# Using Time Series Models for Defect Prediction in Software Release Planning

James Tunnell
Central Washington University
Computational Science Program

March 30, 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

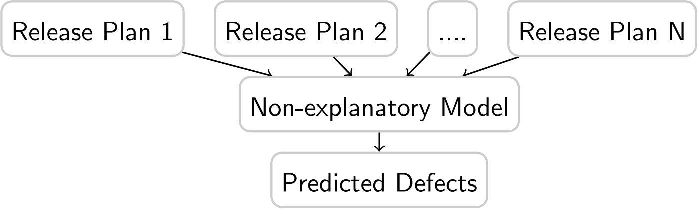


Figure : *A non-explanatory model.*

# 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

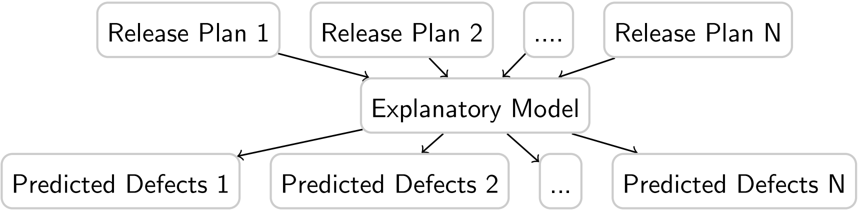


Figure : *An explanatory model.*

# 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

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

where $\phi_1, \phi_2, ..., \phi_p$ are the $p$ parameters, $c$ is a constant, and $\epsilon_t$ 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

Endogeneity

- 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*

Exogeneity

- An alternative to this is when a variable is not explained at all by a model
- Rather, the variable is used to explain other time series
- This type of explanatory variable is called *exogenous*

# 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 be 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

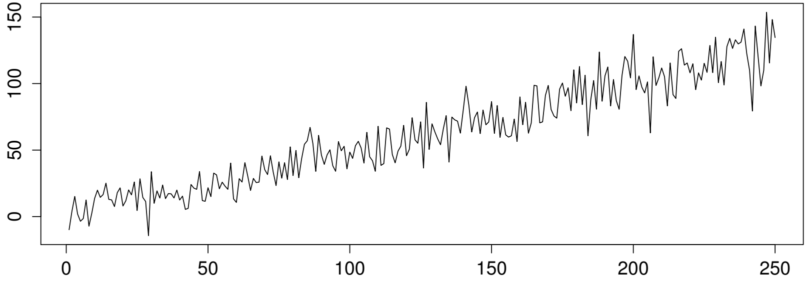


Figure : *Time series with a deterministic trend.*

# 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

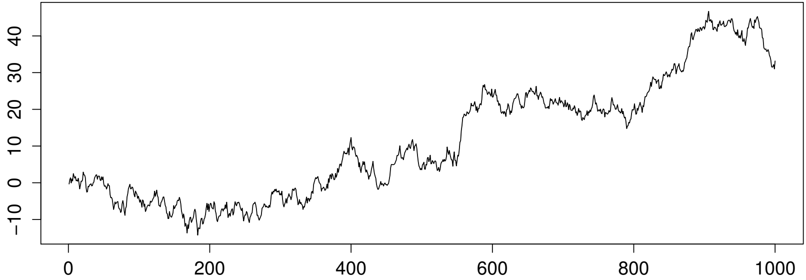


Figure : *Time series with a stochastic trend.*

# Stationarity Testing

- A pure AR model of a time series with stochastic trend contains a unit root [4]
- 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 p-value 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

# Time Series Modeling Methodology

- Time series modeling methodology typically involves
  1. Specification
  2. Estimation
  3. Diagnostic Checking
  4. Selection

# Specification & Estimation

- A $VARX(p)$ model is specified by choosing an order $p$
- Model order is the number of autoregressive terms
- This affects the number of parameters included in the model
- To avoid having too many parameters relative to the number of observations, we use

$$p_{max} = \left\lfloor \frac{n}{mK_{min}} \right\rfloor \tag{2}$$

  - $n$ is the number of time samples
  - $m$ is the number of time series
  - $K_{min}$ is the minimum acceptable ratio of observations to parameters

- Models parameters are estimated for orders $1, 2, ..., p_{max}$

# Diagnostic Checking

- Diagnostics can tell if a model should be rejected
- First diagnostic is for stability
  - AR model can have infinite impulse response
  - To be stable, the roots of the characteristic equation must lie outside the unit circle [3, p. 56]
  - Equivalently, the inverse of the roots must lie inside the unit circle
- Next diagnostic is residual autocorrelation
  - Model residuals should be indistinguishable from white noise
  - White noise is uncorrelated (no autocorrelation)
  - Ljung-Box test forms a statistic from the autocorrelation of the residuals

# Model Selection

- Model selection criteria are used to compare models according to their fit
- Penalties for residual error and the number of parameters
- Some common selection criteria
  - Akaike Information Criterion (AIC)
  - AIC with correction (AICc)
  - Bayesian Information Criterion (BIC)
- Parameter penalty is more severe for BIC and AICC than for AIC [2]
- AIC will be used, since the number of parameters is already limited in the specification step

# Data Methodology

# Data source

- Data for time series modeling will be derived from project historical data
- This historical data can be found in the project issue tracking system (ITS)
- The issues in an ITS can be bugs, features, improvements, etc.
- The *MongoDB* software project was selected to try out the modeling methodology
  - The project has been actively developed since 2009
  - Data from versions 0.9.3 through 3.0.0-rc6 are used
  - This dataset contained 7042 issues

# Data Collection

- *MongoDB* uses *JIRA* for issue tracking and project management
- Issue data was exported from the project's JIRA web interface as XML data
- Issue data was extracted from the XML, and the following fields were kept:
  - Creation date
  - Resolution date
  - Type
  - Priority

# Data Cleansing

- Not all of the data was preserved for modeling
- No-change issues
  - Only issues with resolution *fixed*, *complete*, or *done* were kept
  - Other issues did not result in any change, and were not included
- Orphan sub-tasks
  - Issues that are sub-tasks are first converted to be the same type as the parent issue
  - Sub-tasks whose parent issue is not in the dataset are considered orphans, and discarded
  - Orphan sub-tasks can not be identified as improvement or new feature
  - 20 (0.28%) orphaned sub-tasks were in the dataset

# Data Sampling

- The dataset was operated on to prepare it for time series modeling
- First, data was sampled at regular intervals, measuring
  - Number of bugs created
  - Number of improvements resolved
  - Number of new features resolved
- A 7-day sampling period was used

# Data Sampling (cont'd)

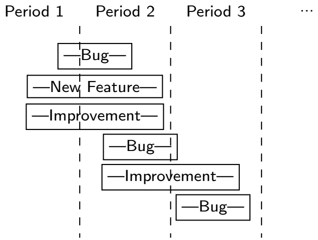


Figure : *Sampling example issue data.*

| Period | Improvements Resolved | New Features Resolved | Bugs Created |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 2 | 1 | 1 | 1 |
| 3 | 1 | 0 | 1 |

Table : *Results of sampling example issues.*

# Establishing Stationarity

- Establish stationarity by testing
  - ADF unit root test
  - KPSS stationariy test
- If test results agree, then no differencing is necessary
- Otherwise, difference data and retest

# Time Windowing

- Assumption: the software development process underlying a given project might change over time
- The VARX model does not accommodate a changing process
- To account for a changing process, data will be time windowed
- Data will be used for modeling only if it occurs within the time window
- This should limit the amount of process change the model is exposed to

# Results

# *MongoDB* Time Series Data

Time series data was obtained by sampling the *MongoDB* dataset with a 7-day sample period

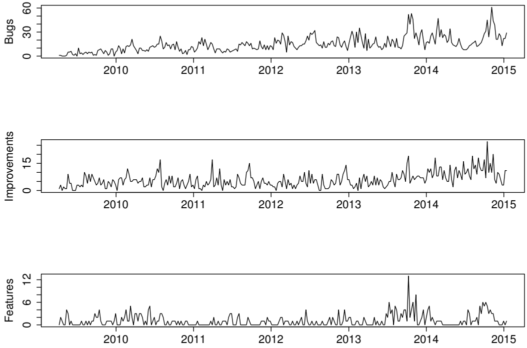


Figure : *MongoDB time series data*

# Stationarity & Differencing

- At first, ADF and KPSS test results disagreed
- After differencing, test results agreed

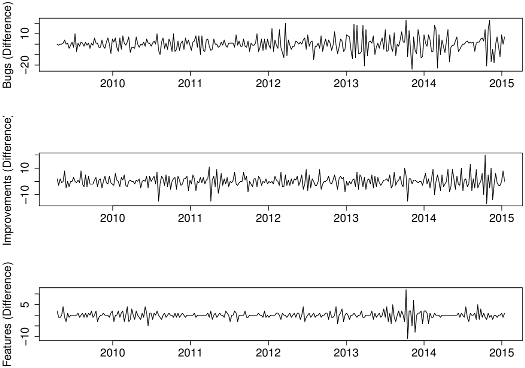


Figure : *MongoDB time series data after differencing*

# Time Windowing

- A 78-week time window (approximately 18 months) was chosen
- Three of these windowed periods, non-overlapping, were used for modeling
- Since the data is being differenced, the first sample (week) is skipped
- These windowed periods are denoted $W_{2-79}$, $W_{80-157}$, and $W_{158-235}$
- Modeling methodology is applied to each

# Model Specification, Estimation, and Diagnostic Checking

- Using $K_{min} = 4$, maximum model order is obtained by

$$p_{max} = \left\lfloor \frac{78}{(3)(4)} \right\rfloor = \lfloor 6.5 \rfloor = 6 \tag{3}$$

- Models of order 1 through $p_{max} = 6$ were estimated for diagnostic checking
- All models were found to be stable
- Several model orders were found to be inadequate by the Ljung-Box test:
  - Orders 1-2 for period $W_{2-79}$
  - Order 5 for period $W_{158-235}$

# Model Selection

- Models found to be stable and not inadequate were considered for selection
- A different model was selected for each windowed period
- Lower AIC score is better

| Model order | AIC score $W_{2-79}$ | $W_{80-157}$ | $W_{158-235}$ |
|---|---|---|---|
| 1 | N/A | **429.8** | **477.9** |
| 2 | N/A | 439.3 | 482.4 |
| 3 | 400.8 | 440.9 | 489.7 |
| 4 | **400.3** | 450.2 | 499.9 |
| 5 | 404.0 | 456.7 | N/A |
| 6 | 414.9 | 461.7 | 508.8 |

Table : *Results of model selection, using AIC score to compare models of different order.*

# One-step Predictions

The fit for each selected model is demonstrated by plotting one-step predictions along with actual values.

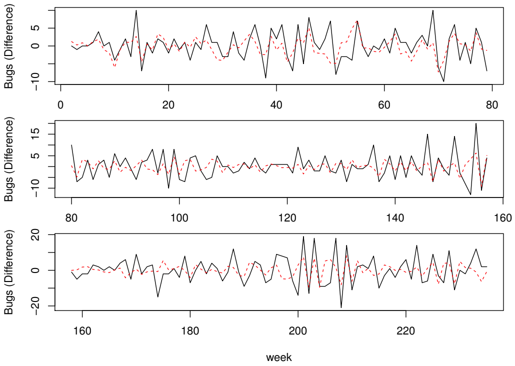


Figure : *Actual values (solid) vs. one-step predictions (dotted), for each model selected by AIC score.*

# Forecasting Results

- A range of hypothetical future values for improvements and new features were used to make defect predictions
- This simulates the use of defect prediction for release planning
- Single-step, out-of-sample forecast
- Inputs were differenced, and difference was removed from output
- Results include 75% and 90% confidence intervals
- Forecast results are shown only for the first time window, $W_{2-79}$

# Forecasting Results (cont'd)

- The actual number of improvements and features was 4 and 0
- Actual number of bugs was 18
- For the actual input values, the 90% confidence interval does not include 18

Table : *Forecasting at the end of the first time window, $W_{2-79}$.*

| Improvements | Features | 90% low | 75% low | Mean | 75% high | 90% high |
|---|---|---|---|---|---|---|
| 2 | 0 | 5.61 | 6.72 | 9.31 | 11.89 | 13.00 |
| 2 | 1 | 5.54 | 6.66 | 9.24 | 11.82 | 12.93 |
| 2 | 2 | 5.48 | 6.59 | 9.17 | 11.75 | 12.86 |
| 2 | 3 | 5.41 | 6.52 | 9.1 | 11.69 | 12.8 |
| 4 | 0 | 6.4 | 7.51 | 10.09 | 12.68 | 13.79 |
| 4 | 1 | 6.33 | 7.44 | 10.03 | 12.61 | 13.72 |
| 4 | 2 | 6.27 | 7.38 | 9.96 | 12.54 | 13.65 |
| 4 | 3 | 6.2 | 7.31 | 9.89 | 12.48 | 13.59 |

# Forecasting Results (cont'd)

- Low accuracy for the predictions is concerning
- For the next window, $W_{80-157}$, the actual number of future bugs was 17
- This was inside the 90% confidence interval, which spanned from 13.38 to 18.00
- This result conflicts with that of the previous window

# Sliding Window Forecasts

- How useful is the VARX model in general, considering these conflicting results?
- To find out, a sliding 78-week window was used
- The sliding window started at the first sample period, and was shifted by one sample period after modeling
- Only the actual number of improvements and features were used in this forecasting

# Sliding Window Forecasts (cont'd)

- Errors between the mean forecasted and actual number of bugs is shown as a histogram
- The histogram appears to be normally distributed (good)
- The variability is quite large (bad)
- The actual number of bugs was inside the 90% confidence interval for 23.87% of the sliding window ranges

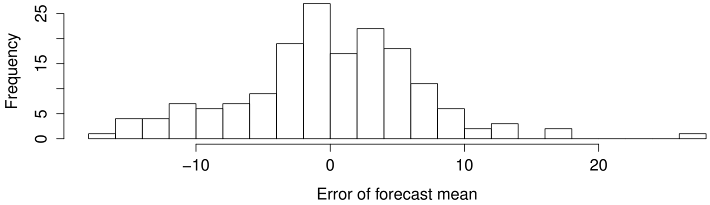


Figure : *Histogram of errors in forecast mean obtained using a 78-week sliding window.*

# Conclusion & Future Work

# Conclusions

- The VARX modeling methodology was successfully applied to the time series data collected from the *MongoDB* project
- Models were created for each of three time windows
- A model was selected for each window
- Forecast results using the models were inconclusive
- A better picture of the prediction performance was obtained using a sliding window
- This resulted in a normally distributed error in the mean forecasted values
- A low proportion (23.87%) of the sliding window ranges included the actual number of bugs in the 90% confidence interval
- These results may indicate that a VARX model will not be useful to make predictions for the the MongoDB dataset

# Future Work

Having applied the VARX time series model to one project dataset, a next step is to apply the methodology to other software project data sets, such as *Eclipse* or *Mozilla*, to more conclusively determine the model's usefulness.

# References I

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

S. Bisgaard and M. Kulahci.
*Time series analysis and forecasting by example*.
John Wiley & Sons, 2011.

G. E. P. Box, G. M. Jenkins, and G. C. Reinsel.
*Time Series Analysis*.
John Wiley, 2008.

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

# References II


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

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.

# References III

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.

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.