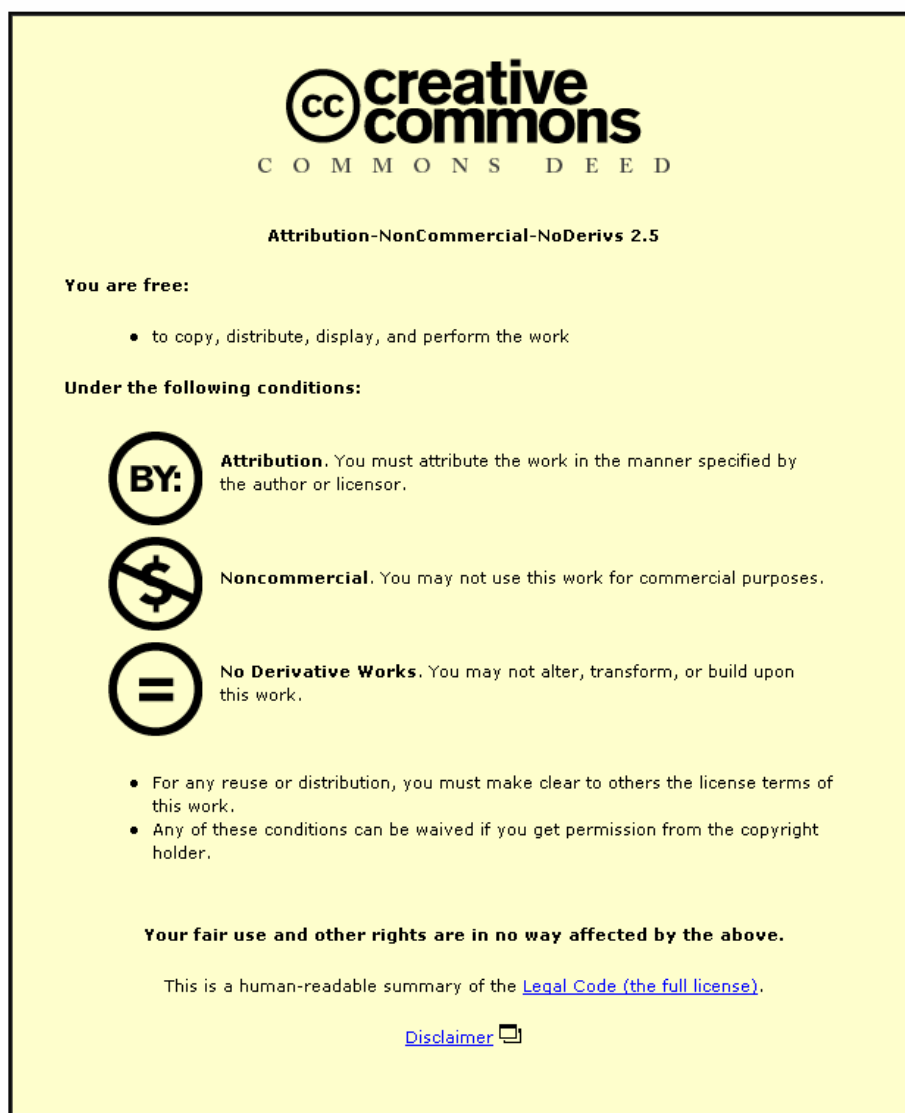




This item was submitted to Loughborough's Institutional Repository by the author and is made available under the following Creative Commons Licence conditions.



For the full text of this licence, please go to:
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

Genetic algorithm based software integration with minimum software risk

L Yang^{1*}, B F Jones², and S H Yang¹

¹Computer Science Department, Loughborough University, Loughborough, LE11 3TU, UK

²Applied Computing Department, University of Derby, Derby, DE22 1GB, UK

Abstract. This paper investigates an approach of integrating software with a minimum risk using Genetic Algorithms (GA). The problem was initially proposed by the need of sharing common software components among various departments within a same organization. Two significant contributions have been made in this study: (1) an assimilation exchange based software integration approach is proposed; (2) the software integration problem is formulated as a search problem and solved by using a GA. A case study is based on an on-going software integration project carried out in the Derbyshire Fire Rescue Service, and used to illustrate the application of the approach.

Keywords: software integration; software risk; genetic algorithm; fire risk management.

* Corresponding author.

Email address: L.Yang@Lboro.ac.uk (Dr L. Yang). The short version of this paper was presented in the International Conference on IEA/AIES 2004 in Canada.

1. Introduction

Any large organization encompasses many islands of information and automation. To decrease costs and increase productivity, these large organizations need to eliminate silos of information and automation through software application integration. There are various ways to categorize the software application integration. Twenty years ago Miller [1] categorised software integration as database integration, operating environment integration, and user interface integration. Twenty years later, with the support of the web and Internet technologies, Johnson [2] categorized software integration capabilities into data integration, process integration and content integration. Data integration lets applications share and exchange relevant data and transactions. The goal of data integration is to deliver a single virtual database through data-oriented application integration. Process integration lets applications and people initiate the consume events and participate in various business processes that span multiple applications or organizations. Process integration can increase the efficiency in executing the process by increasing the level of process automation internally and externally. Content integration lets application users access, aggregate, deliver, and exchange all relevant content, regardless of how the content is managed or where the content resides. The cornerstone of content integration is providing a single point of access to relevant content, no matter where it's managed. These three types of integration are complementary and greater than the sum of their parts. Neglecting any one type of integration will cause the enterprise or organization not to realize its goals of decreased costs, increased efficiency, and increased revenue. There are other categorizations of software application integration – such as Enterprise Application Integration (EAI) [3], Business-to-Business (B2B) integration [4], and Application-to-Application (A2A) integration [5, 6].

This work was initially proposed by the need of sharing common software components among various departments within a same organization in order to reduce IT infrastructure costs. In our previous work [7, 8] a generic architecture of an Assimilation Exchange (AX) has been proposed and this can be used to underpin strategies for enterprise integration in EAI, and knowledge and complexity management in B2B. Such an exchange is constructed through the assimilation of parts of information infrastructure and business processes from various organizations. Integration of a partner IT infrastructure and technology into an AX can mean significant cost savings in terms of reduced IT infrastructure costs and overheads since resources and systems can be assimilated into the exchange and shared with other partners, thus introducing a shared cost saving. One of the key issues in the design of the AX is how to choose the participating software components to meet the specified functionality and achieve a minimum risk as well. This article will extend the AX into a generic

software integration approach and give a systematic method for establishing the AX. The design of the AX is transferred to an optimal search problem with the objectives of both minimizing the software risk and achieving the specified functionality.

Software risk is a measure of the likelihood and loss of an unsatisfactory outcome affecting the software product. It is hard to precisely estimate the software risk. The Software Engineering Institute (SEI) deals with software risk on a high/medium/low (H/M/L) basis. Some researchers [9, 10] used quantitative factors such as complexity factors, connection factors, and severity indices for the estimation of software risk. The more complex the software is, the higher the probability that it will have faults. The complexity of the software is composed of component complexity and connection complexity. The former describes the complexity of the behaviour of individual components, which can be specified using state-charts that define the component's states and how it responds to external stimuli according to its state. The latter describes the interaction between components, which can be specified using sequence diagrams, and defined as the sequence of interactions between components in a timely ordered manner. The complexity of a component is not a sufficient measure for assessing the risk associated with its failure. Those components in the software system, which require special development resources due to the severity and/or criticality of their failures, must be taken into consideration. The Failure Mode and Effect Analysis (FMEA) technique is a systematic approach that details all possible failure modes and identifies their resulting effect on the software system [11].

The rest paper is organized as follows. Section 2 reviews two generations of software integration. The weaknesses of them have been highlighted. Section 3 describes the AX based software integration approach. Section 4 reformulates the software integration problem as a search problem. Software component risk assessment is presented in Section 5. Section 6 is the application of the AX based software integration approach to the Derbyshire Fire Rescue Service. This case study was based on an on-going software integration project. Section 7 concludes the paper and discusses future extensions.

2. Two generations of software application integration

There are various approaches for achieving software application integration [2-6]. These approaches can be summarized as two generations. The first generation involves establishing a basic data interchange infrastructure

between each pair of applications. It has been known as the point-to-point integration. Individual applications are loosely coupled, permitting a degree of application independence. The shortcoming of the point-to-point integration is that the number of interfaces required grows exponentially. With n applications, $n(n-1)$ interfaces may be required since each application may need an interface with every other application. The impact of minor changes in communication requirements and that of adding a new application is significant. Maintenance is clearly a nightmare. Two applications, running on the same computer, written to the same standard can communicate easily. Unfortunately, most point-to-point integration doesn't fit this perfect world. Therefore such integration is impractical for a large-scale enterprise and organization.

The second generation of software application integration involves a mediator to establish a common communication channel for all individual applications. A particular approach in this generation is message bus based integration [6, 12], which requires interfacing each application to the message bus through an adapter. Each application has only one programmatic interface, the message bus. Applications communicate by publishing a message to the bus, which delivers the message to those who require. However, this solution may not be logically much different from the point-to-point solution if every application needs a different messaging interface with every other application. Another more serious problem is that the information flow is embedded in the application and is practically impossible to be viewed at a high level of abstraction. Changes in the flow will require changes in application logic.

3. AX based software integration

It has been recognized by large organizations that many duplicated functions and IT systems have been implemented and used in their IT infrastructure. With the widespread availability of Internet technologies, there is considerable potential for new forms of software application integration to reduce the duplicated IT systems and eliminate the silos of information. One new form of integration is the AX proposed in our previous work [7, 8]. An AX is an advanced exchange that assimilates parts of existing systems from various applications and uses them to create a value-adding infrastructure, shared by the partners. Each participant contributes to the AX a number of components of their information infrastructure, which are assimilated, shared and inter-operated within the AX. In doing so, the AX may offer the realisation of a shared system thus reducing costs, risks and overheads through the sharing of common components and information infrastructure. The concept of the AX is illustrated in Fig. 1. The AX can be constructed through the *assimilation* of parts of existing applications from various

organisations. As a simple case, the AX may present a hub for common communication infrastructure between partnering applications. More complex forms will involve sophisticated forms of information infrastructure. The infrastructure that is assimilated into an exchange may represent significant parts of an organization, even entire departments, and therefore gives rise to a form of synchronised outsourcing as several organisations ‘synchronise’ their on-line cooperation.

In general, most applications can be separated into public, private and back-end part, as shown in Fig. 2. The public parts should be typically developed by detailed agreement between various partners, or have been codified by a standards body or consortium. The private parts communicate with the public parts and are not available to the extended applications. Furthermore, to truly support the public parts, back-end internal parts also need to be seamlessly integrated into the private parts through the application integration. The AX in the logic centre acts as a coordinator and an information post office for all public parts contributed by various partners. Therefore, external IT infrastructures are available for AX participators. All communications between participating applications take place through the AX so that all applications can collaborate and synchronize together to achieve the defined functionality.

From software engineering perspective, the AX is an integration of a number of software components offered by a number of participants to achieve a defined goal with a minimum risk. The challenge is how to choose these components from participants to achieve the defined functionality and minimize the risk at the same time. This challenge is explored in the next section by reformulating the problem as a search problem.

4. Reformulating software integration as a search problem

Harman and Jones [13] proposed search-based software engineering in 2001, which mainly focused on software testing and test data selection [14]. To the authors’ knowledge, meta-heuristic search-based approach has never been used in software integration before. The principal intention of this section is to demonstrate that the reformulation of software integration as a search problem is conceptually feasible.

In general, in order to present any problem as a meta-heuristic search problem, it is necessary to define:

- A representation of the problem which is amenable to symbolic manipulation,

- A fitness function defined in terms of this representation, and
- A set of manipulation operators.

The representation of a candidate solution is critical to shaping the nature of the search problem. Floating-point numbers and binary code are representations, which are frequently used in existing applications. The fitness function is the characterization of what is considered to be a good solution. Generally it will be sufficient to know which of two candidate solutions is better according to the fitness function. Different search techniques use different operators. As a minimum requirement, it will be necessary to mutate an individual representation of a candidate solution. Genetic algorithms include three operators: mutation, crossover and reproduction. These issues are covered in detail in the general literature on meta-heuristic search [15, 16].

4.1 Mathematical model of the AX

As described in Sections 3, the AX is an integration of a number of software components offered by a number of participants. The selected AX components should achieve a defined functionality and minimise the risk as well. In most cases it may be hard to define functionality as numerical values. The symbolic description and the set theory are used in this study for modeling of the functionality of the AX. Obviously, the overall required functionality, F , of the AX is the joint set of all the required sub-functionality, F_k , as shown in Equation 1, and must be provided by the aggregated functionality, f_{ij} , of all the selected components m_{ij} , as shown in Equation 2. $Occur_{ij}$ is a binary value, representing the occurrence of a component m_{ij} in the AX. K is the number of the sub-functionalities. M is the number of participants in the AX. N^i is the number of the components of the participant i ($i=1, \dots, M$).

$$F = \bigcup_{k=1}^K F_k \quad (1)$$

$$F = \bigcup_{i=1}^M \bigcup_{j=1}^{N^i} f_{ij} \times Occur_{ij} \quad (2)$$

where

$$Occur_{ij} = \begin{cases} 1 & m_{ij} \text{ present} \quad (Selected) \\ 0 & m_{ij} \text{ absent} \quad (notSelected) \end{cases} \quad (3)$$

The functionality of the component, f_{ij} , can be the same as one of the sub-functionalities denoted by the symbol '=', or include more than one sub-functionalities denoted by the symbol '⊃', or be excluded from the overall required functionality, F , denoted by the symbol '∩'. They are described in the following equations:

$$\begin{cases} f_{ij} = F_k \\ f_{ij} \supset F_k \\ f_{ij} \cap F_k = \emptyset \end{cases} \quad (4)$$

where \emptyset is the empty set.

Similarly, the overall risk, R , is contributed by all the selected components m_{ij} , and is represented in Equation 5.

$$R = \sum_{i=1}^M \sum_{j=1}^{N^i} R_{ij} \times Occur_{ij} \quad (5)$$

where R_{ij} is the risk associated with the component m_{ij} , which is composed of a number of associated metrics or parameters, p_{ij}^l ($l=1, 2, \dots, L$), such as reliability, coupling, security, complexity, and so on. In principle, R_{ij} is described as Equation 6.

$$R_{ij} = R_{ij}(p_{ij}^1, \dots, p_{ij}^l, \dots, p_{ij}^L) \quad (6)$$

Equations 1 to 6 form the mathematical model of the AX based software integration problem. This problem can be formally described as: *choose the values of $Occur_{ij}$ in Equations 2 and 5 for the components m_{ij} contributed by the participants $\{P_1, P_2, \dots, P_M\}$ so that the overall risk R described in Equation 5 achieves its minimum value and the overall functionality F described in Equation 1 fit Equation 2.*

4.2 Reformulating software integration as a GA problem

The above problem can be solved using genetic algorithms by applying Equation 2 as the constraint and using the overall risk R as the fitness function. First all, a representation of the problem, i.e. a chromosome, is required. The chromosome can be expressed as a binary string, which is the set of the occurrences, $Occur_{ij}$, of all the possible components m_{ij} .

$$Occur_{11} \quad Occur_{12} \quad \dots \quad Occur_{1N^1} \quad \dots \quad Occur_{M1} \quad Occur_{M2} \quad \dots \quad Occur_{MN^M} \quad (7)$$

The string length is equal to the total number of the components m_{ij} , i.e. $\sum_{i=1}^M N^i$. A population is composed of a number of chromosomes. The initial value of the population is formed randomly and then subjected to reproduction, mutation, and crossover to form the next population, which should contain better possibilities to minimize the fitness function. The overall risk representation R is chosen as the fitness function here.

$$f_{fitness} = \sum_{i=1}^M \sum_{j=1}^{N^i} R_{ij} \times Occur_{ij} \quad (8)$$

In order to satisfy the constraint shown in Equation 2, a penalty term is added into the fitness function, as shown in Equation 9. This penalty term is represented by the number of missing sub-functionalities E .

$$f_{fitness} = \sum_{i=1}^M \sum_{j=1}^{N^i} R_{ij} \times Occur_{ij} + E \quad (9)$$

The number of missing sub-functionalities, E , can be found by comparing the joint set of F_k ($k=1, 2, \dots, K$) and the joint set of f_{ij} , i.e.

$$\bigcup_{k=1}^K F_k - \bigcup_{i=1}^M \bigcup_{j=1}^{N^i} f_{ij} \times Occur_{ij} \quad (10)$$

where K is the total number of sub-functionalities, which is obtained from the decomposition of the AX overall functionality. If all sub-functionalities have been implemented E will be equal to zero and the fitness function, $f_{fitness}$, becomes the value of the risk.

5. Software component risk assessments

According to the NASA Technical Standard [17], risk is a function of the possible frequency of occurrence of an undesired event, the potential severity of resulting consequences, and the uncertainties associated with the frequencies and severity. In the most risk assessment risk is defined as the multiplication of two factors: frequency (or possibility) of malfunctioning (failure) and the consequence of malfunctioning (severity), as shown in Equation 11. In large hierarchical systems, a system is composed of several subsystems, which in turn, are composed of components. The system risk is an aggregate of individual component risk factors [10, 18].

$$\text{Risk} = \text{frequency} \times \text{severity} \quad (11)$$

5.1 Frequency of failure

The frequency of failure depends on the probability of existence of a fault combined with the possibility of exercising that fault. For the sake of the simplicity, in this study, the frequency of failure is estimated by multiplying the frequency of failure for each line code with the number of lines of code (LOC). For example, assuming that the frequency of failure for each line code is 10^{-5} per year, the frequency of failure for a 1000 lines code component is computed to be about 0.01 per year. This estimating method was adopted by other researchers as well [10]. For the companies that are using shrink-wrapped software in which the LOC is not known, other proper risk analysis method is required.

5.2 Severity analysis

Severity analysis is a procedure by which each potential failure mode is ranked according to the consequences of that failure mode. Severity considers the worst-case consequences of a failure determined by the degree of injury, property damage, system damage, and mission loss that could ultimately occur. FMEA is suitable for severity analysis. When analyzing failure modes, first, the analyst identifies failure modes of components, then studies the effect of these failures, and finally ranks the severity of each failure, and identifies the worst-case effect on the system. The domain expert determines a severity for the faulty component for each scenario by comparing the faulty result with the normal operation. Severity classification recommended by MIL_STD_1629A [11] is used in this study and illustrated in Table 1.

6. Case study

An on-going project in the Derbyshire Fire and Rescue Service (DFRS) has been selected as the case study to illustrate the application of the proposed software integration approach. There are eight physically independent systems being used in the DFRS. The goal of the case study is to deliver a single virtual application through the AX based software integration so that the common used components in these isolated systems can be shared and inter-operated within this virtual application. These eight systems are:

- Mobilising System (MOB): provide the current locations of the available fire engines at every fire station in the DFRS, and record the emergency call details.
- Management Information System (MIS): provide the access to the fire incident databases, the personnel databases, and the relevant documents.

- Risk Assessment System (RISK): provide a building risk categorization and an access plan to higher risk premises.
- Geographical Information System (GIS): provide the risk information of buildings and areas in a visual way.
- Fire Safety System (SAFETY): is a database system, particularly designed for producing statistics reports for the combined fire authority and assisting in forecasting of fire safety activities and fire occurrence.
- Crime and Disorder System (CRIME): store all the crime and disorder information, such as malicious call, hoax fire call, and vehicle crime.
- Hydrant System (HYDRANT): provide hydrant information, including hydrant location, size, status, and maintenance history.
- Location Optimisation System (OPTIM): provide a computing environment for optimising fire station location, fire fighter and fire engine distribution.

6.1 Mathematical model of the DFRS

In terms of the above system description, there will be 8 participants in the AX. Each of them has a number of public components that might be contributed to the AX. The functionality of each component is represented in a set of symbolic variables, f_{ij} . The risk generated by the components is computed in the form of Equation 11, and is represented in a numerical value. Table 2 lists the desired sub-functions, which the AX is expected to offer. Table 3 summarizes the individual participants, the components provided by the participants, the functionality of each component, and the risk generated by them. The risk in Table 3 is computed by the frequency of failure multiplied by the severity index for each component. Table 4 shows the decomposition of the functionality of each component. The parameters of the mathematical model of the DFRS described in Equations 1 to 6 are list in Tables 2 to 4. Equation 6 has been replaced by Equation 11.

6.2 Results and analysis

After introducing the parameters in Tables 2, 3 and 4 into the mathematical models in Equations 1 to 10, R_{ij} in Equation 6 becomes a constant; the chromosome in Equation 7 is a 20-bits binary string. A GA developed in our previous work [19] is applied to find a 20-bits binary string: namely, the one which makes the fitness function or the risk have the smallest value among all possible values of the risk. We randomly choose 20 chromosomes to form the first generation. The one with the smallest value of the risk is placed in the beginning of each generation. The probabilities for the mutation and crossover operations are set as 0.1 and 0.6 respectively. A

satisfactory solution is derived after 30 generations. The final result and part of the generations are shown in Table 5. The components with the occurrence value 1 are selected to build the AX and are shared by each other. The components with the occurrence value 0 are not integrated in the AX for their functionalities have been implemented by the selected components and/or they make a significant contribution to the overall risk. Because all the sub-functionalities have been implemented in the AX, i.e. E in Equation 9 is equal to 0, the fitness function is equal to the overall risk. The minimum risk achieved is 1.775 with this search result, as shown in Fig. 3.

7. Conclusions

This paper discussed the AX based approach to software integration with the special emphasis on the component selection and the risk minimization. The AX based software integration is proposed by the need of sharing common software components among various departments within a same organization. The AX assimilates parts of existing systems from various applications and is shared by the partners. This paper has addressed one of the challenges in the establishment of the AX that is how to choose the components to achieve the desired functionality and to minimize the risk at the same time. For the sake of the simplicity, the software component risk is estimated by the frequency of failure multiplied by the severity index. The software integration problem has been re-formulated as a search problem and solved by using a GA. A case study from the DFRS is used to illustrate the applicability of the approach. The enumeration method could be used to find exactly the same solution for this case study through examining all possible combinations of available components for the AX so as to achieve the lowest risk and implement the desired functions as well. In this case study the number of all possible combinations of available components is 2^{20} . If the number of components and the number of the functions become bigger and bigger it might be impossible to find the solution by using the enumeration method. The advantage of using the GA for the AX based software integration is that the method is applicable for any scale problem and obviously more efficient than any enumeration.

Acknowledgements

Professor M Gell has made an invaluable contribution to the AX study, which forms the starting point of this paper. The data used in this work was collected from the Derbyshire Fire Rescue Service. Appreciation should be made to their kindly collaboration during the data collection.

References

- [1] M.J. Miller, Software integration, *Popular Computing*, 3 (2) (1983) 106-109, 118-&, 132-138.
- [2] S. Johnson, End-to-end integration: data, process, and content, *eAI Journal*, December (2001) 41-43.
- [3] D.S. Linthicum, *Enterprise application integration*, Addison-Wesley, London, 2000.
- [4] L. Robison, *Implementing B2B Commerce with .Net*, Addison-Wesley, 2002.
- [5] J. Schmidt, Enabling next generation enterprises, *eAI Journal*, July/August (2000) 74-80.
- [6] B.N. Rao, Extreme application integration, *eAI Journal*, May (2002) 36-40.
- [7] L. Yang, J. Hayes, M. Gell, Assimilation exchanges and integrated digital environments, *Proceedings of the 1st CIRP (UK) seminar on digital enterprise technology*, Durham, UK, 2002, pp. 303-305.
- [8] L. Yang, *Machine learning methodologies applied to fire risk management*, PhD thesis, the University of Derby, 2004.
- [9] J. Munson, T. Khoshgoftaar, Software metrics for reliability assessment, in: M. Lyu, (Ed.), *Handbook of Software Reliability Engineering*, 1996, pp. 493-529
- [10] S.M. Yacoub, H.H. Ammar, A methodology for architecture-level reliability risk analysis, *IEEE Transactions on Software Engineering*, 28 (6) (2002) 529-547.
- [11] US MIL_STD_1629A, *Procedures for performing failure mode effects and criticality analysis*, November 1984.
- [12] G. Cumming, K. Hanson, Interaction integration, *eAI Journal*, April (2002) 28-31.
- [13] M. Harman, B.F. Jones, Search-based software engineering, *Information and Software Technology*, 43 (2001) 833-839.
- [14] B. F. Jones, H. H. Sthamer, D. E. Eyres, Automatic structural testing using genetic algorithms, *Software Engineering Journal*, September (1996) 299-306.
- [15] D. E. Goldberg, *Genetic algorithms in search, optimisation and machine learning*, Addison-Wesley, Reading, MA, 1989.
- [16] D. Whitely, A genetic algorithm tutorial, *Statistics and computing*, 4 (1994) 65-85.
- [17] NASA-STD-8719.13A, *Software safety*, NASA Technical Standard, September 1997.
- [18] D.E. Neumann, An enhanced neural network technique for software risk analysis, *IEEE Transactions on Software Engineering*, 28 (2002) 904-912.
- [19] L. Yang, M. Gell, C.W. Dawson, M.R. Brown, Clustering hoax fire calls using evolutionary computation technology, *Lecture Notes in Development in Applied Artificial Intelligence series, LNAI 2718*, 2003, pp. 644-652.

Table 1 Severity classification

Category	Description	Severity index
Catastrophic	A failure may cause death or total system loss.	0.95
Critical	A failure may cause severe injury, major property damage, and major system damage.	0.75
Marginal	A failure may cause minor injury, minor property damage, and minor system damage.	0.50
Minor	A failure is not serious enough to cause injury, property damage, or system damage, but will result in unscheduled maintenance or repair.	0.25

Table 2 AX functions

Function (K=11)	Description
F ₁	Provide the available resource information in the DFRS, including available fire engines and fire fighters.
F ₂	Provide the current fire incident information.
F ₃	Provide the fire incident information during different periods of time
F ₄	Provide the fire risk categorization for a particular building
F ₅	Provide an access plan for any particular higher risk premise
F ₆	Provide the fire risk categorization for a particular area
F ₇	Provide a forecasting function of the fire incident occurrence
F ₈	Provide various crime and disorder information such as hoax fire call, malicious call.
F ₉	Provide the location information of hydrant points
F ₁₀	Provide the maintenance information of hydrant points
F ₁₁	Provide a computing environment for locating fire stations, fire fighters and fire engines.

Table 3 Components and risks

Participant (M=8)	Public components	Functionality	Risk (R_{ij} , $i=1, \dots, 8$; $j=1, \dots, N^i$)	Number of the components (N^i , $i=1, 2, \dots, 8$)
P ₁ , MOB	m ₁₁ , m ₁₂ , m ₁₃ , m ₁₄ , m ₁₅	f ₁₁ , f ₁₂ , f ₁₃ , f ₁₄ , f ₁₅	0.375, 0.1875, 0.375, 0.375, 0.3	5
P ₂ , MIS	m ₂₁ , m ₂₂ , m ₂₃	f ₂₁ , f ₂₂ , f ₂₃	0.025, 0, 0.0625	3
P ₃ , RISK	m ₃₁ , m ₃₂ , m ₃₃	f ₃₁ , f ₃₂ , f ₃₃	0.375, 0.375, 0.375	3
P ₄ , GIS	m ₄₁ , m ₄₂ , m ₄₃	f ₄₁ , f ₄₂ , f ₄₃	0.375, 0.3, 0.3	3
P ₅ , SAFETY	m ₅₁ , m ₅₂	f ₅₁ , f ₅₂	0.1, 0.25	2
P ₆ , CRIME	m ₆₁ , m ₆₂	f ₆₁ , f ₆₂	0.15, 0.15	2
P ₇ , HYDRANT	m ₇₁	f ₇₁	0.1	1
P ₈ , OPTI M	m ₈₁	f ₈₁	0.1	1

Table 4 Functionality of components

Function	Description	Decomposition
f_{11}	Provide the current available resource information in the DFRS, including available fire engines and fire fighters.	$f_{11} = F_1$
f_{12}	Provide the fire incident information during the latest eight hours.	$f_{12} \supset F_2$
f_{13}	Provide the fire risk categorization for a particular building.	$f_{13} = F_4$
f_{14}	Provide an access plan for any particular higher risk premise.	$f_{14} = F_5$
f_{15}	Provide the location information of hydrant points.	$f_{15} = F_9$
f_{21}	Provide the access to the fire incident database	$f_{21} \supset F_3$ $f_{21} \supset F_8$
f_{22}	Provide the access to the personnel database	$f_{22} \cap F_k = \Phi, k=1, \dots, 11$
f_{23}	Provide the access to the relevant documents	$f_{23} \supset F_{10}$
f_{31}	Provide the fire risk categorization for a particular building	$f_{31} = F_4$
f_{32}	Provide an access plan for a particular higher risk premise	$f_{32} = F_5$
f_{33}	Provide the fire risk categorization for a particular area	$f_{33} = F_6$
f_{41}	Provide the risk information of buildings	$f_{41} \supset F_4$
f_{42}	Provide the risk information of areas	$f_{42} \supset F_5$ $f_{42} \supset F_6$
f_{43}	Provide the location information of hydrant points	$f_{43} = F_9$
f_{51}	Produce statistics reports	$f_{51} \cap F_k = \Phi, k=1, \dots, 11$
f_{52}	Provide a forecasting function of the fire incident occurrence	$f_{52} = F_7$
f_{61}	Provide various crime and disorder information such as hoax fire call, malicious call.	$f_{61} = F_8$
f_{62}	Identify the higher crime and disorder areas	$f_{62} \cap F_k = \Phi, k=1, \dots, 11$
f_{71}	Provide the location and maintenance information of hydrant points	$f_{71} \supset F_9$
f_{81}	Provide a computing environment for locating fire stations, fire fighters and fire engines.	$f_{81} = F_{11}$

Table 5 Components proposed by GA and corresponding fitness value

Components proposed by GA																				Fitness
MOB					MIS			RISK			GIS			SAFETY		CRIME		HYDRANT	OPTIM	function f_{fitness}
m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	m_{21}	m_{22}	m_{23}	m_{31}	m_{32}	m_{33}	m_{41}	m_{42}	m_{43}	m_{51}	m_{52}	m_{61}	m_{62}	m_{71}	m_{81}	
0	0	0	0	1	0	1	0	1	0	1	1	0	0	0	1	1	1	0	0	7.975
0	0	1	1	0	1	1	0	0	0	1	1	1	1	1	0	0	0	0	0	7.225
0	1	0	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	0	7.088
1	1	0	0	0	0	1	1	0	0	0	1	0	1	1	1	0	1	1	0	6.900
0	1	0	1	1	1	0	0	0	1	0	1	0	1	0	1	1	1	0	0	6.488
1	1	1	0	1	0	0	1	1	0	0	0	0	0	1	1	0	0	1	1	6.225
0	1	0	1	1	1	0	0	0	1	0	1	0	1	0	0	1	0	0	1	6.188
1	0	0	1	1	0	1	1	0	0	1	1	0	1	0	1	0	1	1	1	5.763
1	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	0	5.450
1	1	1	0	1	1	0	0	0	0	1	1	0	1	0	1	1	1	1	1	5.063
1	1	1	0	0	1	0	0	0	1	1	1	0	1	0	1	0	1	0	0	4.788
1	1	1	0	0	0	1	1	1	0	1	0	1	0	0	1	0	0	1	1	4.500
1	1	1	0	0	1	1	0	0	1	1	0	1	0	1	1	1	1	1	1	3.863
1	1	1	0	0	1	0	1	1	1	1	0	0	1	0	1	1	0	0	1	2.950
1	1	1	0	0	1	0	1	0	1	1	0	0	0	0	1	0	0	1	1	2.225
1	1	0	1	0	1	0	1	1	0	1	0	0	0	0	1	0	0	1	1	2.225
1	1	1	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	1	1	1.775

Note: 1 represents the component selected; 0 represents the component not selected

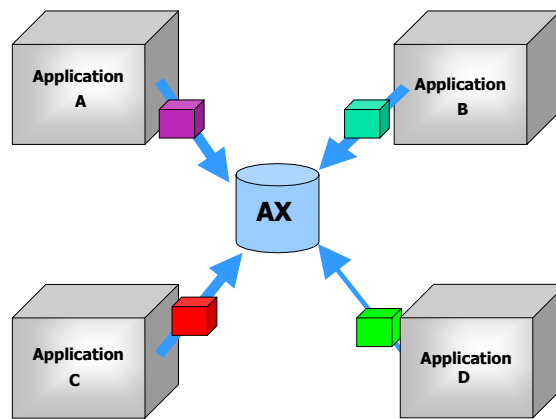


Fig. 1. Conceptual model of the AX [7, 8].

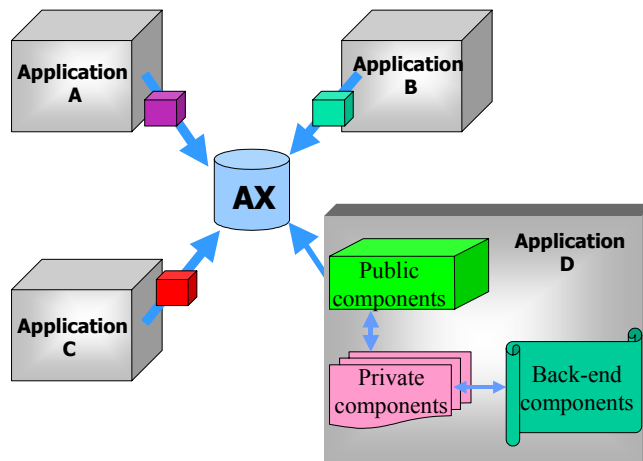


Fig. 2. Decomposition of an application in the AX [8].

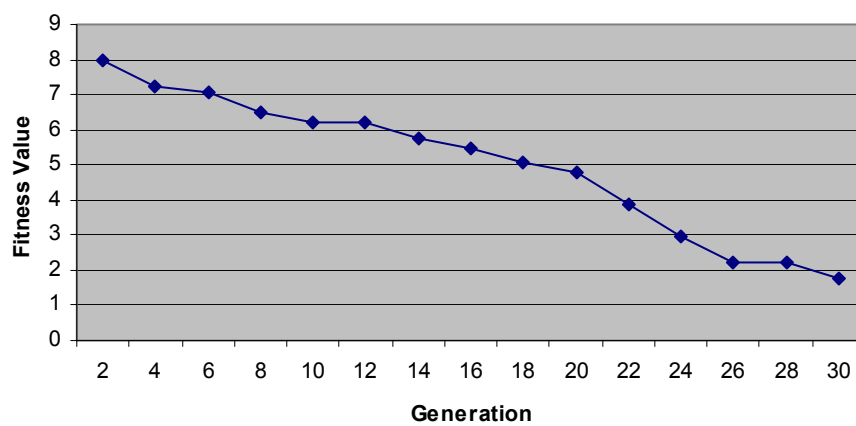


Fig. 3. Fitness values of the GA over the generations.