windows-1256 standards, and to change the date to Julian format.
  - **Prevention:** commission small projects to develop ActiveX and JavaBeans components for text and date manipulation in the aforementioned standards.
- **Problem: The software is hard to deploy on network, costly to administer in remote areas, and can't run alongside other programs**
  - **Cause:** There is only one tier in the application, or it uses 'Fat client' strategy.
  - **Solution:** use the documentation to divide the program into at least three tiers of database, application logic, and user interface. This way, the database can be deployed on a computer where administration is available, while user interface can run on environment computers where administration is not available.
  - **Prevention:** encourage "thin client" strategy, where the software deployed on client computers is small and simple (e.g. browsers) while the server software bears the maximum complication and workload. This is much the same way the World Wide Web works.
- **Problem: There is little user support**
  - **Cause:** There is no official protocol for users to express and exchange their ideas and contribute to software development, since the development process is 'closed'.
  - **Solution:** after preparing the documentation and making the program object oriented and component based, set up a web based forum and publish a summary of documentation on it, hence changing the development process to 'open'. The users can use this forum to express their ideas and even write new components for the software, which can be incorporated after testing. This website can also be used as a base for automatic re-versioning and update of components.
  - **Prevention:** place call for applications on this website when new software is being developed, attract recommendation and advice from the user community, and prevent parallel production of a solution that already exists.
- **Problem: there is no clear protocol as to which standard to follow**
  - **Cause:** The existing standards have not been reviewed thoroughly
  - **Solution:** perform a thorough literature review on the current standards of software development, communication, networking, database development

and message management, deploy the results on the website and use it as a guideline for future development and a base for interaction between applications that use different standards.

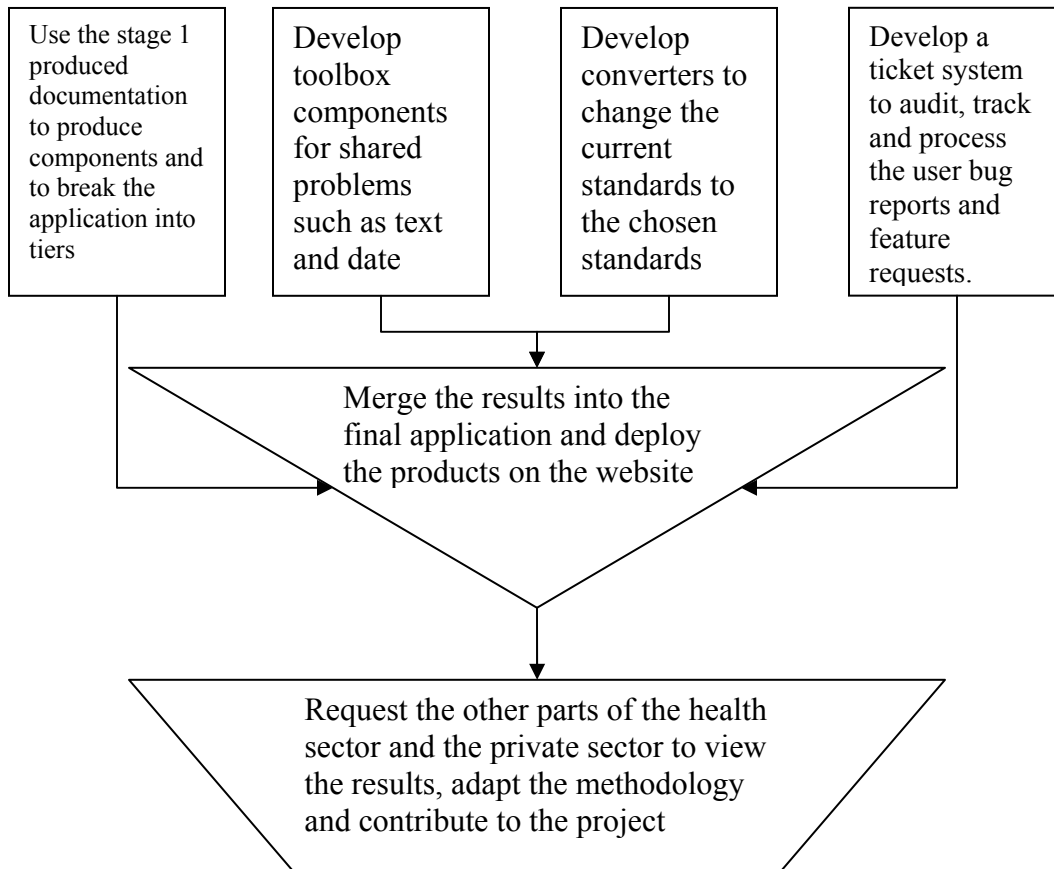
- **Prevention:** invite vendors and programmers to discuss their preferences and selection of solutions on the website, hence allowing for the import of the newer technologies as they evolve. Such discussions and articles would then lead to the availability of enough knowledge to convert older technologies to newer ones.
- **Problem: There is no ‘Big picture’ for the programmers to follow**
  - **Cause:** There is no such thing as a “Big Picture”; i.e. a plan that shows each and every process in the system, each and every data item produced and worked with, each and every need and requirement that has risen in the past, present, and future. Even if developing such a plan was possible, it would take so much time that by the time it was finished, all of the system would have changed. Moreover, there is no guarantee that all the issues will be counted for in any kind of plan.
  - **Solution:** take a well known problem, solve it with software which is component based, has enough documentation and follows a known and well established standard; and deploy the Use Case diagrams on the website. Produce documents to help users express their needs in a common way the can be understood and implemented efficiently. Encourage managers and users to construct their own use cases. Revise and merge these plans to get what can serve the purposes of a big picture.

## Plan of Action

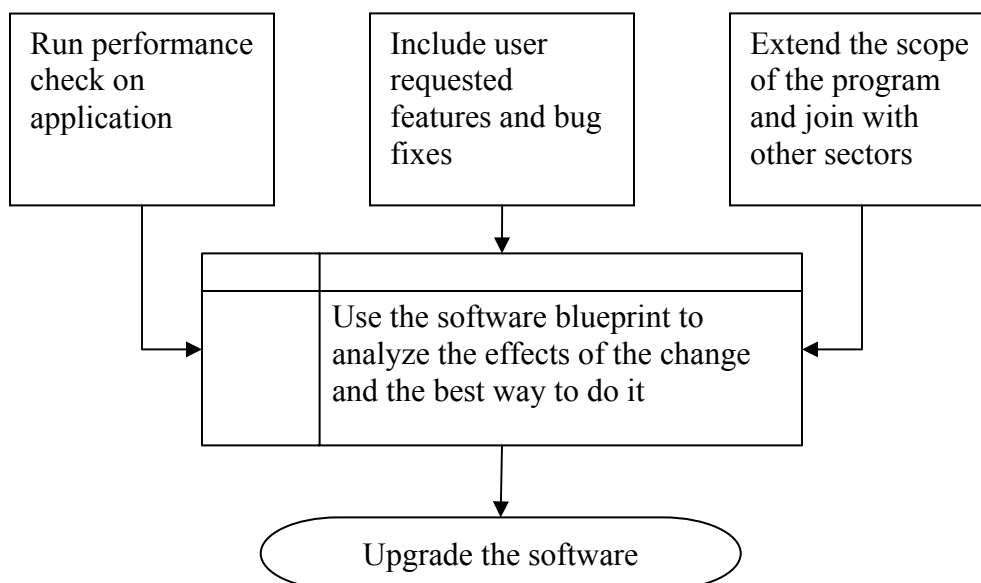
### Stage one: prepare the workspace



## Stage Two: implement the applications



## Stage three: maintenance and upgrade



## References

- Booch G, Jacobson I, Rumbaugh J (1998). The unified modeling language user guide. Addison-Wesley Pub Co; ISBN: 0201571684
- Dean A (1997). Epi-info user manual, CDC publications. [www.cdc.gov/epiinfo](http://www.cdc.gov/epiinfo)
- Hartshorne JE, Carestens IL (1990). Role of information systems in public health services. Journal of Dental Association of South Africa, 45: 313-317
- Sandiford P, Annett H, Cibulski R (1992). What can information systems do for primary health care? An international perspective. Social science and medicine, 34: 1077-1087
- Stroustrup B (1982). The design and evolution of C++, Addison-Wesley Pub Co; ISBN: 0201543303
- Taylor CE (1984) The uses of health systems research. Geneva, WHO public health papers, No. 78
- WHO staff (2001). Design and implementation of health information systems, World Health Organization; ISBN: 9241561998
-