社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

カマルゴ クルズ アナ エリカ[†] 落水浩一郎[†]
[†] 北陸先端科学技術大学院大学 情報科学研究科
E-mail: †{erika.camargo,ochimizu}@jaist.ac.jp

Towards Objective Estimations of Software Implementation Progress

CAMARGO CRUZ ANA ERIKA[†] and KOICHIRO OCHIMIZU[†]
[†] Japan Advanced Institute of Science and Technology, School of Information Science, Asahidai 1-1, Nomi, Ishikawa, 923-1292 Japan
E-mail: †{erika.camargo,ochimizu}@jaist.ac.jp

Abstract Software implementation is often measured subjectively by software developers or project managers. In hopes to find a more objective and accurate methodology to assess software implementation progress timely, we propose the use of CBO (Coupling Between Objects) measures, using as a baseline UML approximations of the same measures. In brief, data is normalized, so that, as the code approximates to its design, some progress is "earned". Our initial evaluation suggests that our methodology can be helpful in the estimation of software implementation progress.
Key words software management, software implementation progress, UML, CBO

1. Introduction

Scheduling and monitoring progress of software development are crucial activities to meet the deliverable deadlines and, in general, to control a project. One of the reasons software products are delivered late is the inability to recognize on time that the project is falling behind schedule. Specifically talking of software implementation, progress reports are often estimated subjectively by software developers. Such reports are often open to misconception leading to inaccurate schedule estimations [1], [2], which is one of the major causes of software schedule slippage [3]. Therefore, objective and accurate measurements for assessing software implementation progress timely are clearly needed.

The number of Source Lines of Code (SLoC) and used memory in bytes are frequently used to measure the size of a software product [4]. However, assessing the percentage of what is being developed in terms of the same metrics is not possible, because their final values remain unknown till the project is completed.

Considering that the overall functionality of a software is achieved by relating every single part or module composing the software, code metrics measuring such relationship might be able to tell us how much of the totality of the software is being developed, only if, the final values of the same metrics are known beforehand.

One of the available object-oriented metrics that can measure a kind of relationship between the different modules of the software is the so-called CBO (Coupling Between Objects). CBO is part of the Chidamber and Kemerer suite of metrics (CK) which were conceived to measure the complexity of the design. While measured from the code, they have been related to managerial factors such as productivity, rework effort for reusing classes and design effort, maintenance effort and fault prediction [5], [6].

Now, let us say that we would like to use the CBO metric to track software implementation progress, we still would need a *target* value of what must be reached at the end of the implementation in terms of the same metric.

In previous research work, we studied how to approximate a subset of the CK metrics, including CBO, using UML diagrams. The evaluation results of our UML CBO metric yielded values which were close to the final code CBO values obtained from different small-size software projects [7]. Therefore, based on these results, we hypothesize that the same UML CBO metric can serve as the sought *target* value to estimate implementation progress readily.

This paper discusses our first attempts to monitor and estimate software implementation progress based on CBO measures, using as a baseline UML approximations of these.

The remainder of this paper is organized as follows: In section two, we provide some work related to our study. In section three, we present and evaluate our approach. Finally, we conclude this paper and present plans for future work in section four.

2. Related Work

In the following subsections, we discuss some work related to the monitoring and assessment of software implementation progress.

2.1 A UCP Based EVA Approach

In [2], Jinhua Li proposes to monitor software development projects using Earned Value Analysis (EVA) and Use Case Points (UCP), addressing also the need of measuring technical progress accurately. According to this study, one of the problems that makes difficult the application of EVA to software engineering projects is that it is difficult to measure technical progress of software projects accurately.

EVA is an industry standard method that measures the progress of a project at any given point in time. What it does basically is to compare the planned amount of work to what has actually been completed. First, some part of the budget is assigned to the different tasks of the project, and then, as the work is completed, part of its assigned budget is considered "earned". Concerns arise when assigning part of the budget to the different scheduled tasks, which are called "Product Values" (PVs), and when measuring technical progress and calculating the proper "Earned Values" (EVs). PVs are often assigned by experienced project managers who weigh the different tasks or milestones of the project, and then assign a part of the planned budget to each of them. On the other hand, EVs are often determined using fixed formulas, e.g. 50% of the PV is earned if the task is incomplete, and 100% of the PV is earned only if the task is completed.

In order to estimate a more accurate technical development progress, Jinhua Li proposes the use of UCPs as a baseline measure. UCPs is an effort estimation for a software project based on counting and weighing the number of actors, technical complexity factors and environmental factors of the software project. Because it is possible to calculate UCPs for the entire system at a very early stage, project managers can somehow assess how much every use case of the system is worth in terms of UCPs, and then easily translate UCPs into dollars or any other currency. However, this approach solves only a part of the problem. Still, determining the EVs of technical progress, particularly of the implementation, is based on a fixed formula which assigns 50% of the total UCPs when the task begins, and 100% after the task is completed.

2.2 A Degree of change in the CK metrics Approach

Another different approach was proposed by Ware et al., who have suggested the use of eight product measures to assist project managers in tracking change and progress within a release [1], and in determining release readiness [8].

For the purpose of assessing change and progress, they present a study case which involved the collection of a set of metrics, among them a subset of the Chidamber and Kemerer metrics. By applying Phase Analysis(注1), monthly snapshots of such metrics were taken through two major release cycles and a beta release of a commercial C++ application. The collected measures represent *the degree of change* from one month to another in the application, they do not represent the current state.

Ware et al. found that 83% of classes showing profound change also showed a period of consolidation. They indicate that a project manager should not expect a class to show profound levels of change immediately prior to a release. Also, if a developer reports his/her task nears completion, but the measures do not show the class stabilizing, the project manager should not take his/her assurances at *"face value"*. They conclude that the measures used are potential, useful indicators of project progress during the release cycle.

From our point of view, the metrics used in this approach provide better insights on how the software implementation is progressing. Yet, an accurate and objective measure on implementation progress, at any point in time, is not available. Moreover, we believe that project measures showing stabilization may in fact indicate stagnation, rather than near completion, and unless the developer acknowledge it, project managers would fail to recognize it.

3. Our Approach

The fundamental intuition underlying our method is explained as follows. We think of the construction of a software as if it were that of a house. The frame or skeleton of a house is first built to serve as a framework for outer coverings. Such skeleton is built of beams, floor joists and several other components. To ensure the structure's strength, these parts are sized and connected according to some blueprints. Likewise, a software skeleton is structured with several key components, that are related to each other in accordance to some blueprints, UML artifacts, so that the most important functionalities of the software can be achieved. Later, this

(注1): Phase Analysis is a technique used for modeling social behaviors. It helps to identify and test a model that explains the characteristics of long series of time-ordered observations [9].

skeleton can be further shaped and used to build outer coverings on top, just as it happens with the frame of a house. Therefore, if the most important components and relationships among them are detailed in the blueprints of the software, we could easily estimate the total number of relationships to be achieved, and use such a measure as a baseline to monitor the implementation progress of the software.

Among the available metrics that measure a kind of relationship between software components is the so-called CBO metric. We propose the use of CBO measures to monitor and assess software implementation progress, for the previous and the following reasons:

- CBO can be measured systematically at any time from code repositories.
- Previous research suggests that CBO measures are potential useful indicators of project progress during the release cycle of the software [1].
- Final code values of CBO can be approximated, at design time, using UML diagrams [7].

The remainder of this section provides the insights of the exploratory study carried out to justify and evaluate our approach. Such a study is divided into two main parts. The main purpose of the first part is to corroborate whether CBO measures can be useful on tracking software implementation progress. We compare the evolution of the CBO measures of 16 different packages and projects to their SLoC and number of Bytes evolution. Then, in the second part, we describe and evaluate our approach to measure progress implementation based on CBO measures.

3.1 Code Implementation Patterns: CBO vs Bytes vs SLoC

In this section, we present the findings of our exploratory study performed on 16 packages and projects, whose main purpose is to compare the pattern's behavior of cumulative values of CBO, SLoC(注2) and of the number of Bytes in the period of time during which every project was implemented.

Twelve packages of the Mylyn system (MYL) from Eclipse and four small-size software projects, developed by Master's students of JAIST, were used for our study. The JAIST systems were developed by different teams of two or three students during a 2-month lecture. Two of these teams followed the design of a Banking system (BNS) and the other two teams followed the design of an E-Commerce system (ECS). The basic design of the BNS and ECS is described in [10].

For every dataset, package or project, we followed the next procedure. Consider a period of time segmented into $T1...Tk$, where Ti represents the different points in time when a different version of one or various Java classes were updated into their CVS repository; and Tk is the last point in time when an update occurred, followed by the termination of the project.

At every time Ti, the total amounts of CBO, SLoC and Bytes were collected from the code of every project and package. All measures were normalized and then plotted for comparison. Some of the resultant graphs of the Mylyn project are shown in Figure 1, where the horizontal axis represents the different days when an update occurred, and the vertical axis represents the normalized measures of CBO, SLoC and Bytes. By visual inspection, CBO patterns behave very similarly to both SLoC and Bytes patterns. Similar patterns can be observed for the JAIST projects in Figure 2.

Furthermore, the correlation coefficients between CBO and Bytes measures range from 0.89 to 1, while those between CBO and SLoC range from 0.84 to 1. Considering a non-directional hypothesis (注3), these correlation coefficients can be regarded as statistically significant and unlikely to occur by mere chance to an $\alpha - value \leqq 0.001$.

From this data exploration, we conclude that the CBO patterns behave very similarly to the number of Bytes and SLoC patterns. Thereafter, CBO measures can serve as a baseline to assess code implementation progress at any point in time during the implementation because, different to the number of Bytes or SLoC which final values remain unknown till the completion of the project, code CBO final values can be approximated using UML diagrams (as explained in a later section). Moreover, code CBO measures can be collected systematically at any point in time during the implementation of the software.

3.2 Monitoring and Reporting Software Implementation Progress

This section provides the core of our contribution to monitor and asses software implementation progress. We used the same 4 small-size software projects developed in JAIST, and their basic design is provided in [10].

UML CBO measures were collected for every class of the three different projects, according to the following explanation. The UML CBO of a given class 'A' can be measured from UML collaboration diagrams as a count of all different messages sent to different objects by all objects of the given class 'A' [7].

When we used the total value of the UML CBO measures as baseline to determine implementation progress, we found that such value was considerably smaller than the total value of the code CBO reached at the end of the implementation. Therefore, when using raw measures, our UML CBO metric

(注2): This study considers SLoC as physical lines of code.

(注3): A non-directional hypothesis considers a non-zero correlation between two variables, either positive or negative.

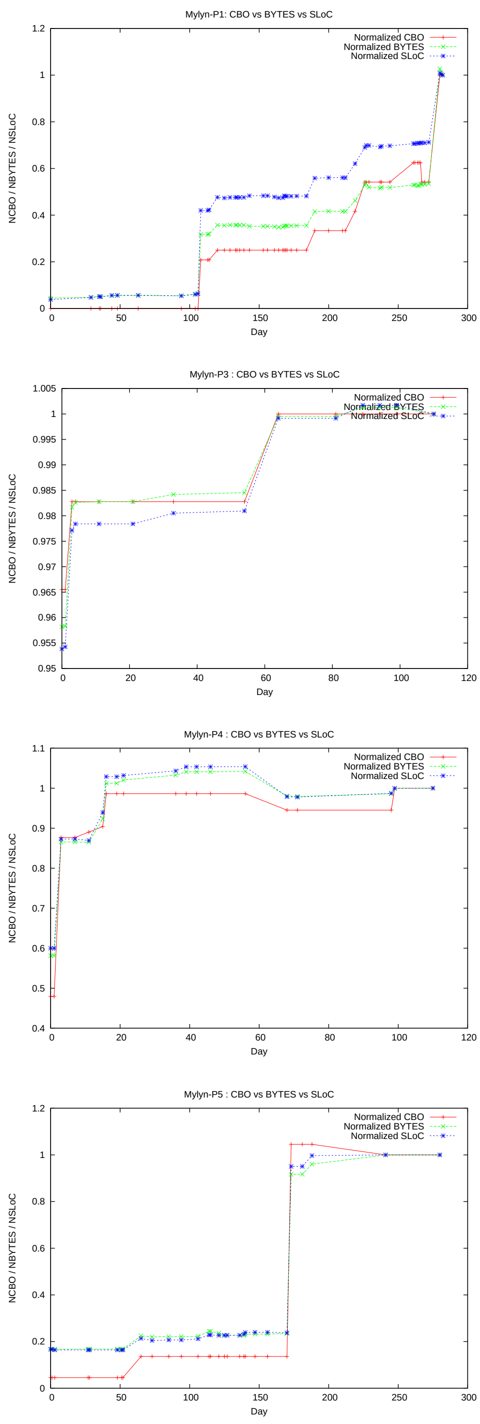

図 1 Mylyn Packages - CBO vs Bytes vs SLoC

cannot serve as a reliable baseline to determine progress. Yet, normalized measures are very close to each other, absolute error approximations are of 0.18, 0.26, 0.11 and 0.09 for the BNS-A, BNS-B, ECS-A and ECS-B projects respectively.

Since raw UML CBO measures are not close enough to the code CBO measures, we propose an approach which is based on *normalization* and *error approximations* of the code CBO measures to their corresponding UML measures. Moreover, the way of assigning progress was inspired by the EVA, which was briefly introduced in a previous section.

In short, for every Java class of the system, code CBO is measured, both of their UML and code CBO measures are normalized, then the absolute error is calculated. If the error is found to be greater than 0.7, no progress is "earned". This procedure is repeated while the project still under development. As the error decreases some progress percentage is "earned", and only when the error is small enough, is when the development of such a class, whose error nears zero, is regarded as completed.

Our approach implies that while the code CBO of *all* the classes considered in the design of a system are not equally proportional to their respective UML CBO measures, the project cannot be considered to be finished (at least in terms of coupling measures). In the following lines, we provide the detailed procedure which was carried out to determine the implementation progress of the four JAIST projects.

For every JAIST project, the following procedure was followed:

STEP 1. UML CBO measurement

Let $UCBO[j]$ be the vector of the collected UML CBO measures (according to our previous explanation) of the project, where $j = 1...N$, and N is the total number of Java classes in the project.

STEP 2. Product Values measurement

The Product Values of all the classes considered in the design of the project are calculated according to Equation 1.

$$PV_{UCBO}[j] = \frac{UCBO[j]}{\sum\limits_{j=1}^{j=N} UCBO[j]} \qquad (1)$$

STEP 3. Normalization of UML CBO measures

We used Equation 2 to normalize the UML metrics of the project. According to [11], if we assume that our data is normally distributed, a transformation such as that of Equation 2 guarantees that 99% of the data lies within the range [0,1]. Values outside this range can be truncated either to 0 or 1.

The resultant normalized measures are stored in a vector $NUCBO[j]$.

$$x' = \frac{\frac{x-\mu}{3\sigma} + 1}{2} \qquad (2)$$

Where x' is the new normalized value, x is the raw value of the measure, μ is the mean of the distribution of x and σ is the standard deviation of the distribution of x.

STEP 4. Collection of CBO code measures

Consider a period of time segmented into $T1...Tk$, where Ti represents the different points in time when a different version of one or various Java classes were updated into the CVS repository of the project; and Tk is the last point in time when an update occurred, followed by the termination of the project.

At every Ti, the CBO(注4) was measured for all the Java classes of the project, and a new vector, $CBO_{Ti}[j]$, was created with these measures.

STEP 5. Normalization of CODE CBO measures

Using Equation 2, $CBO_{Ti}[j]$ is normalized, and the resulting values are stored in $NCBO_{Ti}[j]$.

STEP 6. Calculation of Error Approximations

At time Ti, Equation 3 is used to calculate error approximations between what is being coded and what was designed.

$$ERROR_{Ti}[j] = NUCBO[j] - NCBO_{Ti}[j] \qquad (3)$$

STEP 7. Earned Progress Assignation

We create another vector, $EP_{Ti}[j]$, to store the earned progress of every Java class of the project at time Ti, according to the following rule:

```
if ($ERROR_{Ti}[j] \geqq 0.71$) then
        $EP_{Ti}[j] = 0$;
if (($ERROR_{Ti}[j] \geqq 0.5$) and ($ERROR_{Ti}[j] \leqq 0.71$)) then
        $EP_{Ti}[j] = 0.1 * PV_{UCBO}[j]$;
if (($ERROR_{Ti}[j] \geqq 0.31$) and ($ERROR_{Ti}[j] \leqq 0.5$)) then
        $EP_{Ti}[j] = 0.5 * PV_{UCBO}[j]$;
if ($ERROR_{Ti}[j] < 0.31$) then
        $EP_{Ti}[j] = 1 * PV_{UCBO}[j]$;
```

STEP 8. Assessment of Implementation Progress

Finally, at time Ti, the overall implementation progress is given by Equation 4.

$$TP_{Ti} = \sum_{j=1}^{j=N} EP_{Ti}[j] * 100 \qquad (4)$$

3. 3 Results

We applied our approach to determine the implementation progress of the four JAIST projects under study. At completion time, only the estimated implementation progress of the ECS-A reached 100%. As for the BNS-A, BNS-B and ECS-B, the resulting implementation progress remained at 89.4%, 71.7% and 73% respectively. This occurred because some classes remained with high error approximations $\geqq 0.31$ of their code CBO to their respective UML measures. Inspecting the code of these classes, we found that *all* of them did not follow the design according to the specified in [10]. Their code CBO values were found to be *smaller* than their respective UML CBO values, and in some cases zero. After eliminating these classes (two for the BNS-A and BNS-B, and one for the ECS-B), we re-applied our method and a considerable performance improvement was obtained.

The implementation progress using our approach (UCBO Baseline), CBO measures, SLoC and Bytes are shown in Figure 2. In general, the progress patterns using code CBO measures stabilize faster than SLoC and number of Bytes patterns. As for the progress patterns using our approach, the following observations can be made:

Observation 1.

Implementation progress patterns reached 100% completion more rapidly than when using code CBO measures. 100% is first reached when the code CBO patterns of BNS-A, BNS-B, ECS-A and ECS-B reached 100%, 95.8%, 81.2% and 94.7% respectively.

Observation 2.

We think that the remaining time for the implementation to finish would depend on several factors, such as programming styles and the type of software project. For example, if the software project involves too many graphical user interfaces, and the development team is first giving priority to the "skeleton" of the software, then such time would be much longer than the time lapsed to finish a project with fewer graphical user interfaces. Another example is when the development team dedicates long periods of time for refactoring at the end of the implementation phase, which would result in various but small variations of the progress patterns using either SLoC or number of Bytes, and more stable patterns using our approach and code CBO measures.

Observation 3.

In the specific case of the BNS-B, its implementation progress immediately after the project started increased to 70%. This occurred because code and UML measures resulted in being very close to each other at a very early stage when using normalization, which decreased approximation errors and yielded high progress percentages.

Observation 4.

As we can observe, for the ECS-B project there was a drastic fall of about 11% on the 23rd day. As in the previous observation, this did not occur because of a sudden decrease of the value of raw CBO measures, but because the distributions of code and UML measures at that specific time resulted in being very similar to each other, when using data normalization.

The last two observations are currently the major challenges of our study. If it is true that the normalization method that was used helped us to assess quite accurately when the project is about to be completed, we still need to address these challenges to improve the readiness of our approach.

4. Conclusion

This paper discusses our first attempts of estimating software implementation progress readily. Our approach is based

(注4): The CBO was measured considering only the coupling to classes of the same dataset (package or project).

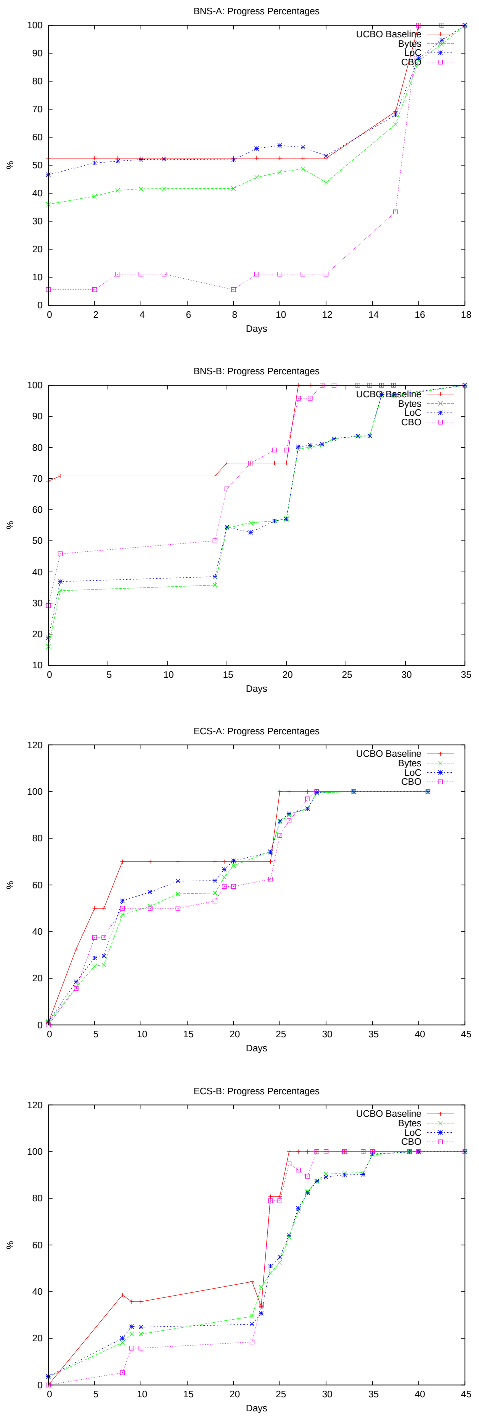

図 2 Progress Implementation Patterns

on CBO measures using as a baseline UML approximations of the same measures. The assessment of the implementation progress consists, basically, in the normalization of both UML and code CBO measures, so that, as the code approximates to its design, decreasing the error approximation, some progress is "earned".

Our initial results suggest that our methodology can be helpful in the estimation of software implementation progress, which encourages us to perform further improvements. So far, we have identified some challenges to be overcome concerning the normalization method. Furthermore, although we cannot tell precisely to what extent our approach can be applied to other software projects, we think that it can be helpful during the development of first releases, when most of the development efforts are dedicated to build the foundations and basic structure of a software.

文　　献

[1] M.P. Ware, F.G. Wilkie, M. Shapcott, and N.G. Lester, "The use of product measures in tracking code development to completion within small to medium sized enterprises," Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications, Washington, DC, USA, pp.263–270, IEEE Computer Society, 2008.

[2] J. Li, Z. Ma, and H. Dong, "Monitoring software projects with earned value analysis and use case point," ACIS International Conference on Computer and Information Science, pp.475–480, 2008.

[3] Z. Ma, J. Collofello, and D. Smith-Daniels, "Causes and solutions for schedule slippage: a survey of software projects," Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the IEEE International, pp.373 –379, Feb. 2000.

[4] S.H. Kan, Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[5] S.R. Chidamber, D.P. Darcy, and C.F. Kemerer, "Managerial use of metrics for object-oriented software: An exploratory analysis," IEEE Trans. Softw. Eng., vol.24, no.8, pp.629–639, 1998.

[6] M. Genero, M. Piattini, and C. Caleron, "A survey of metrics for uml class diagrams," Journal of Object Technology, vol.4, pp.59–92, 2005.

[7] A.E. Camargo Cruz and K. Ochimizu, "A uml approximation of three chidamber-kemerer metrics and their ability to predict faulty code across software projects," IEICE TRANSACTIONS on Information and Systems, vol.E93-D, no.11, pp.3038–3050, 2010.

[8] M. Ware, F. Wilkie, and M. Shapcott, "The use of intra-release product measures in predicting release readiness," International Conference on Software Testing, Verification and Validation, pp.230–237, 2008.

[9] Z. Xing and E. Stroulia, "Understanding phases and styles of object-oriented systems' evolution," Proceedings of the 20th IEEE International Conference on Software Maintenance, Washington, DC, USA, pp.242–251, IEEE Computer Society, 2004.

[10] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison Wesley-Object Technology Series Editors, Boston, MA, USA, 2000.

[11] S. Aksoy and R.M. Haralick, "Feature normalization and likelihood-based similarity measures for image retrieval," Pattern Recogn. Lett., vol.22, no.5, pp.563–582, 2001.