Abstract
Numerous studies have shown that a software project’s cost, schedule and defect density escalate as the rate of requirements change increases. Yet none of these studies have explored the effects of not making requirements changes in response to changes in user needs. This paper explains why a project incurs just as much, if not more, risk when requirements changes are suppressed.

1. Introduction
Although many papers have analyzed negative effects of requirements change, none have addressed positive effects of requirements change. This paper differentiates between requirement and need. Although the two terms are synonyms according to thesauri, we use requirement to indicate a documented, externally-observable characteristic of a desired system [1], and need to indicate the actual need of users, customers, market, etc. Requirements change encompasses
- requirements volatility, a term defined as a measure of the number of requirements changes (additions, deletions, and modifications) [2] divided by the number of requirements for a given period of time [3], and
- requirements creep, a term defined by Jones [4] as “frequent changes in requirements,” and by Carter, et al. [5] as changes that result “in exten-

AAnnnuanuall IEIEEEEE IIntnteerrnanattiioonanall CCoompmpuutteerr SSooffttwawarree aandnd AAppppliclaatitioonnss CCoonnfererenccce

Numerous studies have shown that a software project’s cost, schedule and defect density escalate as the rate of requirements change increases. Yet none of these studies have explored the effects of not making requirements changes in response to changes in user needs. This paper explains why a project incurs just as much, if not more, risk when requirements changes are suppressed.

1. Introduction
Although many papers have analyzed negative effects of requirements change, none have addressed positive effects of requirements change. This paper differentiates between requirement and need. Although the two terms are synonyms according to thesauri, we use requirement to indicate a documented, externally-observable characteristic of a desired system [1], and need to indicate the actual need of users, customers, market, etc. Requirements change encompasses
- requirements volatility, a term defined as a measure of the number of requirements changes (additions, deletions, and modifications) [2] divided by the number of requirements for a given period of time [3], and
- requirements creep, a term defined by Jones [4] as “frequent changes in requirements,” and by Carter, et al. [5] as changes that result “in exten-

Critical Needs Changing

<table>
<thead>
<tr>
<th>Requirements Changing</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>YYY: Req’ts changes made in response to changing needs</td>
</tr>
<tr>
<td>Yes</td>
<td>YYY: Req’ts changes made in response to changing needs</td>
</tr>
<tr>
<td>Yes</td>
<td>YYY: Req’ts changes made in response to changing needs</td>
</tr>
<tr>
<td>No</td>
<td>YNN: Req’ts changes not made in spite of changing needs</td>
</tr>
<tr>
<td>No</td>
<td>NNN: No changes made to req’ts because no real needs are changing</td>
</tr>
</tbody>
</table>

Figure 1. Kinds of Requirements Change
the desire for something new, (3) the situation in which
the system is used changes, and (4) customers use the
current version of the system and that will trigger the
desire for something else they’d like to have done. Nurm-
uliani, et al. [7] investigated how developers classify
requirements changes. Those who classified require-
ments changes by the reason for change (60% of the test
subjects) used a combination of these categories: (1)
changes to product strategy, (2) changes to environment,
(3) scope reduction, (4) design improvement, (5) reali-
zation that a requirement had been missing, (6) unclear
requirement statement, (7) realization that original
requirements were not testable, or (8) enhancement.

Carter, et al., [5] provide insight into how risk analysis
is essential for determining which requirements changes
should be accepted for inclusion in an iteration. Lam and
Shankararaman [8] provide many suggestions on how to
manage requirements change, including being more
aware of changing user and customer needs (as opposed
to documented requirements), performing risk analysis of
making changes (also see Antón and Potts [9], who make
an excellent case for performing a benefits analysis of
requirements change as well), identifying conflicts
among requirements and among requirements changes,
and performing overt prioritization and triage [10].

original requirements change by the time the project
completes. Yourdon [12] and Young [13] report that
projects became over-budget and/or late whenever
requirements changes exceed ½-1% per month. We have
no doubt that this is true, but seriously question the
alternative, namely to suppress requirements changes as
needs evolve (box YYN in Figure 1).

3. Perceived Negative Impact of Making
Requirements Changes

Most authors who discuss impact of requirements
change emphasize negative impacts of such change, e.g.

• “Requirements volatility (RV) is generally
considered an undesirable property” [14].
• “Changing requirements are recognized as a major
cause of project failure” [8].
• “Requirements volatility causes the software to have
a higher defect density” [15].
• “Change . . . is often seen as a menace to established
order” [16].
• “Requirements volatility is considered to be a major
risk to . . . complex software projects” [7].

This is logical when considering only the cost of
satisfying documented requirements. We know for
example that the cost to develop a system (C_D)
is proportional to the number and complexity of the
elements of the set of originally specified requirements
(R) and the number of and complexity of the elements of
the set of changes made to those requirements (∆R)
during the development process, C_D ~ R, ∆R. To see
how silly it is to focus just on the cost to develop, we can
minimize it quite easily: just minimize R. This may seem
ridiculous, but when we talk about how requirements
change causes problems on projects, aren’t we
suggesting that we find a way to minimize ∆R, which is
just as ridiculous?

4. Negative Impact of Not Making
Requirements Changes

It makes little sense to try to minimize C_D; after all, it
is trivial as shown previously. What we should be doing
is trying to minimize the cost to satisfy (C_S) the custom-
ners’/market’s needs. And, this cost is proportional to
the number of elements in and the complexity of the elements of the set of needs (N) and the number of elements in and the complexity of the elements of the set of
changes to those needs (ΔN), i.e., C_S ~ N, ΔN. But
more importantly, C_S is proportional to C_D and the
degree of discordance (δ) between R and N and ∆R and
ΔN, i.e., C_S ~ C_D, δ(R, N), δ(∆R, ΔN). Replacing C_D,
we get, C_S ~ R, ∆R, δ(R, N), δ(∆R, ΔN). Since we
cannot control N or ΔN, attempts to minimize ∆R are
moot. The only way to minimize C_S is to minimize
discordance of ∆R and ΔN regardless of the negative side
effect on C_D.

The preponderance of data indicates that the more
requirements change, the higher the cost, the longer the
development, and the higher the defect density. We will
use the term negative project success factors to capture
the concept of customer dissatisfaction, in general, and
higher cost, longer development, and higher defect
density, specifically, as shown in Figure 2.1 However, let
us examine in more detail the phenomena occurring at
points A, B, and C on the graph of Figure 2.

• At point A, requirements are not changing (bottom
row of Figure 1), so there appears to be no negative
impact on project success. However, what if needs
are changing, and we inhibit change to require-
ments? This is common among organizations who
think the solution to escalating costs is to prevent
requirements change. Figure 3 expands point A.
Note that if needs are not changing, point A on
Figure 2 is accurate and minimal, but if
requirements are not being updated and needs are
changing, then negative project success factors grow
(right side of Figure 3 culminating at point A’).

1 Axes on this and subsequent figures are void of scale;
the non-decreasing nature of the graph is important, not
its shape. As the granularity of requirements increases,
the graph’s shape becomes more predictable, as
demonstrated by Lavazza and Valetto [17].
At point C, documented requirements are changing (top row of Figure 1), so there are negative impacts on project success factors. However, what if actual needs are not changing, and we are simply making changes to the documented requirements for non-project critical reasons such as poor leadership, ego gratification, and so on? Figure 4 expands the situation occurring at point C. Note that if actual needs indeed are changing, point C on Figure 2 is accurate but is a minimal impact point, not a maximum impact point. If actual needs are not changing, but documented requirements are being changed nonetheless, then negative project success factors grow even more (left side of Figure 4, corresponding to boxes YNY, NYY, and NNY, culminating at point C').

At point B, documented requirements are changing moderately, so there are moderate impacts on project success factors. However, what if actual needs are changing more rapidly? Or less rapidly? In both cases, project success becomes less likely. Figure 5 expands the situation occurring at point B (left side of the figure shows escalating risk when needs are not changing; right side shows escalating risk when needs are changing more rapidly than the documented requirements are changing). Note that point B on Figure 2 is a minimal impact point, not a moderate impact point.

The phenomena at these points make it clear what is happening: requirements change must remain proportional to change in actual needs. It is quite easy to control requirements change: simply do not make them. The result will be a project that completes on schedule, within budget, and with minimal defect density, the subjects that are so often correlated with “project success.” However, if we consider the ultimate (most critical) defect to be a failure to meet a customer need, then requirements changes must be proportional to needs changes (regardless of the timing of changes as reported in [15][18]), and will result in the lowest critical defect density. Putting Figs 3, 4, and 5 together gives us Figure 6. Note that points A, B, and C of Figure 2 all become minimal points in Figure 6 when examined from the requirements change axis. This becomes more obvious when we rotate Figure 6 so that the “change rate for requirements” axis is facing us, as shown in Figure 7.
5. Summary & Conclusions

Many researchers have accurately proclaimed that as requirements change, the likelihood of project failure increases. This paper argues that although this phenomenon is important, equal attention should be given to changes to users’ and customers’ needs, over which we have no control. As critical changes occur to these needs, we have a choice of incorporating them in our requirements (thus increasing project risk but simultaneously decreasing product risk [19]) or ignoring them (thus decreasing project risk but simultaneously increasing product risk [19]).

References