Chapter XX

Modeling Defects in E–Projects

John D. Ferguson
University of Strathclyde, UK

James Miller
University of Alberta, Canada

ABSTRACT

It is now widely accepted that software projects utilizing the Web (e-projects) face many of the same problems and risks experienced with more traditional software projects, only to a greater degree. Further, their characteristics of rapid development cycles combined with high frequency of software releases and adaptations make many of the traditional tools and techniques for modeling defects unsuitable. This paper proposes a simple model to explain and quantify the interaction between generic defect injection and removal processes in e-projects. The model is based upon long standing and highly regarded work from the field of quantitative ecological population modeling. This basic modeling approach is then subsequently tailored to fit the software production process within an e-project context.

INTRODUCTION

The Internet now forms a major platform supporting a wide range of commercial, educational, and social applications. While many of the early developments on the Web were based on client-server technology, the position has changed dramatically in the last few years with new technologies forming the basis of current applications. Today, many of the software industry’s current and new developments are destined for use either directly or indirectly, in sophisticated Web-based applications. Unfortunately, these developments have not come without cost, and industry is now faced with problems associated with verifying and maintaining Web-based applications. Current approaches to verification in the software industry are not succeeding. While systems grow evermore
complex, the percentage of total costs consumed by verification continues to grow as defect rates increase (Tassey, 2002).

The characteristics of e-projects differ from traditional information technology projects in a number of ways. E-projects are characterized as any project which evolves software that is deployed via the World Wide Web (Offut, 2002; Ricca & Tonella, 2001). E-projects are typically much smaller than many traditional projects; correspondingly, their production period is much shorter with projects lasting from several weeks to a few months not uncommon. They also tend to evolve at a much faster rate (typically the “shelf life” of an e-project system may be only 3 months) and hence their production is nearly always highly iterative in nature. Starting from a largely descriptive overview of the required functionality they often move through a quick build and test cycle, to be immediately faced with perhaps another quick build and test cycle. This situation can become even more extreme when using modern development methodologies; for example, when using the SCRUM methodology (Schwaber, 2004), the project manager often aims for iteration cycles as short as every 2 weeks.

Many traditional production processes, such as the waterfall or spiral models, defer verification and validation activities until late in the life cycle, resulting in these components often being considered as an afterthought to the production component. Further, the testing activities often get disassembled into artificial sub-activities such as unit testing, integration testing and system testing. The popular “V model” of an integrated testing process is a good example of this type of structuring; This model results in the production team being forced into conducting these sub-activities, regardless of the relative cost-benefit issues associated with each of these subactivities. Clearly, in any arbitrary project the costs and benefits associated with any verification and validation activity will be highly dependent on the domain of operation and the product under development. Since testing now accounts for more than half of the costs on many projects, any production methodology that fails to actively consider the costs and benefits of the testing activities is potentially wasting an extremely large amount of resources and failing to perfect the product for the marketplace. In contrast, e-projects systems with their short cycles tend to have a simpler, more time focused structure, typically comprising intertwined build and test components. For example, the recent rise of popularity of agile methodologies can be directly linked to the increasing number of e-projects in production. These methodologies often view the production processes in a different light from the traditional production methodologies; for example, in an XP production environment, the use of test-first programming (Beck, 2003) is extremely common.

Unfortunately, these differences in the production process of e-projects cause difficulties for many quantification models of artifacts within the software processes or products, including approaches to modeling defects. Traditional views of modeling defects tend to assume that we are using explicit quality targets, which is less likely to be the case within these new types of systems where managing the risks to the project and product are a more common approach. In addition, traditional models often assume that existence of a production model such as the “V,” where we have a long production period followed by a significant period of verification and validation. Commonly, we only model and analyze this verification and validation period. Again, this type of model will not translate effectively into the e-project domain. What is required is an alternative type of defect model which attempts to directly describe these frequent cycles of build and test processes. The principal aim and contribution of this article is to propose and construct such a model.
Motivation: Why Do We Need Defect Models for E-Projects?

Most software quality assurance processes deal with the conflict between processes that introduce defects into a software artifact or processes that seek to eliminate them. These deficiencies can be introduced at any, and all, stages of the development life cycle and similarly can be removed from each stage. Hence, we have a near continuous clash between these two opposing forces. A software quality assurance professional is interested in this situation for a variety of reasons; for example, its progress can be an indicator of current product quality and a pointer to project performance with regard to defects. Software engineering as a scientific discipline has an interest in modeling this confrontation in order to assist this group of professionals.

The principal reasons for constructing a model are to understand the processes at hand. Hence, this article takes the approach that both process types, which account for the presence or absence of defects in software artifacts, should be modeled. This is in contrast with many traditional approaches, which only model the impact of the removal process on the number of defects. This article argues that understanding the injectional processes is equally important. Regardless of the type of modeling, we would like to utilize the models to assist with, or to drive, economic decisions—in terms of the software produced and the production processes chosen for any particular project. Obviously, economic benefits can be derived from limiting the impact of the injectional processes and hence it is essential to model these aspects of the entire life cycle to allow a complete picture to be developed.

In addition, many approaches limit themselves to modeling the number of defects solely within the testing phase. However, this tends to limit the scope of the model; hence, they commonly deal with the specific quality control question: “Is the product ready for release?” This question is often answered by looking at the projected number of defects remaining within the software project against some deployment criteria. In this article, by modeling both types of processes it is hoped that the approach will eventually lead to models that are able to predict the potential economic impact, from a defect perspective, of the various options that exist when selecting a set of processes and approaches which will be utilized during the production of a software project—from its initial conception to its decommissioning.

Another driver towards modeling the entire process is the cost of defect removal. Boehm’s (1981) classic text argues that the cost of removal spirals as we move through the life-cycle. Hence if we accept this argument then we will want to focus our modeling and economic decision making upon the start of the project, where removal activities are cheap, rather than at the end where the costs are soaring. This point is further underlined by many companies, especially in areas like embedded systems, which report that postrelease costs of fixing a defect are often greater than two orders of magnitude higher than prerelease costs.

ARE CURRENT MODELLING APPROACHES APPLICABLE?

While it is tempting to assume that current approaches are applicable to this situation, it is believed that the realistic answer is no. Current approaches make several assumptions in a number of directions which imply that they are not effective modeling approaches to realistic formulations within this domain.

The most common class of models are parametric order-statistics models; a detailed overview of these models can be found in Lyu (1996). These models tend to derive parametric descriptions of the defects within a system, while making a number of assumptions and simplifications about the utilized development process and nature of the product. For example, a common assumption
is that an explicit single testing phase exists that is the final stage before release and that we are only interested in producing and using our model during this analysis. This is believed to be an inappropriate assumption for most e-projects. Further, again, to simplify the model, many approaches assume that any software fault is fixed immediately upon detection and that this “rework” process is prefect. Again, this assumption is unrealistic as projects nearly always have a percentage of modifications requests left “open” during any phase of their life cycle. In fact researchers have empirically demonstrated that many faults encountered by customers are injected during this rework phase (Boland & Chuiv, 2002). This issue has recently been recognized by some researchers who have attempted to construct “richer” models; for example, Gokhale (2004) incorporate a rework component via the numerical solution of their model, and Schneidewind (2004) incorporates a constant rework rate into his previous models (Schneidewind, 1993). However, all current approaches still only provide a very limited picture of these back-end processes and present them in a simplified fashion. For example, these processes are often highly variable in real projects, due to cost and scheduling constraints whereas even these models see them as constants. Nonparametric variations of order statistic models are possible (e.g., Barghour, Abdel-Ghaly, & Littlewood, 1998); normally these are based upon transforming the model into a Bayesian framework; however, these approaches are only currently witnessed limited exploration.

The final class of approaches is pattern matching or computational intelligence approaches. Here we are not attempting to provide a casual explanation of the phenomenon rather we are trying to discover the underlying pattern in the data (e.g., Tian & Noore, 2005). This class of approaches provides a flexible approach which can, in theory, represent iterative processes. However, these approaches suffer from two fundamental limitations that we seek to address. Namely, they offer no underlying causal explanation of the process, or processes, that they are modeling. This seriously limits the project managers ability to understand and explain the model and hence limits its application in areas where the rationale for the decision is considered an essential part of the decision-making process. Further, these approaches assume that the underlying processes are stationary over time. However, software production processes normally change requiring the modelers to regularly restart the pattern matching processes. Further, because of the lack of casual explanation we are in a limited position to understand when an event which introduces a “nonstationary” condition occurs; and instead of thinking through the casual implications of change, pattern matching approaches tend to rely on detecting them from changes in linear statistics. At best, this approach hopes to detect the nonstationary event after a delay, but often the event may not be able to identify the event, resulting in the disparate models, pre and post the event, being amalgamated. Software development processes are particular problematic for these techniques as the processes are regularly nonstationary, and their associated linear statistics are relatively unstable.

**Defect Injection and Removal Processes**

These processes are intrinsic within every component of the development life cycle. Every production process, whether it be requirements capture or documentation construction or the reworking of a defect reported by the customer base, can introduce defects. Since no production process is perfect, every process encompasses the possibility, if not the probability, of the introduction of errors and defects into the set of software artifacts that make up any real-world software project. Every time these artifacts are modified the possibility of transcription errors and oversights exist; these are often compounded by deficiencies introduced
by poor interpersonal communication processes between staff working upon the project. In turn, the background, or even the education of this staff, is likely to be less than perfect for the task at hand. Causes exist everywhere that can lead to the injection of defects and we are almost guaranteed that any and every attempt to evolve any of the artifacts will lead to the introduction of additional defects.

The opposite of the injection process is the removal process where the procedure is more obvious and well defined. The organization undertaking the software project will have explicitly decided to deploy various defect removal techniques as part of their project life cycle. The nature of these techniques will vary across the life cycle and across the different types of artifacts which make up the project. Typical examples are software testing, various forms of inspections and reviews, alpha and beta testing, usability evaluations, and so on. Each project will have its own life cycle and this life cycle will contain a variety of these components.

As the life cycle progresses, the two process types will have different impacts on the software project at various stages. Typically, a project goes through a construction phase (whether the construction is of a requirements document, code, a quality plan, or whatever is immaterial), during which the injection process will be in the ascendancy. This is commonly followed by a verification and validation phase where the removal process will be in the ascendancy. During both periods the other process will be operational (e.g., self-inspection during construction, and an oversight by a “test oracle” during system testing), but we can expect that its impact will be rather limited. Hence, we can anticipate that the balance will see-saw first in favor of the injection process and then reverse to become favorable to the removal process. Unfortunately, the exact nature of this oscillation between processes will be heavily dependent upon the type of processes deployed within the life cycle and hence is beyond the scope of any attempt to produce a generic model.

Within this oscillation it can be seen that the two opposing forces are not symmetrical—as we would see in a traditional conflict. Instead, the situation exists where the output of the injection process becomes the input of the removal process and the injection process has no direct means of influencing the behavior of the removal process. This asymmetric behavior needs to be a cornerstone of any analytical model.

It then has to be considered what is known about the empirical nature of both types of processes. Unfortunately, there is no solid information to draw upon here. It might be possible to hypothesize various abstract probability density functions for the likelihood of defects occurring and being removed. However, since no solid information exists this would be highly speculative.

This article takes an alternative approach by asking if an initial model can be derived by analogy. If another field has a well-established theory for modeling this type of interaction, then this would provide a reasonable starting point for building such a model.

**Choosing a Modeling Strategy by Analogy**

It is believed that a realistic model of this situation must encompass the following six characteristics:

1. Be based upon an iterative development cycle as defined by either, or both, the domain of the application or the development methodology.
2. Must explicitly model both the defect injection and removal processes.
3. Must not include any unrealistic or restrictive assumptions about either points 1 or 2.
4. Must be flexible enough to accommodate the many or varied situations that are encountered during a variety of development situations potentially including “wicked
projects” components. This requirement places a constraint on any model’s ability to predict and absolutely model any single situation. The model should aim at providing reasonable performance across a wide range of circumstances rather than worrying about being optimized for any single situation.

5. Must not be inferred from a small number of potentially unrepresentative preexisting defect data sets. Rather, it should be based upon common modeling ideas with wide applicability.

6. Must be suitable for a wide range of project management tasks which often require a casual explanation of the decision and its derivation.

The discussion above has outlined the basic requirements for the model. While there are undoubtedly a multitude of different approaches to modeling these basic concepts, which will be mathematically valid, this article postulates that these approaches will vary in usability. Hence, this work has undertaken the approach of looking for a “tried and trusted” mechanism in other fields which has been utilized for modeling these types of systems. Fortunately, one exists within the field of quantitative population ecology; this field has a long history of modeling ‘Predator and Prey’ systems. These systems provide a good analogy for the current situation, with defects being analogous to Prey and verification and validation activities being analogous to Predators. Other work

Figure 1. Lotka-Volterra: Phase diagram of defects(x) vs. validation-verification activities(y) for initial values of (x,y) at t=0 of [(0.1,0.1) (1.0,0.5) (1.2,1.2) (1.0,0.7)]; parameters [a:=1; b:=1; c:=1; d:=0.5]
within software engineering has also borrowed models from quantitative population ecology; for example, the work using capture-recapture theory (Chao, Lee, & Jeng, 1992) to estimate the number of defects remaining after a software inspection activity. Eick et al. (1992) and Miller (1999) draw a similar population-oriented analogy.

MODELS FROM QUANTITATIVE POPULATION ECOLOGY

Lotka Volterra models (Lotka, 1925; Volterra 1931) have a long history within various branches of scientific endeavor and can still be seen as an active area of research in many scientific disciplines (e.g., Hernandez-Bermejo, 1998; Pekalski & Stauffer, 1998). The equations have also found several applications outside of the traditional sciences; for example, Ormerod (1994) uses them to describe the average Canadian unemployment rate since the second world war. The simplest of their models proves an excellent starting point for modeling the interaction between defects and their removal. Specifically, the simplest Lotka-Volterra system for modeling a two-entity interaction is given by

\[
\frac{\partial x}{\partial t} = ax - bxy
\]

\[
\frac{\partial y}{\partial t} = cxy - dy
\]

Where,

x = the remaining number of defects;
y = the amount of verification and validation activities;
a = the rate of growth of defects, assuming that no verification and validation activities exist;
b = the rate of decrease in the number of defects due to verification and validation activities;
c = the rate of growth in verification and validation activities in response to finding and removing defects 1;
d = the rate of decay in verification and validation activities; if there are few, or no defects to be found.

Figure 1 illustrates typical behavior of x and y when plotted on a phase diagram where the arrows indicate the flow of time. The plots are drawn for different starting values of (x, y) and indicate periodic solutions or oscillations.

The model forms a good mathematical fit for the defect injection and removal scenario outlined above. In addition, this approach has a long history and has been proven successful with a wide range of applications. Within this application, the model utilizes the amount of verification and validation activities as a variable quantity, unlike some projects that fix the amount of effort on these processes at the start of the project. One of the proposed advantages of this approach is the ability to optimize this quantity throughout the project and hence it is expressed as a function of time.

One further implication is the limiting cases of the two equations. Thus in the complete absence of validation and verification activities:

\[
\frac{\partial x}{\partial t} = ax
\]

This integrates to:

\[
x_t = x_0 e^{at}
\]

Also, in the complete absence of defects,

\[
\frac{\partial y}{\partial t} = - dy
\]

This integrates to:

\[
y_t = y_0 e^{-dt}
\]
This clearly shows that these models are built upon independent exponential growth and decay functions, and hence the models fit our final set of requirements.

RESTRICTING GROWTH AND DECAY RATES

Introduction of a Finite “Consumption” Rate for Defects

Although it can reasonably be expected that validation and verification activities will find more defects when defects are relatively bountiful compared with periods when defects are relatively scarce, the increase in performance will clearly have an upper bound. The obvious rationale for this limitation is that validation and verification activities take a finite amount of time. This time can be subdivided into four components:

- Searching and locating defects
- Familiarization and refamiliarization with the software artifacts
- Reporting or fixing defects
- Other activities associated with the verification and validation activities; for example, team meetings or briefings, periods of learning, and so on

Clearly, although the first component will be (positively) impacted by the quantity of defects (i.e., the average search time will decrease with an increase in the number of defects), the other components are independent of the quantity of defects. Hence, the number of defects found is strictly limited by these additional components. In addition, while the third and fourth components can be considered as constants, the total amount of time spent on the second component is clearly proportional to the number of defects found. Thus, the total amount of time can be characterized as

\[ \sum_{i} T = \sum_{i} T_{\text{search}} + \sum_{i} T_{\text{fix}} + \sum_{i} T_{\text{familiar}} + T_{\text{misc}} \]

Hence, ignoring the constant components, the total amount of time spent on finding and processing defects can be approximately given by

\[ T^* = T_{\text{search}}^* + T_{\text{fix}}^* \]

Where \( * \) indicates summation over all defects found.

If we assume that that \( x_r \)-defects are processed during duration \( T^* \); then

\[ T_{\text{fix}}^* = x_r\bar{T}_s \]

Where \( \bar{T}_s \) is the average time spent on reporting and fixing any arbitrary defect.

If it is assumed that \( T_{\text{search}}^* \) can be modeled as a random process; and that the verification and validation activity is considered to search an “area” or “volume” (\( v \)) within a document or across several documents per unit of time.

Thus after time \( T^* \), the activity will have searched an “area” \( vT_{\text{search}}^* \) and will discover \( vxT_{\text{search}}^* \) defects. Where \( x \) can also be thought of as the density of defects within the searched area. (This also assumes that the verification and validation process is assumed to be perfect; that is, all found defects would be corrected recorded allowing subsequent removal.)

Hence,

\[ x_r = vxT_{\text{search}}^* \]

therefore,

\[ T^* = x_r\bar{T}_s + \frac{x_r}{vx} \]

and,

\[ x_r = \frac{vxT^*}{1+vx\bar{T}_s} \]
This approach to limiting the behavior of the exponential growth and decay functions also exists in the ecological literature; see Holling (1959) for examples of this approach in the ecological arena.

Finally, the area \( v \) representing software artifacts must be finite. Possible models for \( v \) include function point estimates and lines of code measures from traditional software systems. Alternatively, as Web-based systems advance hopefully such measures can be replaced by more specific measures directly related to verification and validation coverage-oriented measures such as the number of HTTP requests or transactions, or an information-theory based measure of the amount of XML-data transferred. This obviously places restrictions on the possible growth of the defect population and hence defines a maximum density of defects, which can exist. This fact places a further restriction on the growth model, which will be introduced in a later section.

**Increasing the Volume of Verification and Validation Activities**

It is also reasonable to assume that many projects which experience a large and increasing defect population may seek to counter this by introducing further verification and validation processes and procedures into their production processes. This adjustment could take many forms, such as the introduction of code inspections to supplement current testing approaches or the change of some component within an already existing component, such as the adoption of checklists or scenarios into the existing inspection process. Existing verification and validation staff may also add to the increase by increasing efforts or durations spent on these activities, as their current undertaking is seen as highly fruitful due to the large number of defects being found during this period of abundant deficiencies.

Although these responses are probable, it is very difficult to quantitatively envisage the impact of the responses on the system. Hence, due to the lack of information it was decided to model this additional component in an extremely simple manner, namely by predicting that any increase in the response by verification and validation activities is simply proportional to the number of defects believed to exist within the system at any point in time.

Finally, with regard to the final perceived component within the model, namely the statistical independence or dependence of verification and validation activities, it has been decided not to model these potential effects. The principal reason behind this decision was the lack of understanding of the scale of impact of this phenomenon. Although it is difficult to believe that some level of dependence does not exist, currently no empirical knowledge exists to suggest that it will have any great impact upon existing numerical models. In fact, a recent empirical investigation into this effect by Miller (2002), in the area of software inspection, suggests that no appreciable level of dependence exists, supporting the decision not to include any dependence component into the proposed model.

If it was decided to add further components to model this dependence, then again ecological modeling may well provide suitable pointers to the shape of these components. In ecological systems, “predator and prey” models are often extended to include interspecies competition and cooperation. It is believed that this further set of models could be adapted to encode any dependence component requiring addition.

**ADDING THE RESTRICTIVE COMPONENTS TO THE INITIAL MODEL**

Recalling that the number of defects found and removed is given by

\[
X_r = \frac{vXT^*}{1 + vXT^*}
\]
Then the rate of reduction in the defect count by all of the verification and validation practitioners, per unit of time, is given by

\[
\frac{vxy}{1 + vxT_x}
\]

Further, as noted the system should also restrict the rate of growth of defects (assuming that no verification and validation activities exist) as this area will be bound by the finite size of the software artifact. While it could be argued that a software artifact could in fact be populated by an infinite amount of defects, it is considered that this argument is extreme, and a more conservative approach of restricting this component is in line with common experience.

Following the work of Verhulst (1838), a limit on the density of defects appearing in any software artifact \((k)\) was introduced. The parameter \(a\) was reformatted so that as the population of defects increases, the rate of new introductions decreases. Verhulst’s solution was to introduce a logistic term into the equation, specifically

\[
1 - \frac{N}{k}
\]

This term decreases as \(N\) (an arbitrary population) approaches the maximum \(k\), reducing by an equal amount for each addition to the population, that is, the reduction is proportionate. This approach ensures that the behavior of the model is smooth and continuous and approaches the theoretical maximum asymptotically. Further, it incrementally penalizes the growth as it approaches the theoretical maximum modeling the increasing difficult of adding to the population when nearing saturation.

*Figure 2. Introducing a limitation to the defect growth rate in the Lotka-Volterra Model: Phase diagram of defects\((x)\) vs. validation-verification activities\((y)\) for initial value of \((x,y)\) at \(t=0\) of \((0.1,0.1);\) parameters \([a_{\text{max}}:=1; b:=1;k:=2;c:=1;d:=0.5]\)*
Therefore parameter $a$ is now defined as

$$a = a_{\text{max}} (1 - \frac{x}{k})$$

Where $a_{\text{max}}$ is the maximum growth rate possible within the software artifact.

Hence, the fully restricted defect population dynamic equation is now given by

$$\frac{dx}{dt} = a_{\text{max}} (1 - \frac{x}{k})x - \frac{vxy}{1 + vxT_y}$$

Figure 2 shows the effect on the basic Lotka-Volterra model of introducing a limitation on the density of defects appearing within a software artifact. Placing a restriction on the rate of growth of defects has ensured that $x$ never reaches a value exceeding the value of $k$. Further, the restriction on the growth of defects has led to a corresponding damping on the validation and verification activities as shown in Figure 3. If the value of $k$ is reduced, the defect population will be further restricted and the validation and verification activities required to obtain equilibrium will effectively tend towards zero, Figure 4. This represents a potential single survivor equilibrium solution that is unlikely to be found with typical application data.

The population dynamics of the verification and validation activities also require to be restricted, as outlined earlier. Here the restriction is a result of the proportional linkage between the sizes of the two populations rather than a restriction on the size of the activities. Again, the introduction of a logistic term has been chosen to implement this restriction.

Hence, the equation for the rate of change of validation and verification activities becomes

$$\frac{dy}{dt} = (cx - d)y(1 - \frac{y}{wx})$$

**Figure 3. Introducing a Limitation to the defect growth rate in the Lotka-Volterra Model: Plot of defects($x$) and validation-verification activities($y$) vs. $t$ for initial value of ($x,y$) at $t=0$ of ($0.1,0.1$); parameters [$a_{\text{max}}:=1; b:=1; k:=2; c:=1; d:=0.5$]**
Figure 4. Reducing the value of $k$, the maximum defect population, drives the validation-verification activities towards zero: Phase diagram of defects($x$) versus validation-verification activities($y$) for initial value of ($x,y$) at $t=0$ of (0.1,0.1); parameters $[a_{max}=1; b=-1; k=0.5; c=1; d=0.5]$.

Figure 5. Introducing all restrictive components to the Lotka-Volterra Model: Phase diagram of defects($x$) vs. validation-verification activities($y$) for initial values of ($x,y$) at $t=0$ of [(0,1,0.1) (1.0,0.5) (1.2,1.2) (1.0,0.7)]; parameters $[a_{max}=1; k=2; w=0.7; y=2; T=1; c=1; d=0.5]$.
Where $w$ is a weighting parameter to control the proportionality of the relationship between $x$ and $y$.

With this approach, as the ratio of $y/x$ tends towards $w$, the multiplying factor tends towards zero, limiting the growth in validation and verification activities. However, if $y/x$ is much less than $w$ the factor tends towards a value of one, giving normal Lotka-Volterra behavior.

Finally, the impact of introducing all three restrictive components to the model is shown in Figure 5. The consequences of restricting the proportionality relationship between $x$ and $y$ can be seen by the direction of the arrows to the left of the diagram. Overall, it can be seen that adding these restrictive components for growth and decay rates, with the parameters chosen in the examples, has had a damping effect on the oscillations within the model and led to a coexistence stationary equilibrium where the injected defects are being kept under control by a constant verification and validation effort.

CONCLUSION

To allow software professionals to develop well defined risk assessment and management strategies with regard to e-projects it is important to model the interaction between the processes that inject defects and the processes that remove them. Further, any model should be quantitative in nature and should support any arbitrary set of injectional and removal processes.

This article has developed a basic modeling framework based on the Lotka-Volterra predator prey interaction that allows a generic description of competing processes within the software production environment. The article has shown that the basic Lotka-Volterra model can be adapted to encode essential information about the interaction between processes and has the potential to give rise to a model appropriate to the interaction between injection and removal processes present within an e-project. The article has illustrated through example the effects of the restrictions on growth and decay rates introduced into the model, illustrating how they can lead to a coexistence stationary equilibrium between injection of defects and verification and validation activities. However, further work still has to be done to obtain suitable datasets that will allow estimates of model parameters that would lead to its application in predicting cost implications within the management of e-projects.

REFERENCES


