Empir Software Eng (2017) 22:57–79
DOI 10.1007/s10664-015-9415-3

# Defect propagation at the project-level: results and a post-hoc analysis on inspection efficiency

**Padmal Vitharana** [1]

Published online: 20 November 2015


**Abstract** Inspections are increasingly utilized to enhance software quality. While the effectiveness of inspections in uncovering defects is widely accepted, there is a lack of research that takes a more holistic approach by considering defect counts from initial phases of the development process (requirements, design, and coding) and examining defect propagation where defect counts are aggregated to the project-level (i.e., application-level). Using inspection data collected from a large software development firm, this paper investigates the extent of defect propagation at the project-level during early lifecycle phases. I argue that defect propagation can be observed from the relationship between defects in the prior phase and the defects in the subsequent phase. Both Ordinary Least Squares and 3-Stage Least Squares analyses support the hypotheses on defect propagation. Moreover, results show that the inspection efficiency (defects per unit inspection time) decreases as the software product progresses from requirements to design to coding. A post-hoc analysis revealed further insights into inspection efficiency. In each phase, as the inspection time increased, efficiency reached an optimal point and then dropped off. In addition, a project's inspection efficiency generally tends to remain stable from one phase to another. These insights offer managers means to assess inspections, their efficiency, and make adjustments to the time allotted to inspect project's artifacts in both the current and the subsequent phase. Implications for managers and future research directions are discussed.

**Keywords** Inspections · Software quality · Defect propagation · Requirements · Design · Coding

## 1 Introduction

In software development, each intermediate artifact has quality attributes that impact the quality of the ultimate software product (Jacobs et al. 2007). While defects injected into the

Communicated by: Tony Gorschek

✉ Padmal Vitharana
padmal@syr.edu

[1] Whitman School of Management, Syracuse University, Syracuse, NY 13244-2130, USA

product during each development phase degrade quality of that phase, defects that escape from one phase to the next exacerbate the effect on the overall quality (Harter and Slaughter 2000; Kemerer and Paulk 2009; Westland 2004). One defect in a given phase could lead to multiple defects in the subsequent phase if left uncorrected (Freimut et al. 2005; Wagner 2006). Defects propagating from a previous phase are more costly to detect and correct than those originating in the current phase (Laitenberger 2001; Laitenberger et al. 2001; Mandala et al. 2012; O'Neill 1997). Defect propagation refers to instances when defects from one phase escape into the next phase. The fact that defects from early phases of development that remain in the software are expected to be even costlier to remove from the implemented system than those defects from latter phases (Boehm et al. 2000; Johnson 1998; Kemerer and Paulk 2009) further exacerbates the adverse impact of propagation.

An important way to control product quality is to monitor defects throughout the development process (Biffl 2000; Unterkalmsteiner et al. 2012). Many authors identify the need to design quality into the software product from its inception instead of testing at the end of product completion (e.g., Fenton and Neil 1999; Harter and Slaughter 2000). Inspecting each deliverable such as requirements specification, design document, and programming code, and correcting defects at their source offers the best approach to achieving this goal (Biffl and Halling 2003; Ebenau 1994; Kelly et al. 1992; Sauer et al. 2000; Strauss and Ebenau 1994; Vitharana and Ramamurthy 2003). Inspection refers to the peer review of a software development artifact by a group of trained individuals for the purpose of uncovering defects. A project-long strategy on quality employing inspections[1] helps managers counter the effects of defect propagation on the quality of final software product (Fagan 1986; Fagan 2002; Mandala et al. 2012; Miller et al. 1998; Rigby et al. 2014). The widely heralded Capability Maturity Model Integration (CMMI) to quality assurance specifies inspections (i.e., peer reviews) to achieve maturity level 3 (defined process).

Research to date on defect propagation during early lifecycle phases including requirements, design, and coding is largely anecdotal (e.g., Freimut et al. 2005; Myers 1979; Wagner 2006) and consists of a few empirical studies conducted a decade or more ago using a handful of software projects (e.g., Briand et al. 1997; Kelly et al. 1992; Raz and Barad 2004; Yu et al. 1988). Furthermore, there is also a lack of research that takes a more holistic approach to examining propagation by considering defect counts from inspections aggregated at the project-level and comparing them across requirements, design, and coding phases. The dearth of empirical evidence on defect propagation can be attributed to the difficulty in systematic data collection across early phases of the development lifecycle. While inspections' ability to detect defects at their origin before they propagate to subsequent phases is widely accepted (e.g., Laitenberger et al. 2001; Laitenberger et al. 2002), more empirical research employing recent and larger datasets aggregated at the project-level from early lifecycle phases is needed to gain more insights into defect propagation and its impact on inspections during subsequent phases. To address this gap in research, I use inspection data from a large software development firm aggregated at the project-level in requirements, design, and coding phases to investigate defect propagation from one phase to the next. I argue that defect propagation can be observed from the relationship between defects in the prior phase and the defects in the next phase. This line of reasoning is based on the assumption that higher the defects found in phase X, higher the defects not uncovered in phase X and thus, propagating to phase $X+1$ resulting in a higher number of defects that could be discovered in phase $X+1$ by software

[1] The role of inspection is to merely find defects, not correct them.

inspection. The results show that there is a relationship between defects uncovered in the prior phase and the defects uncovered in the current phase, hence providing evidence of defect propagation from one phase to another. Moreover, results also show that inspection efficiency decreases as the software product moves from requirements to design to coding. A post-hoc analysis provided further insights into inspection efficiency both within and across lifecycle phases. This paper contributes to our understanding of the impact of defect propagation on the inspection of project's artifacts during subsequent phases that influences the overall software quality. Moreover, with the increasing emphasis on the cost of inspection (Briand et al. 1998; Briand et al. 2000; Porter et al. 1997), findings from the post-hoc analysis offer managers means to realize greater return on inspection investment.

The paper is organized as follows. Section 2 offers background literature on defect propagation. Research model and hypotheses are presented in Section 3. Section 4 describes the methodology followed by analysis in Section 5. Discussion including post-hoc analysis on inspection efficiency is presented in Section 6. Threats to validity are presented in Section 7 while implications for practice are offered in Section 8. The paper concludes with directions for future research in Section 9.

## 2 Defect Propagation

There is an important relationship between software defects that are observable and software quality that is assumed. In highlighting different views on quality, Kitchenham and Pfleeger (1996) identify a manufacturing view to represent a critical aspect on software quality. In looking at quality from the manufacturer's point of view, this view sees quality as conformance to specification during production and after delivery. These authors suggest the use of defect counts as a way to measure quality from the manufacturing view. That is, the number of defects uncovered during inspection of a software artifact (e.g., requirements specification) offers insights into the quality of that artifact as well as subsequent artifacts beyond that phase (e.g., design documents, code, etc.) because of the propagation effects. As inspections are more useful in uncovering semantic defects (validation) because they are expected to have a greater impact on the development than syntactical defects (verification), they further reduce the likelihood of semantic defects that are more difficult to detect and correct from propagating to subsequent phases.

In considering the same software artifact under varying inspection effort, some authors acknowledge a positive relationship between inspection time, the corresponding number of defects found, and the quality of the inspection artifact (Christenson and Huang 1988; Freimut et al. 2005; Laitenberger et al. 2002). Until all remaining defects are found, higher inspection times are expected to uncover a higher number of defects (Christenson and Huang 1988; Laitenberger et al. 2002). Given that there is a fixed number of defects in the artifact, a higher number of defects uncovered (and corrected) would result in fewer defects propagating to the next phase. Therefore, when more defects are uncovered and corrected, and fewer defects escape into the subsequent phase, the quality of the software product is expected to be enhanced (Freimut et al. 2005; Kemerer and Paulk 2009).

Nonetheless, in a real-world scenario, the total number of defects in the artifact being inspected is unknown. This assertion has led many to recognize a positive relationship between the number of defects uncovered in a given phase and the potential number of defects that escape to the subsequent phase. This positive relationship is evident in the model Christenson

and Huang (1988) developed to estimate the number of defects remaining in the code after inspection. On the other hand, Yu et al. (1988) found a positive relationship between the number of defects found in a given phase and the number of defects found in the previous phase. This further highlights a positive relationship between the number of defects found in one phase and the number of defects escaping into the next phase. These findings are consistent with Myers (1979) who observes that the "probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section" (p. 151).

In this paper, I prescribe to the view that number defects uncovered during the inspection of artifacts of a given project provides insights into the project's quality (or lack thereof) and the level of defect propagation into the subsequent phase. To supplement evidence already presented by others (e.g., Christenson and Huang 1988; Kitchenham and Pfleeger 1996; Myers 1979; Yu et al. 1988), I provide justification for this view based on the scenario illustrated in Fig. 1. Assume that an average quality artifact such as an initial requirement specification prior to subjecting it to inspection has i number of defects, $d_1,\ldots,d_i$ (Fig. 1a).

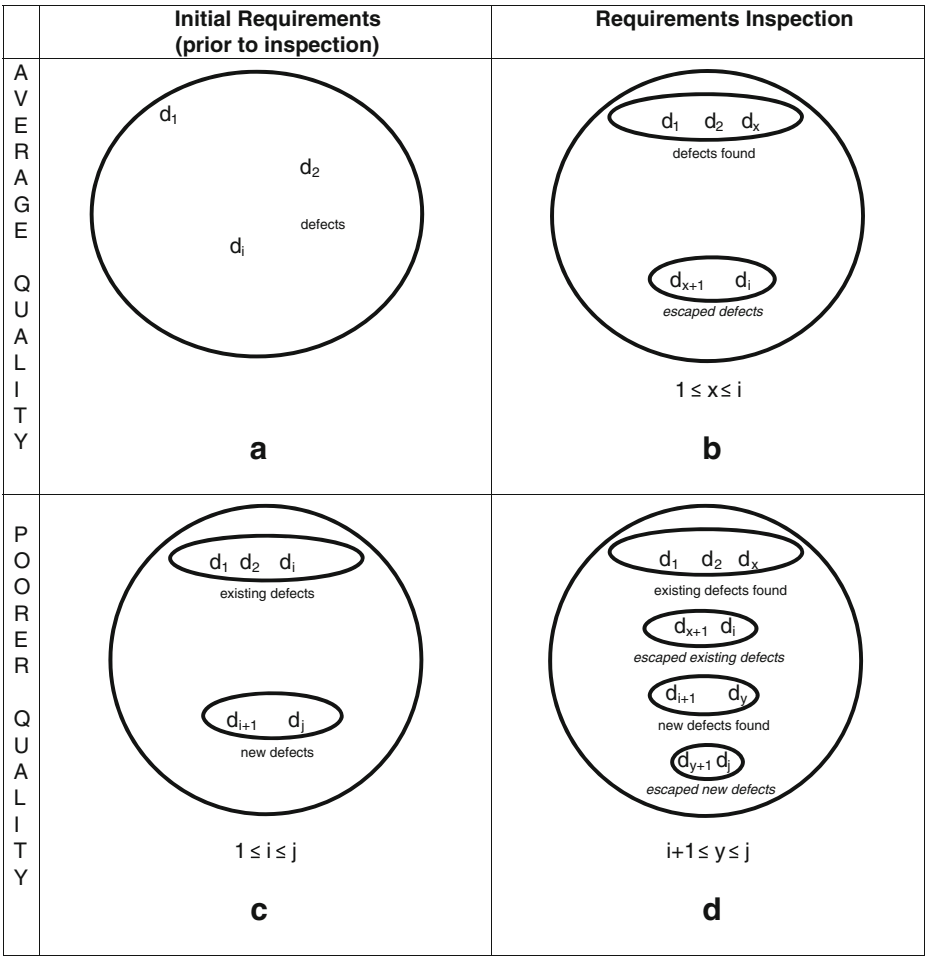



**Fig. 1** Defect propagation scenario

Inspection of such an artifact might result in uncovering x number of defects where $1 \leq x \leq i$ and (i-x) defects escaping into the subsequent phase (Fig. 1b). Now consider an artifact with a similar profile (e.g., number of pages) but of relatively poorer quality due to compromised process standards, suboptimal user involvement, or some other shortcoming. Such an artifact is likely to generate new defects $d_{i+1},\ldots,d_j$ where $1 \leq i \leq j$ (Fig. 1c). Assuming that the time required to log additional defects is relatively minor when compared to the time allotted to prepare and review the artifact for inspection (i.e., inspection time), a similar inspection effort of this relatively poorer quality artifact would result in uncovering new defects $d_{i+1},\ldots,d_y$ where $(i+1) \leq y \leq j$ and as a result, additional defects $d_{y+1},\ldots,d_j$ escaping into the subsequent phase (Fig. 1d). More importantly, the defects that propagate to the subsequent phase have an exponential effect on the quality of the artifacts in that phase (e.g., design, code). For example, each requirement gets mapped to one or more design features and when defects propagate from the requirements specification to design documents, such hierarchical dependencies have even a greater impact on the quality of the design documents.

As Kitchenham and Pfleeger (1996) and others (e.g., Christenson and Huang 1988; Myers 1979; Yu et al. 1988) observed, a comparison of the number of defects uncovered from the inspection of the two artifacts illustrated in 1b and 1d (Fig. 1) provides insights into the number of errors that would propagate to the next phase. Similarly, when defects are aggregated to the project-level, we could expect a project with a higher number of defects in one phase to have a larger number of defects escape into the subsequent phase. This line of reasoning further supports the view that the defect count from the previous phase offers an accurate gauge of the extent of defect propagation to the next phase. In essence, the propagation argument can be made from two perspectives. First, since the actual number of defects are not known with certainty, higher the number of defects found in a given phase, higher the number of defects not found, and therefore escaping into the subsequent phase (Christenson and Huang 1988; Myers 1979; Yu et al. 1988). Second, since some fixes could result in newer defects (Ebert and Jones 2009), higher the number of defects found in a given phase, higher the number of new defects introduced as a result of fixes, and therefore escaping into the subsequent phase.

While anecdotal evidence on defect propagation is abound, to author's knowledge, only a few researchers including Kelly et al. (1992), Raz and Barad (2004), Yu et al. (1988), Briand et al. (1997), and Raz and Yaung (1994) employed inspection in an empirical study to investigate the impact of defect propagation during one or multiple phases involving requirements, design, and coding. Nonetheless, Kelly et al.'s research conducted back in 1992 considers 5 projects totaling 203 inspections while Raz and Barad (2004) used 77 inspections from a single software project. On the other hand, Yu et al.'s (1988) research conducted back in 1988 considers two software products involving 32 inspections of sub-systems. They found a strong correlation between the number of defects found in the previous phase and current phase. Kelly et al. (1992) found highest defect densities in requirements inspections although defect densities decreased dramatically when defects are corrected at their origin and not allowed to propagate into subsequent phases. Briand et al. (1997) used 40 inspections from a single project. Finally, Raz and Yaung (1994) used neural networks to examine defect propagation in code segments of a single project. These authors employed a neural network approach to code inspection based on the back propagation model for identifying inspections with defect escapes into a subsequent inspection. Although these scholars have offered useful insights into defect propagation, with recent advances in software processes, it is questionable whether empirical results from limited data with 1 or few projects that are older than a decade or more that are not aggregated to the project-level during requirements, design, and coding

phases will still have the same effect today. In undertaking this empirical study, I use recent inspection data that are aggregated at the project-level from a much larger number of projects in a large software development firm collected during requirements, design, and coding phases.

## 3 Research Model

The research model presented in Fig. 2 illustrates the set of hypotheses developed to investigate defect propagation from requirements to design to coding phases. Following arguments presented in the earlier section, the research model is based on the premise that when the inspection of related artifacts for a particular project results in a large number of defects, proportionally a large number of defects escape into the next phase. Consequently, the inspection of the related artifacts of such a project in the subsequent phase results in a larger number of defects relative to other projects because (1) the escaped defects from the previous phase adds to the potential number of defects that could be uncovered during inspection of the subsequent phase; (2) the potential number of multiple new defects that could emerge in the subsequent phase as a result of each defect propagating from the previous phase; and (3) fixing a higher number of defects in the previous phase could result in a higher number of new defects (vis-à-vis fixing a fewer number of defects). Hence, I posit that the number of defects uncovered in the previous phase of a project has a positive relationship on the number of defects uncovered in the subsequent phase due to defect propagation. Hypotheses 1 and 2 aim to demonstrate the effects of defect propagation from requirements to design ($H_1$) and from design to coding ($H_2$), respectively.

H1: The number of requirements defects positively impacts the number of design defects.
H2: The number of design defects positively impacts the number of coding defects.

## 4 Method

Data was collected from SwDevCo (a pseudonym), a world leader in software development with average annual net revenue of approximately US$28 billion. SwDevCo employs inspections as part of its overall quality assurance program. Inspections are conducted throughout the CMMI level-5 compliant proprietary development process based on the traditional waterfall lifecycle. Requirements, design, and coding checklists along with the artifact being inspected (requirements specification, design document, programming code) and any related artifact

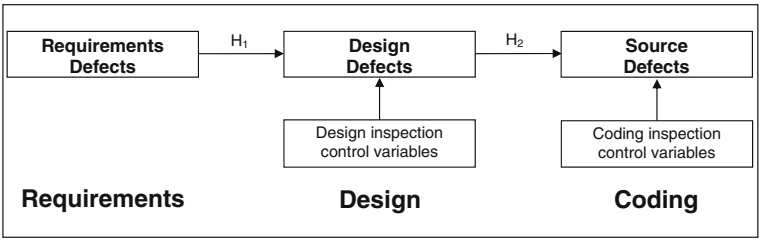



**Fig. 2** Research model

from the previous phase are used during the inspection. The dataset used in this study consists of inspections for requirements, design, and coding conducted over a 4½year period.[2] Since the focus of this research is on defect propagation, only those projects that had at least 1 inspection in requirements and design phases, and design and coding phases were considered – see Table 1 for the profile of inspections and projects. Unit of analysis in this research is the project (application). Hence, data collected, namely number of inspections,[3] number of reviewers,[4] inspection time,[5] and number of defects found were aggregated at the project-level for each project during requirements, design, and coding phases. While the number of defects represents the key construct under study in examining defect propagation, others (number of inspections, number of reviewers, inspection time, and complexity[6]) are used as control variables. Regression is used as the method of analysis to explore the relationship between defects uncovered in the previous phase and defects uncovered in the subsequent phase, highlighting the level of propagation. Regression allows for the isolation and the comparison of the impact of a set of independent variables (including the focal variable) on a dependent variable.

A closer look at Table 1 reveals several interesting observations. The number of design and coding inspections are more than 5-fold higher compared to requirements inspections. The average number of reviewers per inspection, inspection time (hours) and number of defects uncovered per inspection are consistently highest for requirements while lowest for coding. At the project-level, the average inspections per project is lowest for requirements while highest at more than 6-fold for coding. The average number of reviewers, inspection time and defects uncovered per project are all lowest for requirements. Finally, the average number of defects uncovered per project during design is considerably higher than the corresponding number for the coding phase.

The size of the artifact is normally measured in pages. However, some of the inspections did not contain the number of pages of the inspected artifact. In their analysis of industry data, Briand et al. (1997) observed that the size of the inspection artifact is rarely available. Furthermore, Laitenberger et al. (1999, p. 311) found that "the number of detected defects is primarily determined by the preparation effort of reviewers rather than the size of the reviewed artifact" and that "the size of the reviewed artifact has only limited influence on review effort." It was not possible to triangulate missing size from other variables such as number of reviewers or inspection time. Hence, I also excluded size from the analysis.

The dataset from SwDevCo does not make a distinction between major and minor defects in the inspection artifact. In their analysis, Miller and Yin (2004) made no distinction on defect severity due to unavailability of such data. One the other hand, Briand et al. (2004) combined major and minor defects together suggesting the classification into major and minor defects is unreliable. In this study, defects correspond to all defects without reference to their severity. Raw data grouped the complexity of each inspection document into low, medium and high. These groups were assigned numerical values 1 (low), 2 (medium), and 3 (high). This grouping was done by the creator(s) of

[2] Start of this period coincided with SwDevCo's initiative to systematically collect inspection data across requirements, design, and coding phases for projects (applications).

[3] Aggregated "number of inspections" corresponds to sum of all inspections conducted for a particular project during a given phase (e.g., requirement analysis).

[4] Aggregated "number of reviewers" corresponds to sum of reviewers of all inspections for a particular project (during a given phase), not distinct number of reviewers for that project.

[5] Aggregated "inspection time" include both preparation and review time for a particular project during a given phase.

[6] Complexity refers to the complexity of the inspection document that is aggregated to the project level.

**Table 1** Profile of inspections and projects

| | Requirements | Design | Coding |
|---|---|---|---|
| Inspections in total | 2062 | 11533 | 12563 |
| Inspections used in analysis[a] | 1944 | 11262 | 11945 |
| Number of reviewers | | | |
| Average (per inspection) | 3.4 | 2.1 | 1.5 |
| Median | 1 | 1 | 1 |
| Min-max | 1–50 | 1–50 | 1–35 |
| Inspection time (hours) | | | |
| Average (per inspection) | 3.6 | 2.1 | 1.2 |
| Median | 1.25 | 1 | 0.5 |
| Min-max | 0.25–205 | 0.25–90 | 0.25–175 |
| Defects | | | |
| Average (per inspection) | 11.1 | 6.8 | 2.9 |
| Median | 3 | 3 | 0 |
| Min-max | 0–1265 | 0–862 | 0–597 |
| Projects[b] | | | |
| Inspections | 242 | 311 | 248 |
| Average (per project) | 8.0 | 36.2 | 48.2 |
| Median | 2.0 | 8.0 | 7.0 |
| Min-max | 1–46 | 1–242 | 1–245 |
| Number of reviewers | | | |
| Average (per project) | 27.3 | 76.0 | 72.2 |
| Median | 10.0 | 17.0 | 16.5 |
| Min-max | 1–2329 | 1–3418 | 1–1496 |
| Inspection time (hours) | | | |
| Average (per project) | 28.9 | 76.0 | 57.8 |
| Median | 10.1 | 18.0 | 14.4 |
| Min-max | 0.25–2037.5 | 0.25–3019.25 | 0.25–994.5 |
| Defects | | | |
| Average (per project) | 89.2 | 246.2 | 138.7 |
| Median | 23.5 | 43.0 | 29.0 |
| Min-max | 0–4459 | 0–4999 | 0–4524 |

[a] Requirements inspections used in the analysis corresponds to those projects that had at least one inspection in requirements and design phases. Coding inspections used corresponds to those projects that had at least one inspection in design and coding phases. Design inspections used corresponds to those projects that had at least one inspection common with requirement or coding phases. Remaining data in the table correspond to those inspections/projects used in the analysis

[b] A total of 201 projects had at least 1 inspection in requirements and design phases while a total of 216 projects had at least 1 inspection in design and coding phases. A total of 149 projects had at least 1 inspection in all three phases

those documents (e.g., requirements specification, design document, code segment). In order to get a more accurate estimate of the project-level complexity, the complexities of inspected documents are weighted against inspection times during aggregation. Inspection time was used as weights in aggregation since size information was not available. Weighting and aggregation of complexity to the project level was done by the author.

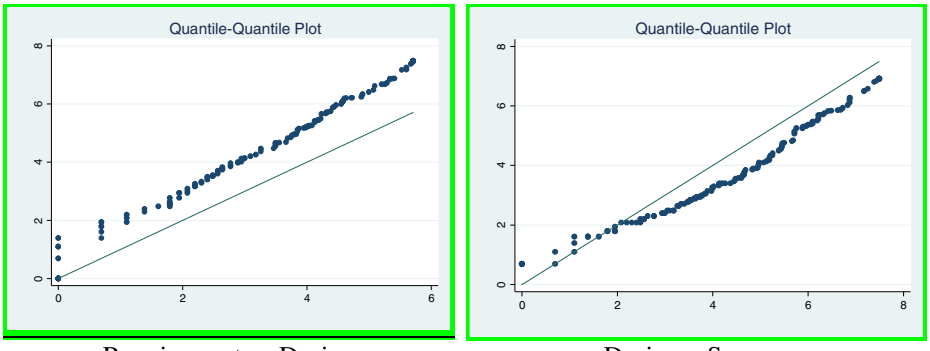
## 5 Analysis

Initial analysis indicated that the number of defects, number of inspections, number of reviewers, and inspection time are distributed in a non-normal fashion. Therefore, logarithmic transformation was applied to these variables to reduce skewness (and satisfy assumption in OLS analysis described later). Banker and Kemerer (1989, p. 1200) argue that "productivity increases on progressively larger projects may also come from the greater use of specialized personnel and tools," thereby signaling the impact of economies of scale in software development. As the number of defects uncovered is likely to be influenced by economies of scale with respect to inspection time spent by specialized personnel, logarithmic transformation is justified. Q-Q plots using transformed defect data demonstrated normality (see Fig. 3). While non-linear models have been proposed in the analysis of inspection data, many scholars have used log transformation in regression models to obtain linear approximations (e.g., Briand et al. 1997; Raz and Barad 2004). Hence, regression models to test the above hypotheses are identified as follows:

$$\begin{aligned} \ln(\textit{Design defects}) = {} & \alpha_0 + \alpha_1 * \ln(\textit{Requirement defects}) + \\ & \alpha_2 * \ln(\textit{Design inspections}) + \\ & \alpha_3 * \ln(\textit{Design reviewers}) + \\ & \alpha_4 * \ln(\textit{Design inspection time}) + \\ & \alpha_5 * \textit{Weighted design complexity} + \varepsilon_1 \end{aligned} \tag{1}$$

$$\begin{aligned} \ln(\textit{Coding defects}) = {} & \beta_0 + \beta_1 * \ln(\textit{Design defects}) + \\ & \beta_2 * \ln(\textit{Coding inspections}) + \\ & \beta_3 * \ln(\textit{Coding reviewers}) + \\ & \beta_4 * \ln(\textit{Coding inspection time}) + \\ & \beta_5 * \textit{Weighted coding complexity} + \varepsilon_2 \end{aligned} \tag{2}$$

Ordinary least squares (OLS) regression was initially used to test the models. There were several outliers; hence, after aggregating the inspection data to the project-level, they were




Requirements x Design Design x Source

**Fig. 3** Q-Q plots

winsorized at 90 %[7] to minimize effects of outliers. Results showed significant multicollinearity among independent variables with number of reviewers showing the highest variance inflation factor[8] (VIF) in excess of 8 and mean VIF for all independent variables in excess of 4 for regression Eq. (1). Similar concern for multicollinearity emerged for regression Eq. (2) with number of reviewers showing the highest VIF in excess of 8 and mean VIF for all independent variables in excess of 4. Following common practice, number of reviewers was dropped from further analysis to reduce the risk of multi-colinearity. For the resulting regression, the maximum VIF was less than 4 while mean VIF for Eqs. (1) and (2) were 2.5 and 2.35, respectively.

Defect propagation is tested by examining the relationship from requirements defects to design defects ($H_1$) and from design defects to coding defects ($H_2$). The OLS analysis supports both propagation hypotheses (requirements to design: $\alpha_1$=0.164, $t$=4.12, $p$<0.001; design to coding: $\beta_1$=0.164, $t$=4.91, $p$<0.001) (see Tables 2 and 3). The positive relationship between number of defects in the previous phase and number of defects in the current phase offer evidence for defect propagation across phases.

Since data was collected from a single organization, it is possible that the error terms are correlated due to a common effect. Seemingly unrelated regression (SUR) corrects for correlated error terms (Davidson and MacKinnon 1993; Greene 2002). In SUR, a linear regression model consisting of a set of regression equations is generated. Each of these regression equations has its own dependent variable and a set of exogenous variables. While the equations appear to be unrelated, they are related through the correlation in the errors. On the other hand, two-stage least squares (2SLS) account for possible endogeneity resulting from correlation between the variable and the error term (James and Singh 1978; Schmitt and Bedeian 1982). In the first stage, 2SLS uses an instrument variable to create a new variable (by replacing the variable that deemed to be problematic). This new variable is expected to be correlated with the problematic variable but not with the error term. In the second stage, estimated values from the first stage are used in place of the actual values of the problematic variables to compute the model.

Hence, three-stage least squares (3SLS) that combines SUR and 2SLS (Zellner and Theil 1962; Greene 1993; Maddala 2008) was run. 3SLS builds on 2SLS to account for correlations between equations in the same way SUR does using a 3-step process: (1) conduct regressions to get predicted values; (2) generate residuals to estimate the correlation across equations; and (3) model estimation. This 3SLS analysis was conducted using the subset of projects that had at least 1 inspection in requirements, design, and coding phases. A total of 149 projects satisfied this criteria. As shown in Table 4, 3SLS also provide strong support $H_1$ and $H_2$ (requirements to design: $\alpha_1$=0.134, $t$=3.10, $p$<0.01; design to coding: $\beta_1$=0.135, $t$=3.23, $p$<0.01). In addition, 3SLS results show that all control variables – number of inspections, inspection time, and weighted complexity – impact the number of defects uncovered in the current phase.

Furthermore, 3SLS results revealed that variables included in the analysis explained 83 % and 76 % of the variance on the number of defects uncovered in design and coding phases, respectively. Results also showed that inspection times and number of inspections for projects have a greater impact on the number of defects uncovered earlier in the lifecycle than later (with coefficients 0.665 in design versus 0.527 in coding for inspection times and 0.435 in

[7] In Winsorizing at 90, 0–5 % percentile values are set to 5 % while 95–100 % percentile values are set to 95 %.
[8] VIF shows the severity of multicollinearity in OLS regression. VIF offers a measure of how much the variance of an estimated regression coefficient is increased because of multicollinearity.

**Table 2** OLS regression analysis for design defects (1)

| Variables | Coefficient | Std. error | *t* | *p* |
|---|---|---|---|---|
| Intercept ($\alpha_0$) | 0.010 | 0.306 | 0.03 | n.s |
| *ln* Requirements defects ($\alpha_1$) | 0.164 | 0.040 | 4.12 | $p<0.001$ |
| *ln* Design inspections ($\alpha_2$) | 0.400 | 0.070 | 5.73 | $p<0.001$ |
| *ln* Design inspection time ($\alpha_4$) | 0.684 | 0.079 | 8.67 | $p<0.001$ |
| Weighted design complexity ($\alpha_5$) | 0.224 | 0.155 | 1.44 | n.s. |
| N | | | | 201 |
| Model | | | $F_{4,196}$ | 207.35 |
| | | | $R^2$ | 0.81 |
| | | | | $p<0.001$ |

design versus 0.380 in coding for number of inspections). That is, each inspection time (hour) and each inspection for a project resulted in uncovering a greater number of defects earlier in the lifecycle. On the other hand, complexity has a greater impact on the number of defects later in the lifecycle than earlier (with coefficients 0.458 in coding versus 0.353 in design). Nonetheless, there is little difference between propagation from requirements to design, and from design to coding (with coefficients 0.134 in design versus 0.135 in coding).

## 6 Discussion

I hypothesized that at the project-level, there is a positive relationship between defects uncovered in the previous phase and defects uncovered in the current phase, thereby providing evidence of defect propagation from requirements to design to coding. Data from a large multi-national software development firm was used to test the two hypotheses. Results from both OLS and 3SLS analyses provide support for this positive relationship.

Furthermore, findings revealed that inspections are effective in uncovering defects throughout the software development lifecycle. While most of the inspection studies report their effectiveness for one phase of the development process and at the individual inspection level, results from this study where inspections are aggregated to the project-level show their effectiveness in

**Table 3** OLS regression analysis for coding defects (2)

| Variables | Coefficient | Std. error | *t* | *p* |
|---|---|---|---|---|
| Intercept ($\beta_0$) | −0.395 | 0.276 | −1.43 | n.s. |
| *ln* Design defects ($\beta_1$) | 0.164 | 0.033 | 4.91 | $p<0.001$ |
| *ln* Coding inspections ($\beta_2$) | 0.414 | 0.070 | 5.95 | $p<0.001$ |
| *ln* Coding inspection time ($\beta_4$) | 0.529 | 0.075 | 7.01 | $p<0.001$ |
| Weighted coding complexity ($\beta_5$) | 0.440 | 0.135 | 3.26 | $p<0.01$ |
| N | | | | 216 |
| Model | | | $F_{4,211}$ | 172.09 |
| | | | $R^2$ | 0.77 |
| | | | | $p<0.001$ |

**Table 4** Three stage least squares (3SLS) analysis for (1) and (2)

| Design defects (1) | | | | |
|---|---|---|---|---|
| Variables | Coefficient | Std. error | *t* | *p* |
| Intercept ($\alpha_0$) | −0.278 | 0.349 | −0.80 | n.s. |
| *ln* Requirements defects ($\alpha_1$) | 0.134 | 0.043 | 3.10 | $p<0.01$ |
| *ln* Design inspections ($\alpha_2$) | 0.435 | 0.076 | 5.72 | $p<0.001$ |
| *ln* Design inspection time ($\alpha_4$) | 0.665 | 0.085 | 7.84 | $p<0.001$ |
| Weighted design complexity ($\alpha_5$) | 0.353 | 0.175 | 2.02 | $p<0.05$ |
| Model | | | $X^2$ | 712.83 |
| | | | $R^2$ | 0.83 |
| | | | | $p<0.001$ |
| Coding defects (2) | | | | |
| Variables | Coefficient | Std. error | *t* | *p* |
| Intercept ($\beta_0$) | −0.307 | 0.356 | −0.86 | n.s. |
| *ln* Design defects ($\beta_1$) | 0.135 | 0.042 | 3.23 | $p<0.01$ |
| *ln* Coding inspections ($\beta_2$) | 0.380 | 0.080 | 4.74 | $p<0.001$ |
| *ln* Coding inspection time ($\beta_4$) | 0.527 | 0.092 | 5.74 | $p<0.001$ |
| Weighted coding complexity ($\beta_5$) | 0.458 | 0.171 | 2.68 | $p<0.01$ |
| Model | | | $X^2$ | 479.92 |
| | | | $R^2$ | 0.76 |
| | | | | $p<0.001$ |

Notes: $N=149$

requirements, design, and coding phases. It also revealed that the number of defects uncovered for every hour of inspection decreases as the software product progresses from design to coding. By conducting an empirical study using aggregated inspection data from a large multinational development company for three key phases of the software development lifecycle, this research answers calls by scholars such as Mantyla and Lassenius (2009, p. 430) who argue that "despite the fact that peer reviews have been extensively studied, knowledge of their benefits is still inadequate." Results demonstrate that there is a positive relationship between inspection time and number of defects uncovered, hence mitigating defect propagation.

Scholars have proposed various measures to assess effectiveness and efficiency of inspections. For example, one such measure for inspection effectiveness is defect density which calculates the number of defects uncovered per unit size (e.g., per page) of the inspection document (Christenson and Huang 1988; Ebenau 1994; Kelly et al. 1992). Efficiency is often calculated in terms of the number of defects uncovered per unit time (e.g., per hour) (Briand et al. 1998; Freimut et al. 2005; Runeson et al. 2006; Winkler et al. 2010). Since inspection time and the number of defects uncovered were available at the project-level, a post-hoc analysis was conducted to examine inspection efficiency in requirements, design, and coding phases. While inspection efficiency has been examined in the literature (e.g., Briand et al. 1998; Runeson et al. 2006), to the author's knowledge, no one has examined inspection efficiency empirically at the project-level both within and across requirements, design, and coding phases. This post-hoc analysis was conducted using the same subset of 149 projects that contained at least 1 inspection in requirements, design, and coding phases.

The requirements, design, and coding data subsets were first divided into five categories based on inspection time (lowest 20 %, lower 20 %, middle 20 %, higher 20 %, and highest

20 %). Table 5 shows the efficiency ratios for the early lifecycle phases based on the five categories. Results show that the average efficiency ratio in terms of the number of defects found per each hour invested on inspection decreases from requirements to design to coding. For requirements and design, middle 20 % achieves the greatest efficiency while for coding, higher 20 % achieves the greatest efficiency. Nonetheless, after reaching the respective optimal, the efficiency ratio decreases with higher inspection times.

To gain further insights into why efficiency ratio reaches an optimal and then decreases, the number of defects uncovered for each category were examined. As shown in Table 6, results are consistent with past research that has demonstrated that as the inspection time increases, the number of defects uncovered increases (Christenson and Huang 1988; Freimut et al. 2005; Laitenberger et al. 2002). Nonetheless, the increase in the number of defects uncovered beyond the optimal point is relatively smaller compared to the additional effort required in terms of inspection time, thereby reducing the inspection efficiency.

Further insights for this non-linear relationship between raw inspection times and efficiency ratios were sought by examining corresponding scatter plots for requirements, design, and coding phases. Figures 4, 5, and 6 shows the corresponding polynomial functions (second degree) for the relationship between inspection times and efficiency ratio[9] for requirements, design, and coding phases. The corresponding R-squared values for requirements, design, and coding phases are 30 %, 40 %, and 40 %, respectively. These results give further credence to earlier findings that efficiency reaches an optimal and then drops off at higher inspection times beyond that point.

While the optimal point in the efficiency ratio for each of requirements, design, and coding phases corresponds to the entire set of 149 projects from SwDevCo, it would be useful to benchmark suitable inspection times for projects based on parameters such as size, defects found in the prior phase, and complexity. Given the greater emphasis on the cost of inspection (Briand et al. 1998; Briand et al. 2000; Porter et al. 1997), such benchmarks and the focus on inspection efficiency will go a long way in justifying the return on investment from software inspections. One approach for such an effort is to group these variables – size, prior defects, and complexity – into several categories[10] and then set benchmarks for inspection times that could potentially achieve the optimal efficiency ratio. Any such efforts need to be continually validated by collecting data and adjusting these benchmarks to realize optimal efficiency. To date, Briand et al. (1998) has offered one of the most extensive guidelines for benchmarking efficiency for design and coding inspections. More industry-specific and organization-specific benchmarks need to be developed to compare efficiencies within and across industries and organizations for requirements, design, and coding phases.

Additional analyses were conducted to examine whether there is a relationship between inspection efficiencies between the previous phase and the subsequent phase. Using the same 149 project subset, inspection efficiencies in requirements, design and coding phases were grouped into three categories: lowest, average, and highest. The corresponding lower and upper bounds for efficiency ratios, project count, and the overall mean, median, and standard deviation of efficiency ratios for the requirements, design, and coding phases are shown in Table 7. While all had the same lower bound of zero, higher bound for efficiency ratios varied from 24, 18, and 22 for requirements, design, and coding phases, respectively. Mean efficiency

[9] Intercept in the polynomial function is set to 0 since a hypothetical inspection time close to zero hours results in zero defects and hence, zero efficiency.

[10] For example, highest 20 %, higher 20 %, middle 20 %, etc.

**Table 5** Efficiency ratio (defects per hour) by inspection time

| Inspection time | Requirements | Design | Coding |
|---|---|---|---|
| Lowest 20 % | 3.9 | 2.7 | 3.0 |
| Lower 20 % | 3.8 | 2.9 | 2.5 |
| Middle 20 % | 4.8 | 3.5 | 2.8 |
| Higher 20 % | 3.3 | 3.3 | 3.2 |
| Highest 20 % | 2.0 | 3.0 | 1.9 |
| Average | 3.6 | 3.1 | 2.7 |

ratios were highest for requirements (3.6) and lowest for coding (2.7) while because of some outliers, median ratios for all three were considerably lower than their respective means.

Next, the lowest third of the efficiency ratios for requirements and design was mapped to the corresponding efficiency ratios in design and coding, respectively. As shown in Table 8, of those requirements projects with lowest efficiency ratios, 49 % of them remained in the lowest third efficiency group in the subsequent design phase. Of those design projects with lowest efficiency ratios, 61 % of them remained in the lowest third of efficiency group in the coding phase. Similar observations were made for the average and highest requirements to design and design to coding mapping. Forty-three (43 %) percent of requirements projects with average efficiency ratios remained in the average efficiency group in the subsequent design phase while 44 % of design projects with average efficiency ratios remained in the average efficiency group in the coding phase. Fifty-eight (58 %) percent of requirements projects with highest efficiency ratios remained in the highest efficiency group in the subsequent design phase while 50 % of design projects with highest efficiency ratios remained in the highest efficiency group in the coding phase.

To further investigate the changes to efficiency from one phase to another, scatter plots for requirements to design, and design to coding each of requirement, design, and coding 2-tuple were graphed. As shown in Figs. 7 and 8, there is a positive, relatively high coefficient for the efficiency ratio plots for adjacent phases. This demonstrates that efficiencies between requirements and design, and design and coding tend to remain stable. However, the coefficient in the efficiency ratio scatter plot between requirements and coding is close to zero revealing a non-existent relationship between inspection efficiencies for these two phases. The examination of correlation coefficients confirmed similar results. The corresponding correlation coefficients between requirements and design, and design and coding were 0.309 ($p<0.001$) and 0.261 ($p<0.01$), respectively. The corresponding R-squared values for requirements to design, and design to coding were 10 % and 7 %, respectively. There was a weak, statistically insignificant correlation between requirements and coding (Fig. 9).

**Table 6** Defects uncovered by inspection time

| Inspection time | Requirements | Design | Coding |
|---|---|---|---|
| Lowest 20 % | 4.6 | 11.4 | 10.0 |
| Lower 20 % | 19.8 | 46.6 | 22.6 |
| Middle 20 % | 65.8 | 134.4 | 54.6 |
| Higher 20 % | 95.1 | 339.0 | 168.5 |
| Highest 20 % | 144.4 | 990.7 | 530.9 |

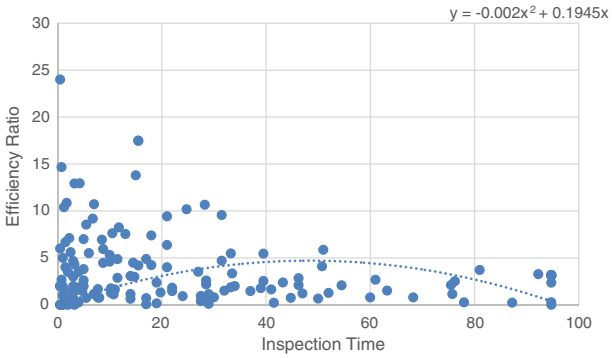



**Fig. 4** Requirements efficiency ratio

These results demonstrate that generally, inspection efficiencies remain stable between adjacent phases. Any drastic changes to the efficiency ratio from one phase to another must be examined with caution. Earlier I advocated benchmarking inspection times based on parameters such as size, prior defects, and complexity. In a similar fashion, benchmarking inspection efficiencies could help organizations track changes in efficiencies from requirements to design to coding, and assign/adjust inspection times as necessary during both the current and the subsequent phase. Such benchmarks are likely to be organization and phase specific, and are likely to stabilize over time, thus increasing their usefulness. Rather than using groups (e.g., lowest, average, highest), managers could easily use percentile scores to identify significant changes in inspection efficiency from one phase to another. For example, if a particular project with a 50 % percentile efficiency ratio in requirements results in 90 % percentile during design, more analysis is needed to determine whether this was due to poor inspection efficiency of the requirements inspection team, injection of a large number of defects during the design phase, exceptional efficiency of the design inspection team, or some other reason. Benchmarking inspection efficiencies and inspection times will help managers take corrective action and as a result, realize strategic organizational goals in implementing software inspection to achieve software quality.

## 7 Threats to Validity

Several threats to validity could be identified. First, using data from a single organization employing similar inspection practices always raises the question of generalizability of the

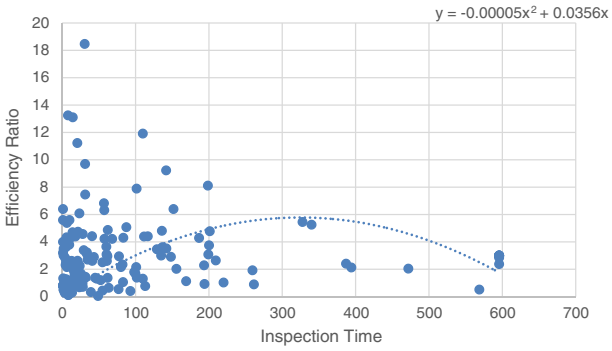



**Fig. 5** Design efficiency ratio

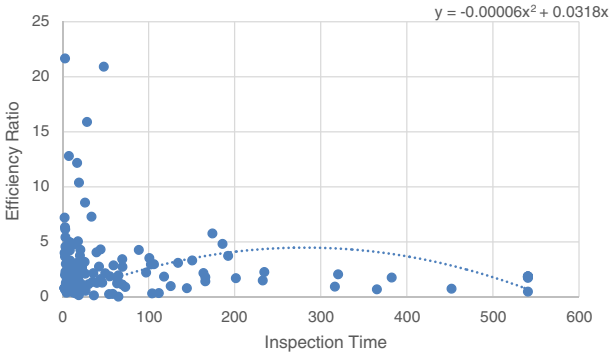



**Fig. 6** Coding efficiency ratio

findings on the extent of defect propagation and inspection efficiency during requirements, design, and coding phases. Organizations vary in terms of their development methods, processes, standards, personnel, among others. This threat to validity is to some extent mitigated by the fact that data came from a relatively larger number of inspections and the corresponding projects used in this study as compared to earlier research.

Second, the extent to which conclusions can be drawn on the relationship between the independent and dependent variable presents a threat to validity. One could argue that to demonstrate propagation, defects not found in the previous phase, thus escaping to the next phase have to be shown to correlate with defects found in the next phase. However, unlike in a controlled experiment with seeded defects, in an industrial setting, the total number of defects in the artifact is unknown. Hence, the number of defects escaping to the next phase is unknown. Instead, based on prior research and as well as with the use of a scenario (Fig. 1), I argued that defect propagation is evident from the relationship between defects in the prior phase (independent variable) and defects in the subsequent phase (dependent variable). This relationship manifests from the fact that the larger the number of defects uncovered in the prior phase, the larger the number of defects that were not

**Table 7** Efficiency ratio by group

| Group | Lower bound | Upper bound | Project count[a] | Overall mean, median Std. Dev. |
|---|---|---|---|---|
| Requirements | | | | |
| Lowest | 0.00 | 1.32 | 49 | |
| Average | 1.33 | 3.80 | 49 | 3.6, 2.1, 3.9 |
| Highest | 4.00 | 24.00 | 51 | |
| Design | | | | |
| Lowest | 0.00 | 1.42 | 49 | |
| Average | 1.45 | 3.39 | 50 | 3.1, 2.5, 2.8 |
| Highest | 3.46 | 18.47 | 50 | |
| Coding | | | | |
| Lowest | 0.00 | 1.30 | 48 | |
| Average | 1.33 | 2.74 | 51 | 2.7, 1.8, 3.2 |
| Highest | 2.75 | 21.65 | 50 | |

[a] Given 149 projects in total, lowest 49 efficiency ratios were assigned to lowest category while average and highest were assigned remaining 50 projects each. However, because of ties, in some instances, some groups had additional projects assigned to them

**Table 8** Efficiency ratio mapping across phases

| | Design efficiency ratio for lowest requirement efficiency ratio | Coding efficiency ratio for lowest design efficiency ratio |
|---|---|---|
| Lowest | 49 % | 61 % |
| Average | 33 % | 20 % |
| Highest | 18 % | 18 % |
| | Design efficiency ratio for average requirement efficiency ratio | Coding efficiency ratio for average design efficiency ratio |
| Lowest | 33 % | 24 % |
| Average | 43 % | 44 % |
| Highest | 24 % | 32 % |
| | Design efficiency ratio for highest requirement efficiency ratio | Coding efficiency ratio for highest design efficiency ratio |
| Lowest | 18 % | 12 % |
| Average | 26 % | 38 % |
| Highest | 58 % | 50 % |

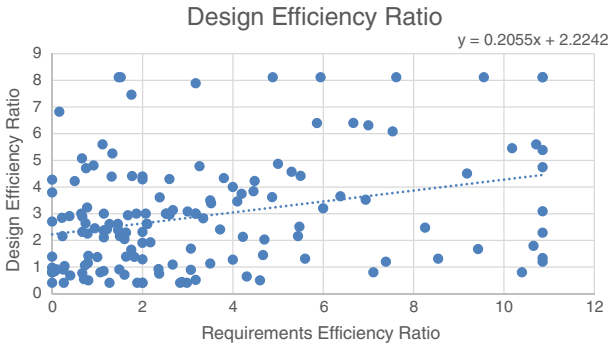



**Fig. 7** Requirements/design efficiency ratio

discovered, and hence escape to the subsequent phase. Also, the relationship between defects in previous and subsequent phase could stem from potentially a large number of new defects introduced as a result of fixing a large number of defects in the previous phase. This threat to validity is mitigated by controlling for project profile characteristics such as inspection time, complexity, and the number of inspections when running the regression analysis. While some scholars acknowledge the size of the inspection artifact as a good predictor of the number defects uncovered and the defects that escape (Raz and Yaung 1997; Raz and Barad 2004), others argue that effort (time) is a more reliable gauge of defects uncovered (e.g., Laitenberger et al. 1999). However, the number of defects uncovered due to variation in size of the artifact is likely to be captured collectively by the number of inspectors,[11] the number of inspections per project, and inspection time.

Third, a related threat to validity stems from the fact that because of hierarchical dependences, there is a 1-to-many relationship in defects from the previous phase to the next phase. As noted earlier, each requirement is mapped to one or more design features, and hence this phenomenon could explain why a higher number of defects in the previous phase result in a higher number of defects in the subsequent phase. Nonetheless, this threat to validity is mitigated as the company fixes all the defects found in the previous phase before preceding to the next phase.

Fourth, another threat to validity emerges from the appropriateness of the assumptions and statistical analysis conducted in the study. For example, logarithmic transformations were used and linear regression analysis was conducted. Some alternatives to logarithmic linear regression have been proposed such as multivariate adaptive regression splines that aim to approximate complex relationships via a series of linear regressions on different intervals of the independent variable (Briand et al. 2004). While these approaches show promise, their effectiveness over logarithmic linear regression has not been universally demonstrated.

Finally, as noted earlier, the dataset from SwDevCo did not differentiate between major and minor defects in the inspection artifact. When defect severity was not available, others have also grouped all into a single category (e.g., Briand et al. 2004; Miller and Yin 2004). Nonetheless, while further studies are needed to prove otherwise, it is reasonable to expect similar or even more significant[12] propagation effects when only major defects are considered.

[11] After initial analysis, the number of inspectors was dropped to reduce multicollinearity.

[12] A major defect in requirement could result in multiple design defects while a minor requirement defect (e.g., misspelled word in the requirement specification) might not even result in a design defect.

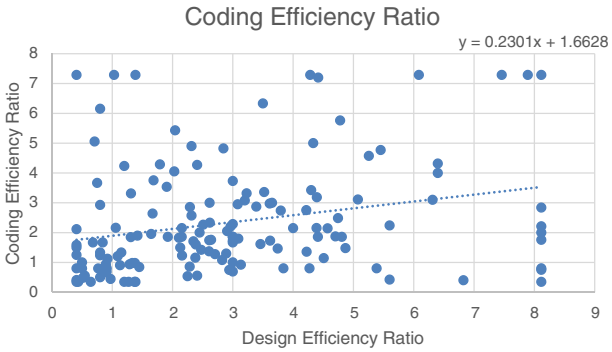



**Fig. 8** Design/coding efficiency ratio

## 8 Implications for Practice

This research has several important implications for practitioners. First, results show that when aggregated to the project-level, there is evidence of defect propagation across phases and that defect propagation impacts inspection of project's artifacts in subsequent phases. This is important because while an individual inspection might attest to quality characteristics of a particular artifact, inspections aggregated at the project-level attest to the quality characteristics of the entire software product. Second, this research shows that number of defects uncovered from inspection in the previous phase could be used to demonstrate extent of defect propagation into the next phase. This is helpful because it is a considerable challenge for many organizations to reliably record defects injected in a given phase and those injected in previous phases in assessing the extent of defect propagation. Moreover, tracking inspections from one phase to previous phases could enable mapping of defects across phases and as a result, subsequently uncover related defects as well as uncover previously escaped defects from one phase to next. Third, results show that inspection times and number of inspections have a greater impact on the number of defects uncovered earlier in the lifecycle than later. This highlights the greater return on inspection investment earlier in the lifecycle. Fourth, findings reveal that complexity has a greater impact on the number of defects uncovered later in the lifecycle than earlier (see Table 2, 3, and 4). While the relationship between complexity and the number of defects uncovered is not surprising, this finding provides a greater impetus for project managers to stay abreast of the complexity of the software product as it progresses from one phase to another.

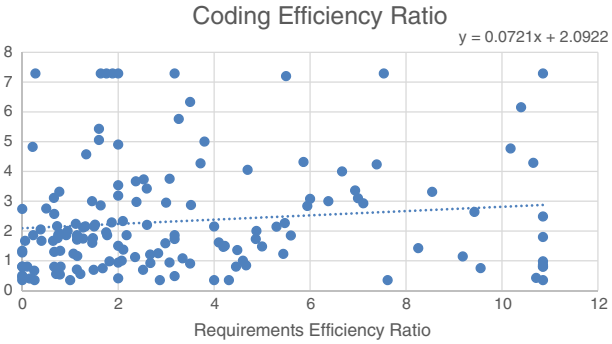



**Fig. 9** Requirements/coding efficiency ratio

This research also offers a greater understanding of inspection efficiencies at the project-level. The efficiency ratio at the individual-level provides insights into the return on investment on each hour of inspection for a particular artifact. A project-level analysis on the other hand gives a much more holistic view on efficiency that focuses on the entire software product. Results revealed that during each phase, as inspection time increases, efficiency reaches an optimal point and then decreases. In search of optimal efficiencies, project managers are wise to benchmark inspection times for projects in their organization based on parameters such as size, defects uncovered in the previous phase, and complexity.

Finally, results show that inspection efficiencies generally tend to remain stable across lifecycle phases. Benchmarking inspection efficiencies for each phase will help managers identify drastic changes to the efficiency ratio from one phase to another to assess whether any additional inspection time needs to be devoted to a given project or whether too much inspection time is already spent on it, thus take corrective action in the future. Developing benchmarks for inspection efficiencies and inspection times helps organizations to better manage inspections and as a result, realize greater return on inspection investment.

## 9 Conclusions and Future Work

In this paper, I show evidence of defect propagation and how defect propagation impacts inspection of project's artifacts in the subsequent phase. Using past research and a scenario as a guide, I show how the impact of aggregated defect counts from one phase to the next is useful in demonstrating the extent of defect propagation. The post-hoc analysis offered greater insights into inspection efficiency both within and across lifecycle phases. Benefits of benchmarking inspection times and efficiencies were identified.

This research offers several avenues for future research. More research is needed to account for possible variations across organizations by including inspection data from multiple sources as well as possible variations within an organization such as changes in software development processes over time and experience levels of analysts, coders, and inspectors. For example, ability to answer questions like, does defect propagation manifest independent of the process or more evident with certain processes such as waterfall lifecycle approach would provide more insights into the software development process and the corresponding quality of the resulting artifacts. With the popularity of incremental and agile development methods with multiple iterations of requirements, design, and coding phases, research examining defect propagation in these methodologies could provide greater insights. Furthermore, studies that examine how changes to requirements after initial requirement specification is formulated impact defect propagation would be useful. Inspecting an artifact with lower levels of abstraction (e.g., code) may in fact simplify the discovery of defects with varying severities due to propagation from an artifact in an earlier phase with a higher level of abstraction (e.g., design). Future research that examines whether inspections simplify defect detection on such a wider perspective based on severity levels could provide useful insights into defect propagation and software quality.[13] Further work is also needed to examine the effectiveness of

[13] I want to thank one of the reviewers for the observation that the inspection of an artifact with a lower level of abstraction downstream can simplify the identification of defects with all severities that have propagated from an artifact with a higher level of abstraction upsteam in the software development process.

inspections in uncovering semantic defects (vis-à-vis syntactical defects) and hence, the impact of inspections in mitigating the propagation of defects into subsequent phases.

This study demonstrating evidence of defect propagation across multiple phases of the software development lifecycle offers the perfect stepping stone for undertaking further research. This paper opens the door for defect propagation-related research such as the study of propagation metrics and propagation rates across requirements, design, and coding phases. Extant research has already identified the importance of measures such as the actual probability of an error occurring in the final software product due to error propagation (e.g., Abdelmoez et al. 2004; Montagud et al. 2012). In focusing on defect propagation, Berling and Thelin (2003) introduced a "goodness" measure to assess whether the defects were found in the earliest possible phase by comparing when a defect was found with when it hypothetically could have been found. Future research should also include the testing phase to assess the extent and impact of defect propagation beyond the coding phase. As defect propagation in upstream phases significantly impact downstream activities such as implementation, testing, and maintenance, more research on this topic will enable us to better manage the overall software development process and ultimately, quality of the software.

**Acknowledgments** The author thanks Empirical Software Engineering Editors-in-Chief and the review team for guidance through the review process. This research was partly funded by grants from the Earl V. Snyder Innovation Management Center and Robert H. Brethen Operations Management Institute at the Whitman School of Management, Syracuse University.

## References

Abdelmoez W, Nassar DM, Shereschevsky M, Gradetsky N, Gunnalan R, Ammar H, Yu B, Mili A (2004) Error propagation in software architectures. In: Proceedings of the tenth international symposium on software metrics

Banker RD, Kemerer CF (1989) Scale economies in new software development. IEEE Trans Softw Eng 15(10): 416–429

Berling T, Thelin T (2003) An industrial case study of the verification and validation activities. In: Proceedings of the ninth international symposium on software metrics

Biffl S (2000) Using inspection data for defect estimation. IEEE Softw 29(5):385–397

Biffl S, Halling M (2003) Investigating the defect detection effectiveness and cost benefit of nominal inspection teams. IEEE Trans Softw Eng 29(5):385–397

Boehm BW, Abts C, Brown A, Chulani S, Clark B, Horowitz E, Madachy R, Reifer D, Steece B (2000) Software cost estimation with COCOMO II. Prentice-Hall, New Jersey

Briand LC, Laitenberger O, Wieczorek I (1997) Building resource and quality management models for software inspections. In: Proceedings of the international software consulting network

Briand L, El Emam K, Laitenberger O, Fussbroich T (1998) Using simulation to build inspection efficiency benchmarks for development projects. In: Proceedings of the twentieth international conference on software engineering, 340–349

Briand L, El Emam K, Freimut B, Laitenberger O (2000) A comprehensive evaluation of capture-recapture models for estimating software defect content. IEEE Trans Softw Eng 26(6):518–540

Briand L, Freimut B, Vollei F (2004) Using multiple adaptive regression splines to support decision. J Syst Softw 73:205–217

Christenson DA, Huang ST (1988) A code inspection model for software quality management and prediction. Proc IEEE Glob Telecommun Conf 1:468–472

Davidson R, MacKinnon JG (1993) Estimation and inference in econometrics. Oxford University Press, UK

Ebenau RG (1994) Predictive quality control with software inspection. CrossTalk 7(6):9–16

Ebert C, Jones C (2009) Embedded software: facts, figures, and future. Computer 42:42–52

Fagan ME (1986) Advances in software inspections. IEEE Trans Softw Eng 12(7):744–751

Fagan ME (2002) A History of Software Inspections. In: Broy M, Denert E (eds) Software design and management conference, software pioneers: contributions to software engineering. Springer, New York

Fenton NE, Neil M (1999) A critique of software defect prediction models. IEEE Trans Softw Eng 25(5):675–689
Freimut B, Briand L, Vollei F (2005) Determining inspection cost-effectiveness by combining project data and expert opinion. IEEE Trans Softw Eng 31(12):1074–1092
Greene WH (1993) Econometric analysis, 2nd edn. Prentice-Hall, New Jersey
Greene WH (2002) Econometric analysis, 5th edn. Prentice-Hall, New Jersey
Harter D, Slaughter SA (2000) Process maturity and software quality: a field study. In: Proceedings of the international conference on information systems, 407–411
Jacobs J, van Moll J, Kusters R, Trienekens J, Brombacher A (2007) Identification of factors that influence defect injection and detection in development of software intensive products. Inf Softw Technol 49:774–789
James LR, Singh BK (1978) An introduction to the logic, assumptions, and basic analytic procedures of two-stage least squares. Psychol Bull 85(5):1104–1122
Johnson PM (1998) Reengineering inspection. Commun ACM 41(2):49–52
Kelly JC, Sherif JS, Hops J (1992) An analysis of defect densities found during software inspections. J Syst Softw 17:111–117
Kemerer CF, Paulk MC (2009) The impact of design and code reviews on software quality: an empirical study based on PSP data. IEEE Trans Softw Eng 35(4):534–550
Kitchenham B, Pfleeger S (1996) Software quality: the elusive target. IEEE Softw 13(1):12–21
Laitenberger O (2001) Cost-effective detection of software defects through perspective-based inspections. Empir Softw Eng 6:81–84
Laitenberger O, Leszak M, Stoll D, El Emam K (1999) Quantitative modeling of software reviews in an industrial setting. In: Proceedings of the sixth IEEE symposium on software metrics
Laitenberger O, Emam K, Harbich T (2001) An internally replicated quasi-experimental comparison of checklist and perspective-based reading of code documents. IEEE Trans Softw Eng 27(5):387–421
Laitenberger O, Beil T, Schwinn T (2002) An industrial case study to examine a non-traditional inspection implementation for requirements specifications. Empir Softw Eng 7:345–374
Maddala GS (2008) Introduction to econometrics, 3rd edn. Wiley, New York
Mandala N, Carver JC, Nagappan N (2012) Application of Kusumoto cost-metric to evaluate the cost effectiveness of software inspections. In: Proceeding of the ACM-IEEE international symposium on empirical software engineering and measurement, 221–230
Mantyla MV, Lassenius C (2009) What types of defects are really discovered in code reviews. IEEE Trans Softw Eng 35(3):430–448
Miller J, Yin Z (2004) A cognitive-based mechanism for constructing software inspection teams. IEEE Trans Softw Eng 30(11):811–825
Miller J, Wood M, Roper M (1998) Further experiences with scenarios and checklists. Empir Softw Eng 3:37–64
Montagud S, Abrahao S, Insfran E (2012) A systematic review of quality attributes and measures for software product lines. Softw Qual J 20:425–486
Myers GJ (1979) The Art of Software Testing. Wiley, New York
O'Neill D (1997) Issues in software inspection. IEEE Softw 14(1):18–19
Porter AA, Siy HP, Toman CA, Votta LG (1997) An experiment to assess the cost-benefits of code inspections in large scale software development. IEEE Trans Softw Eng 23(6):329–346
Raz T, Barad M (2004) In-process control of design inspection effectiveness. Int J Qual Reliab Eng 20:17–30
Raz T, Yaung AT (1994) Inspection effectiveness in software development: a neural network approach. In: Proceedings of the conference of the centre for advances studies on collaborative research, 61–72
Raz T, Yaung AT (1997) Factors affecting design inspection effectiveness in software development. Inf Softw Technol 39:297–305
Rigby PC, German DM, Cowen L, Storey M (2014) Peer review on open source software projects: parameters, statistical models, and theory. ACM Trans Softw Eng Methodol 23(4):35–67
Runeson P, Andersson C, Thelin T, Andrews A, Berling T (2006) What do we know about defect detection methods? IEEE Softw 23(3):82–90
Sauer C, Jeffery D, Land L, Yetton P (2000) The effectiveness of software development technical reviews: a behaviorally motivated program of research. IEEE Trans Softw Eng 26(1):1–14
Schmitt N, Bedeian AG (1982) A comparison of LISREL and Two-stage least squares analysis of a hypothesized life-job satisfaction reciprocal relationship. J Appl Psychol 67(6):806–817
Strauss S, Ebenau R (1994) Software inspection process. McGraw-Hill, New York
Unterkalmsteiner M, Gorschek T, Moinul Islam AKM, Cheng CK, Permadi RB, Feldt R (2012) Evaluation and measurement of software process improvement—a systematic literature review. IEEE Trans Softw Eng 38(2):398–424
Vitharana P, Ramamurthy K (2003) Computer-mediated group support, anonymity, and the software inspection process: an empirical investigation. IEEE Trans Softw Eng 29(2):167–180
Wagner S (2006) A model and sensitivity analysis of the quality economics of defect-detection techniques. In: Proceedings of the ACM/SIGSOFT international symposium on software testing and analysis, 73–83

Westland J (2004) The cost behavior of software defects. Decis Support Syst 37(2):229–238
Winkler D, Biffl S, Faderl K (2010) Investigating the temporal behavior of defect detection in software inspection and inspection-based testing. In: Baber, M A Vierimaa, M Oivo M (eds) Product-focused software process improvement. Proceedings of the eleventh international conference on product-focused software process improvement, 17–31
Yu T, Shen VY, Dunsmore HE (1988) An analysis of several software defect models. IEEE Trans Softw Eng 14(9):1261–1270
Zellner A, Theil H (1962) Three-stage least squares: simultaneous estimation of simultaneous equations. Econometrica 30(1):54–78

**Prof. Vitharana** is an associate professor at the Whitman School of Management at Syracuse University. His research primarily focuses on reusable software and software quality. He received his Ph.D. from University of Wisconsin – Milwaukee. His research has been published in IEEE Transactions on Software Engineering, IEEE Transactions on Systems, Man, and Cybernetics, Journal of MIS, Information and Management, Marketing Science, among others. He also edited special issues in Information Technology and Management (journal) and edited a book titled "Service-Oriented Perspectives in Design Science Research" (Springer). He has been serving as an Associate Editor at Communications of AIS journal since 2010.