12-2003

Software Development Productivity and Cycle Time Reduction

Victor A. Clincy
Kennesaw State University, vclincy@kennesaw.edu

Recommended Citation
Victor A. Clincy. 2003. Software development productivity and cycle time reduction. J. Comput. Small Coll. 19, 2 (December 2003), 278-287.

SOFTWARE DEVELOPMENT PRODUCTIVITY AND CYCLE TIME REDUCTION *

Dr. Victor A. Clincy
Computer Science & Information Systems Department
Kennesaw State University
Kennesaw, Georgia 30144
770-420-4440
vclincy@kennesaw.edu

ABSTRACT:

Increasing software developers' productivity and reducing the software development process' cycle time are key goals for organizations responsible for building software applications. This paper proposes four major areas impacting an organization's ability to increase developer productivity and reduce development cycle time. The four areas are (1) organizational structure and climate, (2) reward system, (3) software development process and (4) the use of software design and testing tools.

0. INTRODUCTION

Increasing software developers' productivity and reducing the software development process' cycle time are key goals for organizations responsible for building software applications. This paper proposes four major areas impacting an organization's ability to increase developer productivity and reduce development cycle time. This paper characterizes an organizational structure, reward system, software development process and the use of technology in enabling and promoting increased productivity and reduced cycle time. The characterizations and perspectives presented in this paper is a summarization of a survey given to some team leaders of a Fortune 100 company with proven track records of cultivating and managing effective teams.

In addition to the four areas effecting an organization's ability to increase productivity and decrease cycle time, some common attributes of highly effective teams were realized and documented.

1. ATTRIBUTES OF EFFECTIVE SOFTWARE DEVELOPMENT TEAMS

The following common attributes were identified for highly effective software development teams. In Table 1.0, each attribute is described with the associated impact on cycle time and productivity.

Attribute Description	Cycle Time and/or Productivity Impact
The teams were greatly involved with the customers and users through out the life cycle of the project (not just initially).	Reduced handoffs and formal communications, thus reducing overall cycle time.
The team sizes were typically small (less than 15 members)	Cut down on the lines of communications and handoffs amongst team members, thus reducing cycle time. Also by having smaller teams, less cost was spent on human resources (typically the largest cost factor)
The team were near-to-completely self-contained (not depending on any external support in completing the project). The teams were mostly comprised of generalists versus specialists.	Because the team could handle the entire project, it minimized external handoffs and bureaucracy, thus decreasing cycle time and increasing overall productivity. By team members being generalists, productivity per member was increased.
Each software developer was competent in more than one language.	By each software developer being competent in more than one language, productivity per member was increased.
Each software developer was competent in more than one function (ie. design, coding, testing, etc..) throughout the life cycle of the project.	By each software developer being involved in more than one function throughout the life cycle of the project, it boosted productivity per developer.
The manager (or leader) of the team had a broad range of responsibilities relating to gathering requirements, software development and project management.	Increased the teams' overall efficiency by increasing productivity and reducing cycle time at the same time.
The manager (or leader) of the team functioned as the motivator, coordinator, and supporter of the team and its members.	Increased the teams' overall efficiency by increasing productivity and reducing cycle time.at the same time.

The team members were hand picked (not assigned) and were chosen based on their abilities (not general experience level).	Right sizing with the right mix of skills optimized the human resource cost factor.
The team members worked well together; and routinely did activities together outside of work.	In some fashion, cycle time was reduced due to the added informal communications amongst members.
Each team member was empowered to take whatever action is needed to succeed.	Reduced bureaucracy and development idleness
All team members were focused on project goals.	N/A
The team welcomed new "value-added" ideas, approaches and technologies (and tools)	Reduced cycle time and increased productivity
For the most part, each software developer worked autonomously with very little direct supervision.	Reduced formal and informal communications, thus increasing productivity per member and decreasing overall cycle time.
The team members communicated very frequently in an informal manner.	Reduced formal communications provided decreased cycle time.
The teams generated a minimum (and essential) amount of documentation.	Reduced documentation provided decreased cycle time.
Team members were willing to change roles in completing the project	Members were active throughout the projects' life cycles which increased each member's productivity
A quality plan was integral from the start of the projects. Quality By Design Versus Conformance	By incorporating a proactive approach to quality at the start of the project, it minimized reworked and gave rise to decreased cost and decreased cycle time.
The members in the groups were geographically close.	Reduced formal communications but increased informal communications, thus increasing productivity per member and decreasing overall cycle time

Table 1.0:Attributes of Effective Development Teams

2. ORGANIZATIONAL STRUCTURE RECOMMENDATIONS

The organizational structure is critical to creating and maintaining highly effective software development teams. Key recommendations for organizational structure are described in Table 2.0 and for each recommendation, impacts and needs are listed.

Recommendation Description	Impacts	Needs
• Staff projects with small effective teams • Keep effective teams intact • Effective teams should take on new projects over time. • Teams should be self-contained, self-directed and small (< 10 people). • Team members should be able to perform tasks that require more than one skill set (ie. UI, DB, etc..)	• The number of effective teams will increase over time thus increasing overall performance of the company. • Critical, high-priority projects will be powered by an effective team	• Upper management will need to make resource plans around this concept. • Upper management will need to make necessary organizational changes to align resources to fit this concept.
• A key to creating or maintaining a high performance team is having an effective Technical Team Lead: • Technical Team Lead must be officially Empowered and Accountable • The Technical Team Lead should have significant knowledge of systems engineering, systems architecture, software development techniques and tools, and project management. • The Technical Team Lead must have proven/potential leadership skills. • The Technical Team Lead should be able to foster a climate of innovation and managed risk taking.	• Technical Team Lead will be responsible for end-to-end delivery of a solution, cycle time, handoffs and bureaucracy will be greatly reduced. Also efficiency and productivity will increase. • A Technical Team Lead being accountable without legitimate empowerment will not work.	• Upper management will need to appoint technical team leads with proven track record of effectively performing the described role. • In providing accountability and empowerment to Technical Team Leads, this will require some risk-taking on the part of upper management.
• Collaborative/complementary skills are key in functioning as a high performance team. • All members should embrace flexible spiral development processes • Team members must understand business needs and growth directions • Team should develop and employ off-the-shelf component integration expertise • Skill development in people should include mentoring and formal training within the team	• The right sized skills mix helps make teams more self-contained, effective and productive. • Integrative approaches reduce cycle time.	• On a per project basis, objectively define skill set; only add people to the team that have the set of skills needed (otherwise, you could be creating unnecessary overhead) • Be willing to disband a team if performance goals aren't reached • Be willing to move people from team to team in creating high performance teams

Table 2.0:Organizational Structure Criteria

3. REWARD SYSTEM RECOMMENDATIONS

An objective reward system is key in boosting a team's productivity..Highly productive teams must be rewarded accordingly; rewards could be financial or non-financial in nature. By properly rewarding effective teams, it sets the environment to be more productive. If all teams, both high performance teams and not so high performance teams, are rewarded the same, it (1) penalizes the high performance teams, (2) reduces the high performance teams' morale and (3) doesn't give average-to-low performance teams an incentive to do better since their reward is practically the same as the high performance teams' rewards. Key recommendations of a reward system that promote effective software development teams are described in Table 3.0. For each key recommendation, impacts and needs are listed.

Recommendation Description	Impacts	Needs
• Reward system needs to be objective in fostering high productive teams: • Rewards can be financial and non-financial in nature • The non-financial rewards can be giving the team a high visibility or critical projects. • Individual financial rewards can be traditional raises and promotions.	An objective reward system should encourage desired individual and team behaviors to increase performance	Use various metrics in measuring teams performance
• In the application of objective factors in the rewards systems, consideration must be given to circumstances beyond the control of project teams. • Rewards should also be considered for research, application of new technologies, and infrastructure improvements	Encourage desired individual and team behaviors to increase performance	Define criteria to use in measuring these areas (ie. research, infrastructure, improvements, etc..)
Provide incentive for successful Technical Team Leads to move to lesser productive teams; also provide a vehicle for the Technical Team Lead to gracefully exit the project provided they are not able to turnaround average-to-low performance team.	• Will increase the number of high performance teams • Will increase the number of effective Technical Team Leads by opening up slots on high performance teams	Define the criteria for Technical Team Leads moving and exiting; the reward system should be tied to the criteria

Table 3.0:Reward System Criteria

4. SOFTWARE DEVELOPMENT PROCESS RECOMMENDATIONS

The software development process should be adapted to the team and not the team adapted to the process. In enabling increased productivity, a multi-path process should be used

that is robust and adaptive in nature. The process should be designed to accommodate three general types of software projects: (1) *Problem Resolution type* (ie. building non-production software applications to quickly resolve business problems), (2) *Creativity/Experimental/New Technology type* (ie. software projects requiring new technology or approaches; critical solutions needed very fast; make use of rapid prototyping) and (3) *Tactical/Execution/Legacy type* (ie. existing systems requiring enhancements and new features). Key recommendations for a software development process that enables productivity are described in Table 4.0. For each key recommendation, impacts and needs are listed.

Recommendation Description	Impacts	Needs
Define a multi-path process that is adaptive and robust; the process should fit the project versus the project fitting the process	Reduces cycle time by eliminating overhead	Define a multi-path process; put together a team of effective Technical Team Leads in devising process
The Developer role should expand into a System Integrator Role which consists of the responsibilities of end-to-end development of features and systems.	Reduces cycle time by eliminating handoffs; role reduction and consolidation	Develop community with breath of knowledge for better flexibility in staffing and job portability
• Architecture and OOD skills • Coding proficiency (UI, application and DB) • Testing skills (unit, integration, system) • Product Delivery and deployment skills		
Systems engineering should be a task within the development process and not a hand off; Systems Engineers should be involved with the project during it's entire life cycle; SE performs numerous different tasks in addition to requirements gathering.	Increase the number of people performing System Integrator Role; Reduces cycle time by eliminating overhead and handoffs	Define systems engineering role as an integral role with in the development process
Don't document anything for the sake of documenting; multi-path process should address this more specifically	Reduces cycle time by eliminating overhead	Determine documentation actually needed

Table 4.0:Software Development Process Criteria

5. TECHNOLOGY USE AND ARCHITECTURE RECOMMENDATIONS

Architecture sets the stage for a project's success or failure. There should be some investment up-front from an architectural perspective in devising an architecture that will allow rapid development in the future; in addition to this, it's not suggested that the devising of that architecture take long periods of time or that the development of that architecture's infrastructure takes long periods of time, due to the fact there are new technologies being constantly

introduced in the market. Key recommendations of technology use and architecture that enables decreased cycle time and increased productivity are described in Table 5.0. For each key recommendation, impacts and needs are listed.

Recommendation Description	Impacts	Needs
• Define suitable and robust architecture up-front; • The cost and time-to-market delays associated with re-architecting have to be viewed as an investment in higher productivity and lower costs for future releases. • If modifiability/Reuse is not built-in to the architecture, productivity will not be good.	Architecture sets the stage for a project's success or failure Don't expect a quantum leap in productivity unless you make quantum leap architectural changes.	Perform architecture discovery before project starts (make sure knowledgeable folks are involved)
Don't use new technology for technology sake. Convince yourself that the technology will bring some tangible and needed improvements and business value	Reduces the probability of slipping schedules and increasing cycle time and cost; increases probability of success	
• As a part of the process, encourage teams to use modeling tools, testing tools, a common modeling language (UML), etc. • Also as part of the process, encourage teams to reuse code, write code generation programs for the more repetitive coding tasks (ie. UI forms, DB CRUD) and review, evaluate and use off-the-shelf components to boost productivity	Reduces cycle time over time, everyone speaks the same language	Identify useful tools

Table 5.0:Technology Use Criteria

6. CONCLUSION

The potential amount of improvement is expected to vary by the type and size of the project and particularly by the nature of any existing systems architecture that may be foundational for the project efforts.

On high performance teams, each member of the team is motivated to work hard and do whatever it takes to make the project a success. Somehow this work ethic is achieved. It is believed that factors in the organizational climate, processes, rewards systems and use of design tools can encourage and enhance effective team performance.

7. APPENDIX

A. Survey Questions

What is your project's title ?
Give a brief overview or purpose of the system.
Is the project a new start-up or is it an enhancement to an existing application ?
Is your application critical to the company (ie. example of a critical application would be an application that touches some network element). Will it be very detrimental to the company if your system went down for a day or so ?
When did the project start ? Number of releases thus far ?
What are the Function Point Metric Results for all releases analyzed (include FP, FP/SM, $/FP and time-to-market) ?
Give a brief history of the start
What is the Project Team size ? # of systems engineers ? # of developers ? # of non-developers and non-systems engineers ? Is there a project manager ?
Does your team consist of contractors ?. If so, what percentage ?
What is the structure of your project team ? Is the project broken in functional areas ? (ie UI, application, data, etc..) If so, are there team leads for each functional area ? Are the project team members from different organizations ? Is there a lead developer ? Given more than one systems engineer, is there a lead systems engineer ?
How was the team formed (ie. hand picked, assigned, etc...)
What criteria (if any) was used to choose team members
For developers, do they perform more than one technology task ? (ie. developed UI, developed application, developed DB, etc..)
For developers, do they perform more than one function (ie, design, development, testing, etc..).
Do the systems engineers stay with the project through it's life cycle or do the SE only gather and document requirements? If the latter, how long did the hand-off take? Was the handoff formal ?
What type of approach is used in gathering requirements ?. Do you gather ALL requirements upfront before starting development ?. Or do you gather some requirements and perform some prototyping before starting development ? Do you have multiple releases throughout the year ?
Is the problem domain very stable, fairly stable or not so stable ?
In general, how often does the team interface with the users or user reps ?. Are demos given to the users ? If so, how often. Is there a formal requirements document requiring signatures before development work began ? Who interfaces with the user or user reps (ie. lead SE, project manager, lead developer, etc....)
Is your team responsible for all of the development or is some of the development done by other teams ?.

How often does the team meet ?. In general, how is communications conducted amongst the team members (ie. meetings, conference calls, emails, etc..). What's the frequency of the various communications vehicles ?
Did you make use of prototypes - did you practice prototyping ? If so, why ?
Is there Upper Management involvement ?. If so, how often. What issues did upper management bring forth as they related to your project ?. Who sought funding for your project ?. Who communicated issues regarding the project to upper customer management ?. In general, was upper management involvement critical to your project's success. If so, why ?
Did your project conduct an architectural discovery before development ? If so, who conducted the architectural discovery ?. Did you need approval for the architecture ?. For the architecture recommended, was it a new technology or an existing technology ?
Are you following some company formal process?. If not, describe the process used ?.
Are you making use of off-the-shelf components ?. If so, which components ?
Are you making use of OOD/P ? If so, explain briefly ? If so, what areas are these concepts applied to (ie. application, etc...) ?
Are you making use of a database ?. If so, what type and why ? Are you initially creating database models ? If so, who does this ?
Are you practicing "reuse" ? If so, explain ? If so, what areas ?
What documentation are you generating ? Are you creating formal design documents ?. If so, who creates them ?. Are you creating test plans. If so, who creates them ?
Do you do testing within the team ?. If so, what testing. Is there any testing done outside of the team ?. If so, who does this testing ?. If so, do you have to create some documentation before turning it over for testing ? How long acceptance testing last ?
What software tools did you make use of (ie. Sablime, etc...)

B. Bibliography

[1] Steve McConnell, *Rapid Prototyping: Taming Wild Development Schedules*. Microsoft Press, 1996

[2] Barry Boehm, *Software Engineering Economies*. Prentice Hall, 1981.

[3] Tom DeMarco and Timothy Lister, *Peopleware: Productive Projects and Teams.* Dorset House Publishing, 1999

[4] J. Albano, J. Bramley, V. Clincy, M. Colluci, and N. Hultman, *High Performance Team Task Force*, AT&T, November 5, 1999

C. Author's Profile

Dr. Victor Clincy is currently an Associate Professor of Computer Science and Information Systems at Kennesaw State University. Prior to joining KSU's faculty, he was a

Senior Engineering Manager with Scientific-Atlanta Corporation responsible for network analysis. Prior to Scientific-Atlanta, he worked for AT&T and managed a group of systems engineers and software developers responsible for building applications and tools for network designers. In his earlier years with AT&T, he was a Member of Technical Staff with Bell Labs in New Jersey. He has also worked for companies such as NorTel, ALCOA, Bridgestone/Firestone and Texas Instruments.

Dr. Clincy holds a post-graduate degree from Columbia University in Computer Systems Engineering. He also holds a doctorate in engineering from Southern Methodist University and two Masters degrees in engineering, one from the University of Pittsburgh and another from North Carolina State University. His undergraduate degree is in Electrical Engineering from Mississippi State University.