Using Time Series Models for Defect Prediction in Software Release Planning

James Tunnell
Central Washington University
Computational Science Program

March 27, 2015

Outline

Introduction

Release Planning Objectives

- Two primary objectives of software release planning are:
  - Improving functionality
  - Maintaining quality
- Both of these objectives are constrained by limits on development time and cost.

Quality Control

- Software defects (bugs) are inevitable
- Sufficient time should be available to ensure good quality (by testing and bug-fixing)
- Otherwise, there is a risk of
  - Low quality (failure to meet objective)
  - Schedule slip (failure to respect constraint)
- This quality control (QC) time can be allowed for by limiting the scope of work in the planned release

Quality Control (cont'd)

- To support release planning, QC time can be estimated
- Assumption: QC time depends (at least partly) on the number of software defects introduced
- Then, a basis for estimating QC time would be the predicted number of defects

Defect Prediction

- Approaches to defect prediction tend to focus on either
  - Code analysis
    - Lines of code
    - Number of decisions
    - Code churn
  - Historical information
    - Regression analysis
    - Time series modeling
- A multivariate time series model with exogenous inputs was chosen

Motivation

Release Plan Optimization

- A release plan is formed by selecting features and improvements to work on
- Release plans can be compared by the expected revenue they will generate
- This optimization problem is posed as The Next Release Problem (NRP)

Release Plan Optimization (cont'd)

- The NRP is an abstract optimization problem
- In practice, QC time should be considered to ensure constraints are respected
- With the help of a defect prediction model, QC time can be estimated
- In this context, release plans are being compared
- For a defect prediction model to be useful, it should depend in some way on the basic elements of the release plan (planned new features and improvements)

Explanatory Model

- Assumption: the number of defects in the future depends on more than just the number of defects in the past
- A defect prediction model that depends only on previous numbers of defects is not explanatory
- Such a non-explanatory model would always predict the same number of defects

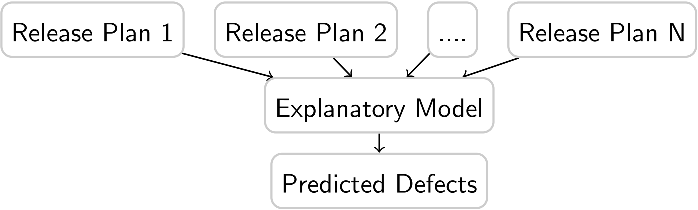

Figure : A non-explanatory model.

Explanatory Model (cont'd)

- A model could also depend on the key factors of a release plan
- This would be an explanatory model structure
- Such a model can potentially predict a different number of defects for every release plan

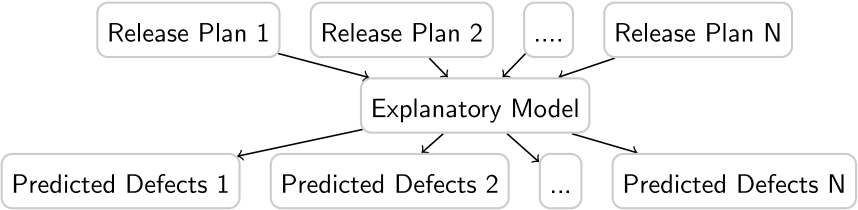

Figure : An explanatory model.

Related Work

Defect Prediction using Code Analysis

- Approaches using code analysis:
  - Akiyama used lines of code (LOC), number of decisions, and the number of subroutine calls [1]
  - Gafney also used LOC [3]
  - Henry and Kafura use information taken from design documents [4]
  - Nagappan and Ball use relative code churn (lines modified) [6]
- These approaches all depend on specific design or implementation information
- This information is not available at the release planning stage

Defect Prediction using Historical Information

- Approaches using historical information:
  - Li et al. extrapolate parameters of a regression model [5]
  - Singh et al. use an ARIMA time series model [7]
- Both approaches are non-specific to design or implementation
- However, neither approach is explanatory

Time Series Modeling

Time Series

- A time series is a collection of observations that occur in order
- The process underlying a time series is assumed to be stochastic (non-deterministic)
- Each observation might depend on one or more previous observations
- This dependence is termed autocorrelation

Autoregressive Models

- A basic autoregressive (AR) model is a linear combination of previous values
- A white noise term accounts for stochastic fluctuation
- An $AR(p)$ model for predicting a value X at time t is

$$X_t = c + \sum_{i=1}^{p} \phi_t X_{t-1} + \epsilon_t \tag{1}$$

where $\phi_1, \phi_2, ..., \phi_p$ are the p parameters, c is a constant, and ϵ_t is the white noise term

Autoregressive Models (cont'd)

- Extending the AR model to be multivariate results in a Vector AR (VAR) model
- This model can support time series for defect count, improvements, and new features

Endogeneity and Exogeneity

- Under a VAR model, the behavior of each time series is explained by both its own past values and the past values of the other time series
- This makes the variables endogenous
- An alternative is that a time series is only used to explain other time series
- This type of explanatory variable is called exogenous, and could be considered an input
- Exogenouse variables are not explained by the model

Endogeneity and Exogeneity (cont'd)

- The desired model does not need to explain features and improvements
- Instead, these are used to explain defects
- Planned features and improvements can be made exogenous
- By also considering exogenous variables, a VAR model would become a VARX model

Stationarity

- A stationary time series has time-invariant statistics
- The time series models so far require time series to be stationary
- Differencing a non-stationary series may produce a stationary series
- Stationary can determined by testing for trends

Deterministic Trends

- A time series with a deterministic trend has a non-constant mean
- The time series movements will generally follow the deterministic function
- Fluctuations above or below this function are non-permanent
- Such a time series is said to be stationary around a deterministic trend

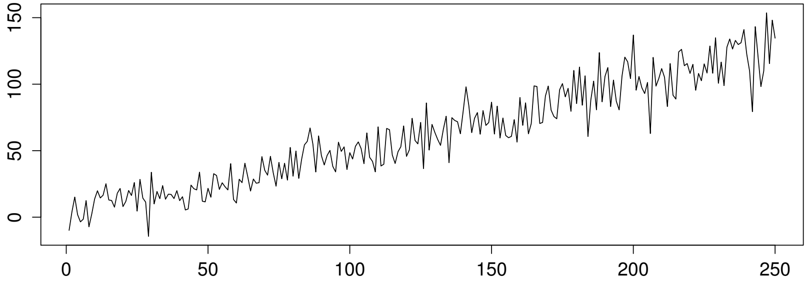

Figure : Time series with a deterministic trend.

Stochastic Trends

- A stochastic trend shows permanent effects due to random variations
- A series with stochastic trend will not necessarily fluctuate only close to the area of a deterministic function
- A time series with stochastic trend is non-stationary
- Differencing can be used to remove a stochastic trend

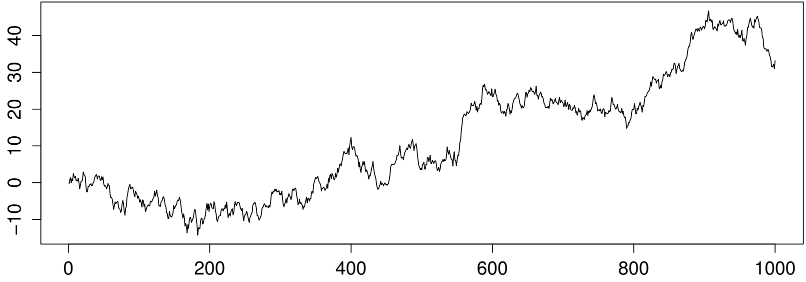

Figure : Time series with a stochastic trend.

Stationarity Testing

- A pure AR model of a time series with stochastic trend contains a unit root [2]
- Testing for the presence of a unit root can therefore be used to test for non-stationarity
- A unit-root test starts with the null hypothesis that an AR model has a unit root
- The alternative hypothesis is that an AR model of the time series does not have a unit root
- Next, a test statistic is measured
- If the test statistic is below the chosen significance level, the null hypothesis is rejected
- Rejecting the null hypothesis provides reason to accept the alternative hypothesis
- The Augmented Dickey Fuller (ADF) test is commonly used for unit root testing

Stationarity Testing (cont'd)

- On the other hand is a stationarity test
- This test starts with the null hypothesis that a time series is stationary around a deterministic trend
- If the test statistic is above some significance level, this shows that the null hypothesis can be accepted
- Then the time series should be considered stationary
- The Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test can be applied for testing stationarity.

Modeling Methodology

Data Methodology

References I

F. Akiyama.
An example of software system debugging.
In *IFIP Congress (1)*, volume 71, pages 353–359, 1971.

P. H. Franses.
Time series models for business and economic forecasting.
Cambridge university press, 1998.

J. E. Gaffney.
Estimating the number of faults in code.
Software Engineering, IEEE Transactions on,
SE-10(4):459–464, July 1984.

References II

S. Henry and D. Kafura.
The evaluation of software systems' structure using quantitative software metrics.
Software: Practice and Experience, 14(6):561–573, 1984.

P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam.
Empirical evaluation of defect projection models for widely-deployed production software systems.
SIGSOFT Softw. Eng. Notes, 29(6):263–272, Oct. 2004.

N. Nagappan and T. Ball.
Use of relative code churn measures to predict system defect density.
In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 284–292. IEEE, 2005.

References III

L. L. Singh, A. M. Abbas, F. Ahmad, and S. Ramaswamy.
Predicting software bugs using arima model.
In *Proceedings of the 48th Annual Southeast Regional Conference*, page 27. ACM, 2010.