Continuous Integration in Software Management

Gautam Kumar, Prof. David Socha
Computing and Software Systems
University of Washington, Bothell
{gautamk,socha}@uw.edu

Abstract

Continuous Integration (CI), a practice where developers integrate frequently [21] is considered an integral part of Agile development. CI is often thought to be a safety net that prevents developers from deploying broken code to production.

A similar process and ideology of continual improvement and integration can be applied to Software Management, the art and science of planning and leading software projects [22].

This paper attempts to explore the process of creating such an integration practice within an Organisation.

Keywords: Software Management, Agile development, Continuous Integration

Introduction

Martin Fowler in his seminal article [10] on Continuous Integration (CI) describes it as a practice where members of a team integrate their work frequently and each integration is verified by an automated build to detect problems quickly.

In the context described above integration is the process of combining the work of all the developers of a project into a cohesive software. On the other hand if considered generically, Integration can be thought of as a practice of combining the work of multiple people and verifying its correctness. Using a similar collective, Continuous integration could be the process of merging the work of multiple people after specific events or set periods of time.

In this paper we explore the history, benefits and the value of continuous integration from a software development perspective. We then try to apply the same concepts towards continuous integration in software management as a whole with a sample case study in people management.

Continuous improvement

The American Society of Quality define continuous improvement to be "The ongoing improvement of products, services or processes through incremental and breakthrough improvements." [9]

Continuous improvement as a concept has existed for a long time. We can find mentions to continuous improvement in texts dating back to the mid-19th century [15].

During the industrial revolution of the mid-19th century workers were often trained not to think but rather just perform just their task. Thus the task of thinking was effectively cut off from the task of doing, which lead to the thinkers not understanding the practical restrictions of the real world and the increasing complexity of the systems [15]. The same concept applies even today with software systems, computers have taken on the role of workers while humans have moved on to the task of thinking.

This separation of thinkers from the practical implications of real world lead to the development of Continuous improvement which emphasizes a continuous stream incremental innovation [3].

Continuous improvement is one of the primary pre-requisites for continuous integration. A stable, unmaintained project does not change and thus has no need for a continual process of integration. We assume here that any changes introduced into a project is designed to improve the project in some way. There are situations where changes don't necessarily improve the project such as when malicious code in secretly introduced.

Continuous Integration

History

The term continuous integration was first used by Grady Booch in his 1994 book [4]. Booch refers to continuous integration as a micro process which produces many internal releases. He also mentions that testing should also be conducted as a continuous activity during the development process.

Kent Beck and Ron Jeffries, the inventors of Extreme Programming(XP), wrote about implementing continuous integration in their book on XP [1]. Beck and Jeffries had worked together on building a payroll application for the Chrysler corporation and pioneered the process of XP during their time working on the Chrysler Comprehensive Compensation System (C3). Beck and Jeffries implemented the CI process within the C3 development team as a method to escape integration hell while also promoting the concept that all developers must work on the most recent release of the system [7].

Complexity

Continuous improvement incentivizes a structure where anyone in the organisation who possess a basic skill set can contribute to the implementation of solutions for a common problem set [3]. Such a process brings along with it an explosion of ideas and as some of those ideas when implemented drastically increase complexity. Such an increase in complexity eventually leads to a decline in either the quality or velocity of development. [23].

Continuous integration thus evolved as a mechanism to manage the risk associated with complex software systems [23] and attempts to provide a safety net [10] to developers.

Though continuous integration doesn't actually fix bugs, it offers a lot of benefits which help in improving the overall quality of the project such as Improved communication, Project predictability and increased developer productivity [20]. So Continuous integration in many ways can be considered a modern method of complexity control [2].

Process

Continuous integration is typically setup as an automated process which fetches new changes from a source repository after a developer checks-in changes and builds,tests and sometimes even deploy code. Fig 1 describes such an implementation.

Benefits

Risk reduction: The greatest and most wide reaching benefit of CI according to Martin Fowler [10] is risk reduction. In an organisation without CI, after a long development cycle, neither the managers nor the developers are sure the time it would take complete integrating the various components. CI completely eliminates this risk.

Bug discovery: CI doesn't necessarily fix bugs, it merely makes bug discovery dramatically easier when paired with self-testing code (code with a large percentage of test coverage) because this makes it possible for the CI system to automatically test for bugs on every build. Since there are many builds throughout the day bugs are discovered easily and earlier in the development life-cycle.

Frequent deployment: A long integration cycle is one of the biggest barriers to frequent deployment [10] and since continuous integration eliminates this problem frequent deployment becomes a viable endeavour. Frequent deployment is important because, apart from the fact that users can get new features and bug fixes much more rapidly, frequent deployment allows users to give rapid feedback thus reducing the risk of delivering an unusable and disliked product further.

Software Management and Continuous Integration

The inherent complexity in software management is best described by the law of leaky abstractions [19] which states that all non-trivial abstractions leak to some degree.

The Law of leaky abstractions

An abstraction in the context of computers and software refers to a mechanism that tries to hide the complexity of the underlying system while still offering a useful interface to interact effectively with the underlying system. Abstractions are meant to simplify the process of interacting with an inherently complex system by hiding away certain irrelevant details.

Though abstractions vastly simplify interactions with a complex system, the abstractions are still limited by the capabilities of underlying system and thus some serious failures always trickle up through abstractions and surprise the user in unexpected ways. Fig 2 describes such a trickle from hardware all the way to the application level.

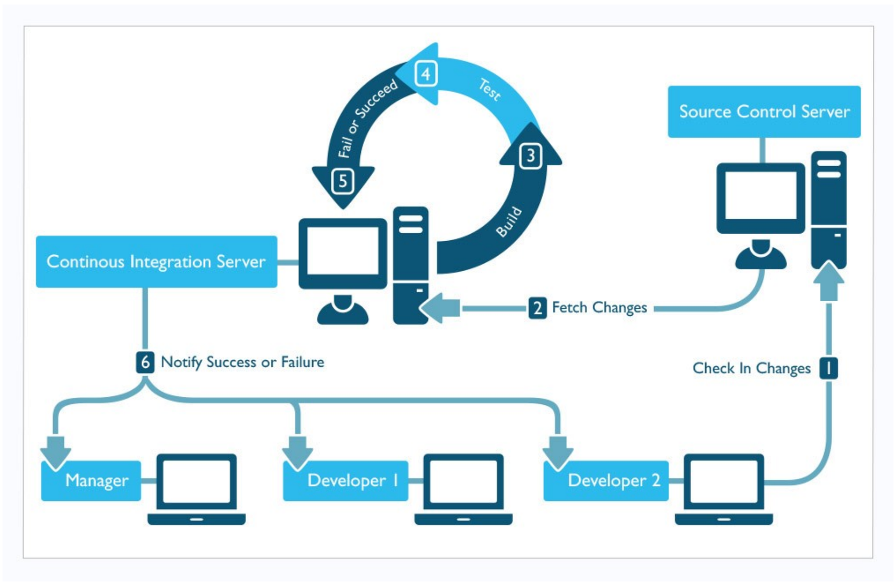

Fig. 1. A typical CI process where a developer checks in changes into the source code repository, the CI server fetches the changes, builds the code, run unit, integration and functional tests and marks the build of its status. If the build was a success the code changes are expected to be propagated to all developers. source: [6]

A classic example of leaky abstractions would be Operating Systems. On the windows operating system any failure in the underlying hardware or its drivers can potentially lead to a BSOD (Blue screen of death) in which the computer displays a cryptic message on a blue screen and prompts the user for action. Though BSOD is a rare occurrence under normal operating conditions, its appearance often perplexes an average user who is unable to make a decision on the next course of action [14].

Abstractions in Software Management

Software Management as a subset of Project management is primarily concerned with balancing the needs of stakeholders with the abilities of developers to deliver value.

Software management practices abstract away the inner workings of processes in the organisation. For example Agile methodologies such as Scrum and Kanban are an abstraction which use the best practices from Time management, Team communication, Performance management and Theory of constraints to derive value from developers collaborating within a team.

Another example of abstraction in Software management in teams would be embracing diversity which is an abstraction over Psychology and Economics to help teams take varied view points on any particular problem.

Cynefin perspective

The Cynefin model describes the relationship between the ability to determine causality in a situation and the way people respond to said situation. The model describes four states. The obvious state where cause and effect are easy to determine so people can sense, categorise and respond. The complicated state where the cause is not immediately apparent but can easily be determined by sensing, analysing and responding. The complex state where it neither the cause nor the effect is immediately apparent but we can run experiments to verify our direction so we probe sense and respond. The chaotic state where the system is not stable and we cannot wait to determine the state of the system so we act, sense and respond. Finally we have a state where it is not possible to determine the domain of the system Fig 3 describes the various states of the cynefin model in a easy to visualize manner.

When considered from the perspective of the Cynefin model [17], the relationship between software management processes and the people in the organisation can range from complex to outright chaotic. For example in an organisation where many

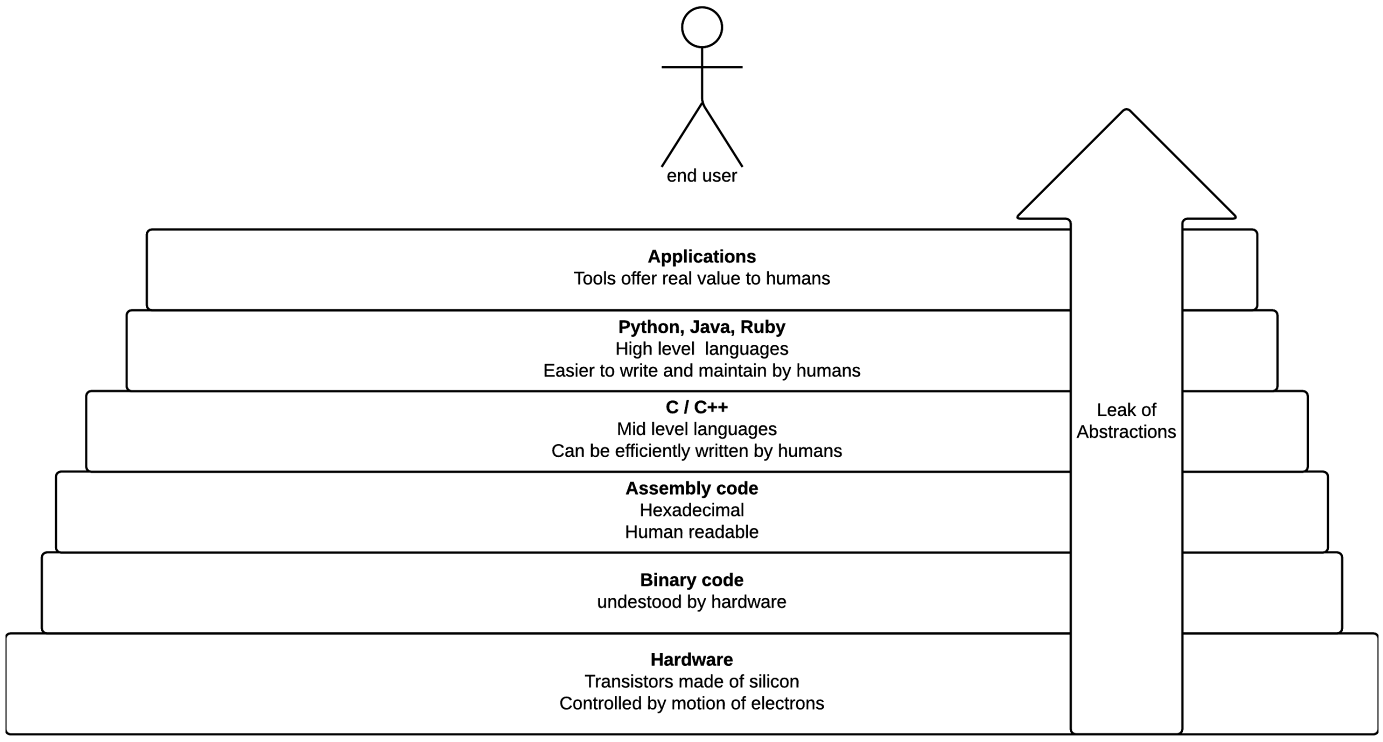

Fig. 2. The law of leaky abstraction: All non-trivial abstractions, to some degree, are leaky. In this scenario we consider a typical computer system where any failure or changes to the hardware affects the end user even as 5 or more layers work towards hiding the complexity involved in working with complex electronics. The arrow shows a leak in the abstractions layers propagating upwards.

people have experience working in an agile environment working within the bound of the organisation's implementation of agile can be a complex task for an individual. On the other hand when an organisation moves from waterfall towards agile, people in the organisation have a chaotic relationship with the new software management process and struggle to adapt to the new way of thinking. Agile software development thrives just at the edge of chaos and its easy to slip into a chaotic state when trying to transition from other software management methodologies [12].

Software management and risk

The cynefin model shows us that as a process software management is inherently a risky endeavour. If we consider software management to be analogous to software development we can see that the software management process can have bugs and integration issues from teams not working together, teams making mistakes, a process which doesn't consider the needs of a certain section of the workforce in an organisation and so on. Thus bugs and integration issues in the software management process increase the risk to both the project and the organisation as a whole because failure in the software management process can potentially leave employees disgruntled and eventually lead to attrition.

As with any non trivial abstraction, software management is also leaky and implementing continuous integration into Software management practices could potentially offer benefits similar to software development.

SCOPE OF SOFTWARE MANAGEMENT

The software management process can include areas of management which traditionally come under the purview of Project management such as people management and customer relations. Including these factors into the software management process is however crucial to the overall goal of efficient and effective management of the software process to

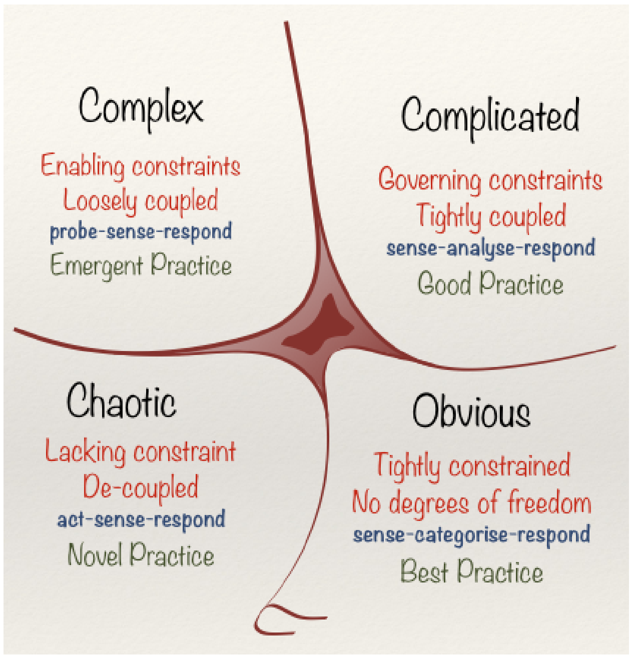

Fig. 3. The Cynefin model, The states on the left are un-ordered sates where causality cannot be determined and the states on the right are ordered where the system is highly constrained so causality is either obvious or easy to determine. The central dark area refers to un-determinable state when the state of the system cannot be determined by any reasonable means [18]

enable developers to build products that deliver value.

Some potential areas which might come under the purview of software management are

- People Management
  - Hiring
  - Culture
  - Expectations
  - Organisation structure
  - Diversity
- Project management
  - Requirements gathering
  - Technical choices
  - Project priorities
  - Team selection
- Product management
  - Market analysis
  - Solving the right problem
- Process management

Continuous integration in people management

Continuous integration has been well defined from the software development perspective, definitions from the perspective of software management on the other hand are quite sparse.

Continuous integration is best implemented in an organisation where a significant emphasis is placed on continuous improvement and failure is considered a part of the learning process [8].

In the paper we limit the scope of implementing continuous integration in the software management process within people management, specifically on-boarding new hires. Continuous integration in terms of people management could be thought of as a way to verify that the everyone and/or every team in the organisation is working towards a common goal.

Relation to software management

An organisation's products and the value derived from them are built by people in the organisation. In a software organisation the primary product is software so managing the people who build software would come under the purview of software management.

Implementing continuous integration

Continuous Integration and improvement are by their nature an iterative process thus implementing them within the context of software management requires the management team to follow a disciplined process of continuous evaluation and an evolution similar to the process described in fig 1.

On boarding new hires

As an example we consider on boarding new hires to highlight ways of introducing continuous integration into the software management process of an organisation.

Adoption: The initial adoption of any new process within an organisation may require a steep learning curve when people learn ways of efficiently implementing and utilizing said process. One of the goals of implementing an agile continuously integrated process should be to determine an amicable middle ground between people adapting to the process and the process adapting to the people. For example, some the factors which affect the success

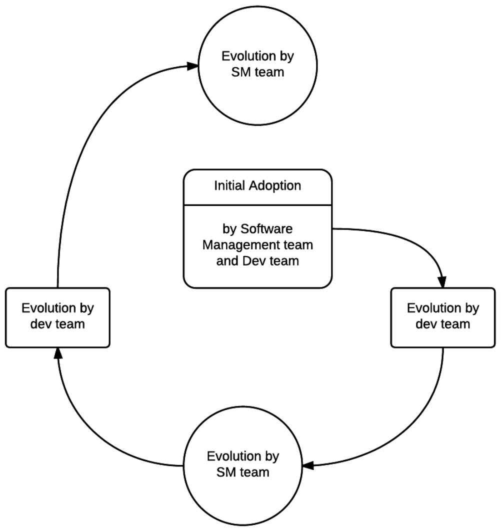

Fig. 4. The Software management team has to evolve the adoption process as the dev team deviates from the initial Software developement model

of adopting agile methodologies in an organisation are customer collaboration, satisfaction and commitment, communication and negotiation skills of the team and also cultural factors [13].

These success factors differ when bringing in new team members. For example a team member fresh out of university might adapt to an existing process at an organisation with ease while a seasoned developer who is loyal to the waterfall model might have a hard time adapting to the rapid pace of an agile organisation.

As teams being to understand the expectations of their customers and their organisations the previously defined software process evolves thus as time goes on each team / organisation starts building a unique variation of the agile process.

It is the responsibility of the software management team to enable new developers to adopt and adapt to the unique process of the dev team they are going to be working with.

Success criteria: The software management team needs to have a clear and evolving success criteria which monitors each new hire and how well they are able to adapt to their new teams. This differs from Definition of Done (DoD) based on the fact that DoD is merely a set of activities which need to be performed to get the user story to a shippable state while Success criteria is a broader set of expectations from the candidate which are recorded based on subjective observations.

Some of the metrics to measure under success criteria could be Team communication, Knowledge sharing [5], Productivity acceleration [11] and Empathy [16].

Success criteria in the case of on-boarding could potentially be an milestone goals for each metric previously described after predetermined periods of time. An example of this could be measuring team communication by asking a new employee after three, six and 12 months the number of people they have had useful interactions with, on their team and on other teams. Depending on the size of the organisation a nominal range can be defined to discover anomalies and possibly offer assistance if needed.

Continuous Integration: As an example continuous integration from the perspective of on-boarding new hires can potentially be implemented as an internal process which evaluates the effectiveness of the on boarding process with both the team, the new member and the management to evaluate value of the on boarding process, effectiveness of its implementation and any potential flaws or breaks in its implementation.

Any implementation of the on boarding process could potentially break due to changes from either the dev team or the people management team. For example the dev team might introduce a new method of code review but this code review technique might be missing from the orientation packet for new hires. The continuous integration process for evaluating new hires could potentially catch this so that it can be corrected quickly.

SUMMARY

In this paper we describe the how software management is a complex task using the Cynefin model and the law of leaky abstraction. We also describe why continuous integration can help discover flaws in the process early with a case study about on-boarding new hires.

REFERENCES

[1] Egidio Astesiano. *Fundamental Approaches to Software Engineering: First International Conference, FASE'98, Held as*

Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings. Springer Science & Business Media, March 1998.

[2] Kent Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[3] J. Bessant, S. Caffyn, J. Gilbert, R. Harding, and S. Webb. Rediscovering continuous improvement. *Technovation*, 14(1):17–29, February 1994.

[4] Grady Booch. *Object-oriented analysis and design with applications*. Benjamin/Cummings Pub. Co.,, 1994.

[5] Elizabeth F. Cabrera and Angel Cabrera. Fostering knowledge sharing through people management practices. *The International Journal of Human Resource Management*, 16(5):720–735, May 2005.

[6] Aaron cois. Continuous Integration in DevOps, 2015. https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html.

[7] cunningham. Cunningham and Cunningham, C2 wiki: Continuous Integration, 2011. http://c2.com/cgi/wiki?ContinuousIntegration.

[8] Amy C. Edmondson. Strategies for Learning from Failure, April 2011.

[9] American Society for Quality. Continuous Improvement Model - Learning Resources| ASQ.

[10] Martin Fowler and Matthew Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, page 122, 2006.

[11] Dick Grote. Making Onboarding Work, June 2011.

[12] Karlheinz Kautz and Sabine Zumpe. Just enough structure at the edge of chaos: Agile information system development in practice. In *Agile Processes in Software Engineering and Extreme Programming*, pages 137–146. Springer, 2008.

[13] Subhas Chandra Misra, Vinod Kumar, and Uma Kumar. Identifying some important success factors in adopting agile software development practices. *Journal of Systems and Software*, 82(11):1869–1890, November 2009.

[14] Scott ROSENBERG. The Law of Leaky Abstractions. *Technology Review*, 110(1):46–47, February 2007.

[15] Dean M Schroeder and Alan G Robinson. America's Most Successful Export to Japan: Continuous Improvement Programs.

[16] Emma Seppala. The Hard Data on Being a Nice Boss, November 2014.

[17] David Snowden. *Cynefin, A Sense of Time and Place: an Ecological Approach to Sense Making and Learning in Formal and Informal Communities*.

[18] David J. Snowden and Mary E. Boone. A Leader's Framework for Decision Making, November 2007.

[19] Joel Spolsky. The Law of Leaky Abstractions - Joel on Software, November 2002.

[20] Daniel St\a ahl and Jan Bosch. Experienced benefits of continuous integration in industry software product development: A case study. In *The 12th IASTED International Conference on Software Engineering,(Innsbruck, Austria, 2013)*, pages 736–743, 2013.

[21] Daniel Ståhl and Jan Bosch. Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, 87:48–59, January 2014.

[22] Andrew Stellman and Jennifer Greene. *Applied software project management*. " O'Reilly Media, Inc.", 2005.

[23] Yury V. Zaytsev and Abigail Morrison. Increasing quality and managing complexity in neuroinformatics software development with continuous integration. *Frontiers in Neuroinformatics*, 6:31, 2013.