Test strategies in distributed software development environments

Anuradha Mathrani *, Sanjay Mathrani

School of Engineering and Advanced Technology, Massey University, Auckland, New Zealand

ARTICLE INFO

Article history:
Received 6 September 2012
Accepted 13 September 2012
Available online 5 October 2012

Keywords:
Software development lifecycle
Distributed teams
Test strategies
Synchronize and stabilize

ABSTRACT

Distributed software development has resulted in formation of business partners spread across different
economic, temporal, and organizational zones collaborating together for shared authorship of evolving
software artifacts. However, the distributed approach is not without risks, and organizations implement
specific test strategies to assist in the verification process of work-in-progress software artifacts. This
document discusses the test strategies adopted in the software development lifecycle by a service provider
pursuing distributed software development in New Zealand, Australia, and India. Verification and
validation processes have been deployed to ascertain the quality, security and traceability of artifacts
developed in distributed sites. Findings reveal that strategies are based on protection of sensitive data
through management of test database, use of drivers and interfacing stub supports between modules, as
well as compliance verification on incremental releases through a customized “Synchronize and Stabilize”
lifecycle model. A staging environment, within the case business context, is used for evaluating the
robustness of the software product before its official launch in production environment. The actual
ongoing work practices within distributed software business environments are presented, which provide
value to academia, industry, ICT sector and government institutions.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

Distributed software development (DSD) has gained momentum since the growth of Internet, and necessitates engagement of
distributed teams working together over a common development platform. Teams situated at different geographical locations,
including offshore, collaborate using communication technologies to develop interrelated software modules that are integrated in
shared project workspaces. The software design process involves many iterative cycles of development before arriving at a
deliverable which can be deployed on a production server for live use by customers. Any inconsistency in one module can impact
the overall software design and lead to expensive overruns in the project development schedule. More importantly, any anomalies
found after the release of the software product on the production server can have severe ramifications to the products commercial
value and the organisations’ reputation. The Internet is known for its unforgiving memory, and any failures detected after the live use of
the final product can leave an unpleasant public trail. Hence, testing is a major consideration for any software design process.

Software testing is both a verification and validation process, and entails methods to test the software for risks, successes, and
failures at various levels of granularity that divide the development and integration process into several phases [1]. The correctness of
the overall integration process is unpredictable in dynamic real-time situations as performance varies with operational issues
occurring during the application service lifetime. Issues could relate to load imbalance, network bandwidth restrictions, architectural
constraints due to unregulated networks, or other unforeseen platform interoperability issues [2]. To better manage
interoperability across networked environments, organizations apply heuristic approaches to interpret local knowledge and
accordingly develop strategies for monitoring and optimizing processes related to domain-specific systems [3]. Hence, context
driven test processes are defined and a test specification criterion is aligned to match the dynamics of various building blocks in the
networked environment. Due to these reasons, test strategies cannot be adopted as is across product families and business
organizations even in localized environments, not to mention distributed environments.

The recent rise in global software development has resulted in formation of distributed teams spread across different economic,
temporal and social zones collaborating together for shared authorship of evolving software products. The distributed approach
to software development is considered more cost effective and competitive opposed to developing a software product in one
localized building, company or even country [4,5]. But, distributed

* Corresponding author at: School of Engineering and Advanced Technology, Massey University, Private Bag 102904, North Shore Mail Centre, Auckland 0745,
New Zealand. Tel.: +64 9 414 0800x9521; fax: +64 9 441 8181; mobile: +64 21 295 9746.
E-mail addresses: a.s.mathrani@massey.ac.nz (A. Mathrani), s.mathrani@massey.ac.nz (S. Mathrani).

0166-3315/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.compind.2012.09.002
software development is not without associated risks and challenges due to imbalance in knowledge overlaps and lack of familiarity between team members belonging to diverse social, cultural, and organizational locations. Some potential DSD risks identified in literature are inability to set common standards in quality due to operational issues in design choices and quality governance, liaison issues with virtual team members, and ownership issues regarding responsibility and maintenance of documentation and configuration management [5,6]. Issues relating to technical processes in application of test scripts to review system functionality, evaluate the design of components, and delve into software code to resolve defects have been discussed in literature to include static, white box, black box, and load testing [7–9]. However, little is known about the software practices drawn in definition of testing processes in DSD environments, such as execution of test schedules, application of test plans, or usage of security tracking devices in distributed environments where the component development is spread across different countries.

In this paper, we examine the test strategies deployed by a leading provider of software solutions to primary healthcare professionals in Australasia. This provider (referred by pseudonym HealthNZ) is involved in distributed software development activities with virtual teams spread across New Zealand, Australia, and India. Two case studies are conducted: one with HealthNZ as the primary healthcare provider and another with the IT service provider for HealthNZ. There are major implications of this study. First, the study reveals the high-level policies and governance procedures employed by organizations for managing their test strategies in globally distributed environments. The policies are based on business practice specification such as protection of commercially sensitive data across teams spread in diverse cultural and organizational regions. Second, the software development lifecycles to include risk mitigation and enable incremental releases of product lines by synchronizing tasks across distributed teams has been illustrated in a live environment. Finally, the study demonstrates how development environments are staged for verifying and validating upgrades made to established product lines through pilot (or beta) tests in expanding business markets. The findings from this study not only contribute to academia and practitioners in industry but also to policy makers and leaders in software development business.

The paper is structured as follows: This section has introduced the background of the study and proposed empirical investigation of test strategies employed in distributed software development projects. The next section reviews literature in relation to software test strategies and DSD environments. Next, the research questions are posed and the case study research methodology is explained to answer the research questions. A brief background of the two cases is provided next. Practices associated with risk assessment and test strategies are described. Each test strategy is discussed in relation to the potential risk perceived by the medical provider case (HealthNZ). The last section summarizes the findings and discusses how organizations manage their test strategies in distributed software development projects.

2. Importance of a test strategy

The primary motivation for establishing an organizational test strategy is risk assessment to identify and reduce business risks. Organizations assess the software specifications with user requirements to define test strategies which comprise elaborate plans for validation. These include writing meaningful test cases, planning storage and administration of tests, scheduling the test execution, setting standards for test completion criteria, defining responsibilities among teams, optimizing the hardware configuration, conducting regression tests, and debugging of errors [7]. Tests are conducted to confirm specifications and requirements by cross functional teams as well as verify both the expected and unexpected outcomes. Use of a modular design approach that involves reuse of standard modules (with existing and tested source code) wrapped by interfaces into components can eliminate some burdens of the verification and validation process. These standard modules enveloped by interfaces are packaged as black box components, use considerably less detailed test processes. But, as new user requirements emerge, designs incorporate new functionality needing development of new components which may not readily integrate due to mismatch of interfaces with pre-existing components. This leads to re-design of drivers and interfacing stubs between components to match data flows between the new and pre-existing parts [10]. Testing is a process of investigation to measure both correctness and incorrectness of software artifacts, and having an overarching test strategy assists management in keeping checks on the intermediate software deliverables within the software development lifecycle. Functional and performance testing of the intermediate deliverables is a critical activity that saves a lot of effort and development time, and ensures efforts are not being expended repeatedly on the same module with evolving code activities. However, having a test strategy does not guarantee complete correctness in software functionality; though it minimizes the risk of overlooking defects and helps to locate hidden deficiencies.

A lifecycle approach to information systems projects offers a more comprehensive representation and provides more insight into the quality of decision methods and balancing resources in the development process [11]. The traditional software lifecycle model called the waterfall model advocated “freeze a product specification, create a functional design specification, build components, and then merge the components – typically in one large integration and testing phase at the end of the project” [12, p. 60]. Lessons learned over the past years on software practices have taught us that requirements are not meant to be frozen as user requirements evolve with changing market needs and testability of software is ongoing during the entire software development lifecycle. To build reliable and robust software, the term ‘soft’ in software implies having flexibility and non-rigidity in accepting the changing user requirements. Technology too, is rapidly changing with changing market conditions involving emergence of new software platforms and development frameworks. Teams apply new techniques with emerging tools for faster execution of data flows and coordinate their efforts to adapt the changes into the current design. With regression tests, development teams ascertain whether addition of new functionalities and upgrades have affected the functionalities of previously developed modules to bring in stability during various phases of software lifecycle stages. In fact, regression tests are not a distinct level of testing, but a sub-phase of any testing activity since software modules go through many iterations of revisions and upgrades, requiring ongoing verification and validation to confirm their fitness with previous and current contexts [10,13].

The V model has originated from the waterfall model, but differs from the sequential and non-iterative nature of the waterfall model. While the waterfall model suggests that testing is more predominant near the end of a project development phase, the V model emphasizes the importance of testing to be done in parallel with software development. The V model describes how verification and validation activities go hand in hand with the software development cycle. Testing is linked to all development stages, through reviews of user requirements, test plans describing scope of system, component testing, integration testing and finally user acceptance testing [14]. Verification activities confirm in detail the consistency between descriptions (design and specification) and code, while validation activities “attempt to gauge
whether the system actually satisfies its intended purpose” [15, p. 17]. To include iterative and concurrent development with evolving technologies, developers design iteratively, not sequentially, and the V model serves as a baseline for many new lifecycle models. In the late 1990s, Microsoft proposed the “Stabilize and Synchronize” technique for iterative development, where team members synchronize their work frequently as designs evolve iteratively, and then periodically stabilize their design changes with user feedback to balance speed, flexibility, and discipline in work practices [12]. Principles of Extreme Programming (XP) are applied for making small incremental releases by synchronizing intermediate builds with daily test routines. These developmental releases are integrated into common builds at stabilization points where errors discovered during the integration process are fixed [16]. Testing is performed in controlled environments, such as in usability labs, by software teams to stabilize and synchronize evolving code base with incremental features before the software is released to the live or uncontrolled environment through phased alpha and beta releases. When external releases are made to the beta community, testing is performed by a wider user group on an ongoing basis with validation feedback on both functional and non-functional attributes [16]. Functional attributes are measured by analysis of data outputs with control flows between modules to verify internal and structural functionality with user specifications, while non-functional attributes identify aspects related to issues in performance (e.g., response time, maximum number of transactions), modifiability, reusability and reliability in the integrated experimental environment [17].

Boehm and Lane [18,19] have described the Incremental Commitment Model (ICM) that identifies three essential roles in the software development lifecycle, that is, the role of hardware, the role of software, and most importantly, the role of human factors or stakeholders involved in the development process. The authors of ICM advocate use of anchor point milestones or key decision points to assess the consistency, compatibility, and feasibility of software developed concurrently in multiple sites. Reviews performed at each milestone ensure concurrent activities are synchronized, stabilized, and risk-assessed at the end of each phase using stakeholder produced evidence. Further, developer-produced and expert-validated documents assist in making risk management plans to define core deliverables and any bottlenecks in schedules. In this manner, ongoing verification and validation methods are used during all phases of the development lifecycle for measuring performance, as stakeholders (i.e., development teams at concurrent sites, management teams, and user community) interact with each other to assess the incremental deliverable comprising both, hardware and software. Based upon the result of the joint review, the development teams build commitments across concurrent sites with accountability for their role in management of project scope and schedule of the software development process.

However, inconsistencies may occur in operational activities at various development stages due to non-coordination of individual lifecycle activities or due to shortfall in evidence supplied by stakeholders [19]. Proper governance practices to support decision makers and guide them in utilizing resources effectively for integrating incremental builds at various software development phases are advised to avoid unanticipated delays in schedules and minimize cost overruns in project realization.

3. Related work on software testing

Most research on software testing pertains to tactics used for verifying functionalities through black box testing, white box testing and usability testing [7,8,20]. Black box tactics describe technical processes used to write test scripts for analyzing boundary values of modular components, measuring overall performance and drawing cause-effect graphs. White box test strategies include practices for measuring code coverage (e.g., statement, decision, and condition) and analyzing class designs for dependencies and measuring their interrelatedness. Usability tests inspect work-domain environments to evaluate participant interactions by measuring navigation paths, detecting layout issues or quantifying hand movement counts on keyboards. Defect reports through error tracking tools are used to analyze qualitative data (e.g., analyzing participant’s reflections and opinions with task activity scenarios) and quantitative data (e.g., locations and type of defects, when they were found and fixed, names of developers and teams). Defect data are analysed via research tactics to inform management on quality and reliability issues [21,22]. Detailed mathematical and statistical methods are applied to correlate defect for development time estimation, prediction of severity of defect, and identification of implementation issues when a number of team members are involved in a project. Metrics such as defect density (the ratio between the number of defects and lines of code), irreproducible defect count (the irreproducible nature of reported defect), crash defect count or stability metric (number of times the system has crashed) help in gauging a sense of confidence in the software product before it is handed over for user acceptance testing [23].

Whilst these are essential elements of writing and verifying code during the software development process, the software platform on which the development environment is hosted is key to the whole process. Distributed teams collaborate over common project workspaces during the various stages of development and test strategies are executed on network enabled architectural layouts. Organizations create a shared environment with virtual private networks, Web servers, and utilize security protocols (e.g., https) to share evolving software artifacts with distributed members. Test strategies are deployed on software deliverables hosted on these environments through white box tests, black box tests, usability tests, and other integration tests. It is a common practice for organizations to create a separate computing environment called “staging” or “migration” or “deployment” environments for inspecting the overall completeness prior to ‘going live’ [9, p. 153]. Practice research on how organizations plan their test process and manage execution of tests (black box, white box, usability) in the staging environments is limited. The staging environment’s configuration needs planning in much detail considering issues associated with network authenticated logins, data management, and number of staged servers used during low and peak loads. The staging environment is context dependent that involves intellectual property of the organization and hence, to address the confidentiality and security concerns, organizations go through extensive planning for all of the staging components used. Many organizations set the staging environment identical to the production environment, and engage users for alpha and beta testing. Alpha tests typically involve testing by few customers and developers in a strictly controlled environment, while beta tests involve a larger group of customers in a less controlled environment. Due to tight project schedules, some defects discovered during alpha and beta tests go unrepaired and organizations often decide to defer the fix to future releases and inform customers about the current defects [21].

4. Distributed software development

With the recent phenomenal growth in communication media, centralized development of software is an activity of the past and is no longer valued by the software community. Distributed software development has become the norm, in which software developers interact in a virtual team environment and engage in online
discussion forums to jointly develop software artifacts. These developers may not have met each other, but are peers sharing development skills and domain expertise on a shared global domain rather than on an organizational platform. This has spurred growth in global markets in software development and restructured traditional organizational working styles from being localized or national, to being distributed or international [5]. The software development team is no longer isolated to a local office, but is situated across different geographical and often organizational boundaries [24]. Dispersion of an organization’s commercial and technical knowledge across international domains with offsite partners having different organizational allegiances raises new risks. These risks could be associated with conflict of interests that may arise among the collaborating partners belonging to different cultural and organizational groups situated in different spatial and temporal zones. Some identified risks associated with sharing of business knowledge during distributed development, particularly among internationally distributed software teams, relate to loss of control over intellectual property, threat of opportunistic behaviour by offsite teams, public relations mishaps, and having to deal with different legal systems [25,26]. Hence, to safeguard their knowledge portfolio from risky aspects of DSD, some organizations involve “selective outsourcing” of software projects to offshore teams belonging to third party business partners located in low wage economies [27,28]. Selective outsourcing involves transfer of non-core business activities to offsite teams, while core activities are retained by onsite teams belonging to client organizations. With selective knowledge dissemination, the risk of opportunistic behaviour from offsite business partners is reduced; however it does have an impact on practices used for integration of software artifacts built in different development domains. This demands organizations to develop capabilities to manage knowledge asymmetries such as apply appropriate control strategies to specific software development and maintenance project properties to allow an offsite team to work from detailed specifications (or even software design) without the need to fully understand the application domain background [29]. Additional knowledge transfer is likely to occur during the service delivery stage [30], as practices are put in place to assess performance of each deliverable with pre-existing artifacts in the application domain.

The importance of having a development and test strategy cannot be underestimated in a distributed software development environment. Communication of changing user requirements to all teams, separation of core and non-core activities aligned to specific development teams, updates on interface stubs for integrating modules developed at different sites, scheduling of regression tests, and overall management of work across different time zones are added issues to non-localized teams. Managements have extra responsibility to ensure that teams at both ends are committed and accountable for their product designs and interfacing requirements. Additionally, not having met the development team at the offsite location adds more uncertainty to accountability and responsibility perceptions of team members. An issue (fix) perceived as having very high priority may not be perceived with the same priority at another site. Furthermore, the establishment of the development process suitable to different time zones and cultural groups must be considered. Communication media must aid the team members, so that they participate in online discussions and do not feel isolated from the offsite team members. Finally, regression test practices such as running of selected test scripts (for existing modules) along with new test scripts (for changed modules) should be explicitly carried out across distributed team members [17] to reduce surprises during the later stages of development.

5. Research questions

This study utilizes a case study research design to capture the knowledge of professionals (specialists) and document the experiences of practice [31]. Case studies are appropriate for answering “how and why questions being asked about a contemporary set of events, over which the investigator has little or no control” [32, p. 9]. The study investigated how test strategies are executed in distributed software development environments for a health care organization (referred as HealthNZ).

Two research questions were posed in the study:

1. Which lifecycle model is considered suitable for managing distributed software development projects spread across New Zealand and India? Why?
2. How test strategies are built in the chosen lifecycle model for management of verification and validation processes in distributed software development environment?

6. Research settings

To answer the research questions, this study investigated the following case studies: (1) HealthNZ, a leading medical health care service provider in New Zealand and (2) Provider1, an Indian IT service provider engaged in contractual software development work in New Zealand.

1. **HealthNZ** is a medical services provider with headquarters in New Zealand. HealthNZ has built a proprietary commercial software product to service healthcare professionals (e.g., doctors, laboratories, healthcare providers, and claim offices) to enhance quality of patient care through electronic health records management and reporting. This is achieved by interfacing the application with laboratories and medical professionals, creating communication channels for informing medicine recalls and substitutes to local agencies, providing demographic analysis of immunizations to government institutions, and establishing audit tools to maintain data security and privacy. Their medical products are used by healthcare communities in many countries, namely New Zealand, Australia, Ireland, Singapore, and Pacific islands. The software product lines have evolved from a legacy system operating on a text based DOS platform to a sophisticated graphical user interface environment with digital imaging, graphs, and anatomical templates. The upgrades to the application domain are ongoing, as HealthNZ adopts further technological advancements to their existing product lines. However, HealthNZ is aware of confidentiality and privacy concern requirements for maintenance of medical data, and are very careful with the software revision process. At first, HealthNZ managed the updates of their product lines at their local development centre in Auckland. However, with emerging outsourcing scenario in IT sector, specifically software development, where many software firms in western countries have contracted out software development tasks to low wage countries (e.g., India, China, the Philippines) to increase their profit margins, HealthNZ too ventured in this direction. They first contracted software development tasks to an Indian provider (Provider1). Provider1 was selected for two reasons (1) Provider1 offered benefits of low cost, with operations in India and (2) Provider1 maintained a local presence through development centres in New Zealand (Auckland) and Australia (Melbourne). From 2001 to 2004, HealthNZ and Provider1 worked on upgrading the software product, where the main development work was done at the Auckland offices under supervision of HealthNZ’s project managers. Having gained
confidence in working with an offshore albeit locally situated provider. HealthNZ recently entered in a joint venture partnership with another software provider based in India (referred to as Provider2). In 2004, HealthNZ discontinued contractual work to Provider1 and started a new business contract with Provider2, where they also have 10% ownership. Currently, HealthNZ has about 100 employees based in Auckland of which 20 are software professionals, who interact with development teams located in Australia and India.

(2) Provider1 is an Indian organization engaged in contractual software development work for overseas client organizations. They have a main development centre in Vizag (India), but also have development centres in Auckland (New Zealand), Melbourne (Australia), and Dallas (United States). The development centre at Auckland has 20 developers who have come to work with offshore clients from the Vizag development centre with temporary 6 monthly work permits. On expiration of their work permits, Provider1 sends another team of developers to their Auckland centre to continue the servicing of offshore clients. Provider1 is an ISO 9001–2000 certified company with CMMI (level 3) accreditation from Software Engineering Institute, Carnegie Mellon.

Interviews were held with top and middle management members of HealthNZ and also Provider1 separately in their Auckland offices. The purpose of the interviews was to gain insight on how these organizations managed the software development lifecycle and test strategies for an already established product having a major market share in the Australasian region. Most of the interviews were conducted with HealthNZ staff to understand the risks and challenges perceived by them from distributed software development team members and live customers. Furthermore, how they aligned their software development lifecycles in view of these risks and challenges. Subsequently, separate interviews were conducted with Provider1. Provider1 offered details on how they work with clients based in New Zealand and Australia, without referring to HealthNZ or any other client names during interviews.

The next section discusses the interview data to explain HealthNZ’s perspectives and reasons for the chosen software development lifecycle. Interviews with Provider1 have made available another perspective on how third party providers support the software development process.

7. Case data

The empirical data from the two case sources are presented.

7.1. Provider1

Provider1 has undertaken many contractual software development projects in New Zealand and Australia. They prefer to assign their developers to work at client locations during the week and at their offices during the weekends. The senior manager explained this strategy: “We provide a dedicated resource and he [the developer] works as an extended arm of the client and so he gets well trained in the customer process and domain knowledge of the customer requirements. ….. this is both a knowledge strategy as well as a marketing strategy. ….. and helps to remove the exclusiveness in working styles”.

With certifications from international agencies (e.g., CMMI and ISO), Provider1 maintains stringent documentation of all projects undertaken by them. The project manager explained that “with certifications comes strict documentation” of all artifacts (software code, test scripts, data flow charts, etc.) which are generated during the development effort. These artifacts and associated documentation are all handed over to the client teams which also help both partners understand and be aware of common expectations regarding project scope, deadlines, and work priorities. The project manager at Provider1 added that most of their New Zealand clients maintain poor documentation, and hence are “very appreciative of [our] documentation processes”. Furthermore, Provider1 uses many sophisticated project management tools (e.g., Bynet) to integrate data communications across their four development offices. Virtual meetings are also held daily between development centres through VOIP tools (e.g., Skype) to discuss work issues within distributed team members.

7.2. HealthNZ

The medical solution provider has established its product lines in live environments of Australasia with incremental upgrades made to their product. The software development tasks are carried out concurrently at three locations – Auckland (New Zealand), Melbourne (Australia), and Vizag (India) – who interact over virtual platform. However, the main development tasks for upgrades and conversion of certain legacy modules into more maintainable software components are carried out in India by Provider2. The team at Provider2 interacts on a daily basis with the New Zealand development teams, while the Australian teams are concerned more with front-end tasks associated with existing product being used in the live commercial environment. The staff at HealthNZ explained that the software development tasks contracted to Provider2 (and also earlier to Provider1) is “reverse engineering and coding” of software modules, which were described in extensive details in their architectural designs. They further explained that only the technical knowledge are shared with offshore partners (India) such as preferred software platform, internal structure of modules, and functionalities, but the business domain knowledge is not. They are cautious of sending too much details of their business application domain to the Indian development centre due to the sensitive nature of their medical services. The management at HealthNZ maintain strict data discipline in handling and information reporting within their local office, in view of the confidentiality and security concerns of customers’ medical records, government agencies and third party groups (laboratories, claim offices). Customized configuration settings are in place to restrict access to authorized personnel only for confidential medical data. This is monitored with restricted access using authorized logins. The project manager explained: “Our clients are doctors in Australia and New Zealand, and we deal with their confidential data. So, we must have a plan before we can delegate work at the operational level. When a person joins us, we train him on our PDP [product development process], and he works with a senior team and learns about our processes. We believe in having strict policies in place mainly because of the confidential nature of our data”. Thus, HealthNZ uses ‘selective outsourcing’, with core business practices confined to teams at New Zealand and Australia and contracts out routine coding or non-core tasks to Provider2 in India. The development team at Provider2 is a committed resource for HealthNZ and does not work on any other project for other clients. The managing director of HealthNZ voiced: “the deal was that they [Provider2] would not work for anyone else, so they have dedicated a part of their resource for us alone. These developers cannot work on any other project except ours”. A deed of non-disclosure has also been signed with Provider2, to protect the proprietary and commercial information of HealthNZ.

Earlier experience with Provider1 at their local Auckland site had resulted in many upgrades in software modules and streamlined documentation at HealthNZ. A developer at HealthNZ explained the extensive use of documentation during the development process: “We manage our work by proper documentation, by knowledge transfer, and by having an appropriate induction
program – and in this way you have much more rigorous knowledge continuity progress like knowledge bases. We plan properly so that looking at knowledge bases is now a routine job. When people understand them as a routine, then they are able to use them”.

HealthNZ described their lifecycle model similar to Microsoft’s strategy of Synchronize and Stabilize with verification and validation of small releases in development centres. Upgrades to existing software meant that they have identified many stabilization points or milestones in their PDP process. Virtual teams at distributed locations interact daily to synchronize the incremental releases, and the product build (e.g., dll file) is tested for defects during the synchronization process. The seven-hour time difference between New Zealand and India is used to extend the overall working time from 8 h to 15 h each day, with 4 h of time overlap between HealthNZ and Provider2. During the synchronization time, the software module is re-designed with new components, as the product goes through iterations of design, code, build and test. The evolving modules are validated by using white box tests and black box tests to verify functional requirements by both teams. Once the project is considered to be synchronized to a satisfactory level, the team at HealthNZ starts the stabilization process. The project manager explained “the project deadlines are set by us. During the stabilization process, we look at bug fixing and overall release process very seriously. We look at functional tests, regression tests, and load tests. The progress reports are sent to Australia too, and we decide on user acceptance test together”.

The user acceptance tests at stabilization points are performed over a server containing test data. In the “stabilization window” load balancing and integration tests are conducted. Stubs that emulate functionality of core components are used to combine modules and detect interface incompatibility errors. Several iterations of synchronize-and-stabilize cycles occur during the development phase in which white box tests are incrementally conducted on modules developed at distributed sites. More comprehensive black box tests are further conducted in a planned manner between modules using different test scenarios for verifying completeness. Testability is built into the design during synchronize stages of component development. The project manager explained HealthNZ’s software upgrade lifecycle process as shown in Fig. 1.

Some of the test reports generated are time value maps (active and inactive server time), resource utilization matrices (memory cycle usage analysis), pair-wise validity (boundary value checks between two components), and end-to-end validity (boundary value checks when all components are integrated together). The test data reports are sent to the senior management at both Australia and New Zealand for final agreement before the user acceptance tests are signed off. The management analyzes the overall defect reports, and makes a decision if the product is ready for beta release by performing pilot tests in another server supporting the staging environment.

Due to the highly sensitive client data, the HealthNZ management does not share its database interfacing details with offshore development teams in India. Accordingly, they have set up a specialized development environment with VPN (virtual private network), test servers, test databases, and tracking tools. The managing director explained “For a project plan we are very transparent. Each one of us knows the project plan for each one of us. We also have a weekly update. There is a daily update in terms of issues and we have a development tracking tool called Event Track which we have bought and have synchronized between each country for our medical applications. EvTrack is a great tool and both teams work out their plans smoothly. The project deadlines are set by us. The progress reports are sent to us and we have a team in house both in Australia and New Zealand. We do keep a local presence here and do not send all work to be done offshore. The local presence fixes and sees to the release process. They look at acceptance testing, bug fixing, and load balancing, and they communicate with Provider2. Later, our team here will deploy the product on another server, as we do not give Provider2 access to the other server”. White papers on the tool used by HealthNZ (i.e., EventTrack) have revealed that the tool gives out details such as information access rights, printouts made by which user logins of documents labelled as sensitive content, and monitoring of replication events on the database. These results could be monitored separately on different servers placed in the cluster. EventTrack can also be used to make detailed monitoring criteria for each server hosting a configuration database, maintain logbooks of accesses made, and also define events considered to be sensitive in the database.

HealthNZ have deployed a test database on a shared server for verification checks in controlled environments which are jointly monitored by all virtual teams. Software tools are installed and configured to identify risk scenarios by monitoring developers’ activities for all of the three sites in the development environment. After the software has been verified with test data on the shared

Fig. 1. Synchronize and stabilize lifecycle model.
server, HealthNZ releases the modules for further validation testing to be performed in a staging environment.

HealthNZ is currently exploring new markets for further expansion, and are planning to enter the medical services markets in Ireland. Accordingly, they have set up a staging environment in Ireland with separate servers for medical professionals and agencies. The managing director stated that with their successful product lines, they “plan to enter more overseas markets, and are currently exploring the user requirements with a few select customers in Ireland. Provider2 does not have access to the pilot [staging] environment at all”. The New Zealand and Australian teams along with a small team of developers at Ireland are managing the work flows in the staging environment and do not include the teams from Provider2 for data security reasons. The test reports from pilot implementation are monitored by the New Zealand team, and any reported errors are escalated to the offsite team located in India for further action. Fig. 2 synthesizes the development environment used across HealthNZ’s DSD environment.

As shown in Fig. 2, all logins to the development environment go through HealthNZ’s VPN to ensure that authorized users enter the right environment. The VPN is equipped with security policies in place to protect sensitive customer data through management of three server configurations (i.e., test server, pilot server, and production server). After compliance checks on incremental releases through a customized “Synchronize and Stabilize” lifecycle model, the use of drivers and interfacing stub supports are incorporated between the components to establish different environments for test, staging, and production through the application server. The application server also triggers the EventTrack tool to monitor all of the user activities. The exposure of the EventTrack application is to maintain a log of regular activities and to report any untoward activity. Notifications are programmed in database settings to inform administrators of any attempt to access the confidential files and at what time and by which user logins. On-site project managers in charge of distributed teams act as facilitators who supervise the virtual environment through inspections of documents (e.g., test reports and system logs), and coordinate the related transaction activities between teams.

8. Discussion

Business risks are perceived more in distributed global markets, such as offshore software development environments where development team members are situated in different cultural, temporal, and social zones, and may not have met each other. However, cost benefits offered by development of software in low wage countries (e.g., India and China) can no longer be ignored, and organizations are adapting their business models to take advantage of low cost contracts with software organizations in these economies. This study has identified benefits such as extending each working day from 8 h to 15 h where teams at different sites interact over rich communication media in different time zones to cumulatively add more working hours for each day. Moreover, to streamline work processes and improve performance across distributed sites, organizations are enforcing standards and documentation to bring consistency and clarity in interpretations of designs made by teams at another site. Tools are used to measure progress of activities against some set standards and make organizations aware of whether current activities are satisfactory. These tools eventually are formalized by management to assess overall performance starting with measurements of simple infrastructural elements (artifacts) until the final deployment of the finished product [33]. Documentation tools such as templates, coding standards, screen shots, sample reports, and defined metrics are being used by management and development teams to ascertain the quality and progress of artifacts generated. Cataldo et al. [6] support the view that documentation in DSD helps managements resolve issues in the software evolution process. They state that architectural documents helps project management teams identify dependencies among modules early in the development process and accordingly, analyze task allocation to remote teams to minimize dependencies and bottlenecks in later stages of development. Furthermore, documentation provides static testing of system documents for correctness, completeness, and accuracy in which computer execution is not always required [9].

New business models are emerging as organizations are entering into distributed software development. Traditional styles
of developing software by one particular organization are being replaced by partnerships across diverse economies to bring together sharing of skill sets and domain knowledge. However, to safeguard their business position and commercial knowledge portfolio from opportunistic business partners, organizations are cautious of what they should share and how much they should share. Good risk mitigation strategies are applied as organizations evaluate the pros and cons of what should and should not be contracted to third party business partners. This is more of a concern if issues of data privacy and confidentiality are key aspects of the business application domain. Apprehensions of information leakage and breach of intellectual property are raised. Accordingly, organizations demarcate core activities which cannot be contracted out and routine activities which can be contracted out. The case described in this study deals with sensitive data, and has contracted out routine or ‘coding’ tasks to third party offshore business partner in India. Furthermore, they are making small incremental changes to an already existing product, which has evolved from legacy systems. Yu and Mishra [5] suggest to minimize risks during software development of projects involving continuous evolution and restructuring in distributed environments, teams should (1) apply component-based software development methods, (2) increase standardization of modeling practices and programming concepts, and (3) use standardized project management tools. This view is supported by the health care provider in this study. This study further emphasizes use of a prescribed lifecycle approach, with key anchor point milestones or stabilization points, which provide reassurance on project scope, schedule, and progress. Fig. 1 shows how software development evolves with synchronization of related activities performed by distributed teams using a modular or component-based approach with standardized practices. This is followed by each iteration of development undergoing many stabilization points involving user acceptance tests before it is released for further testing in a staging or pilot environment.

After product development stages are over and before the product is officially launched in the market, organizations seek to gain further reassurance about the usability of the product in a real environment with different configurations from a small number of friendly customers through beta testing. The idea of beta testing is to obtain feedback from a limited number of customers who commit a considerable amount of time and thought to their evaluation [23]. Beta test execution provides useful information to software development teams with regard to number and frequency of defects and offers new insight on possible scenarios where the software functionality cannot be guaranteed to work. Beta tests are opportune for development teams to identify deficiencies and find a workaround solution to ensure that such failures and defects do not occur in live environments where the number of customers using the system is unknown. This study has shown how the health provider organization has established a beta test referred to them as “use a pilot” to investigate product quality before it is deployed in a production environment. The sensitive nature of medical data raises concerns of data privacy and user confidentiality with third party business partners. The study reveals that organizations have set up an elaborate staging environment with secure tracking tools to perform beta tests. Defects identified with pilot data in the staging environment are replicated with test data in test environment and evaluated by development teams for resolution before the final release. Load tests are performed to measure resource utilization, interoperability issues with third party products, and performance under high load (i.e., more network traffic) and average load (i.e., intended network traffic). Data obtained from execution of these tests provide information on overall performance of services offered by the software product and assist in defining sustainable interoperability standards for meeting the dynamics of global markets. For instance, configuration settings are optimized for non-functional requirements to make reliable, stable, and robust product lines which can withstand heavy loads. System crashes are investigated to inform on robustness of software designs and determine reliability and scalability of the evolving product. Further, decisions are also made on future adaptations needed for subsequent releases (e.g., software patches, updated versions) for existing product lines.

9. Conclusion

This case study investigates the business model used by software groups in the evolving IT industry as organizations are moving from localized development to distributed software development markets. The study captures test strategies and experiences used by a global case study organization involving software development by distributed virtual teams. It also highlights the perceptions of risk by the case organization and explains what actions have been taken to mitigate the perceived risk without compromising on user confidentiality and quality. The use of the customized Synchronize and Stabilize lifecycle model demonstrates how test strategies are defined and responsibility shared across onsite and offsite locations. The chosen lifecycle model (Synchronize and Stabilize) has built-in sequence of verification and validation checks involving many stakeholders, who are together accountable for pass/fail milestone criteria at each incremental stage.

Testing studies on knowledge dispersion and integration in the context of distributed software development have been analysed to understand how test practices are embedded in diverse organizational settings. Additionally, the research recognizes the support of tools to create secure workspaces and restrict access of clients’ confidential data to key personnel. Tracking tools are embedded in shared workspaces to monitor development and test activities by distributed teams during the different lifecycle stages of the software development.

Current lifecycle process models such as V model emphasize on mitigating risks by being flexible and adapting to the evolvability of software through iterative development. However, little practice research has been reported on how organizations manage their software development lifecycles, more so, in distributed settings. This paper provides insight on how organizations define test strategies to support the software evolution process. Many iterations of formal and informal testing are executed during the lifecycle stages involving distributed and multiplatform environments to verify alignment of evolving software with standard specifications. Test simulation environments are set up based upon high-level policies to safeguard the organization’s commercial knowledge and intellectual property. However, a number of challenges remain, as each domain specific test strategy is based on its current context and cannot be applied as a universal strategy. This leads to further research direction of evaluating practice methods used by organizations to mitigate risks during software evolution in different business contexts.

References

onshoring, in: Proceedings of the Conference Name, Conference Location, Conference Date, Publisher, Year of Conference, pp. Pages.


Anuradha Mathrani is a Senior Lecturer in Information Technology within the School of Engineering and Advanced Technology, Massey University, Auckland, New Zealand. She holds an Engineering degree in Electronics and Telecommunications, a Master’s degree in Management Sciences, and a PhD in Information Technology. Anuradha has a rich industry experience, having worked in research and development laboratories of reputed multinational organisations before joining academia. Her research interests include software application lifecycle management, software assessment and governance methods, quality and reliability measurements, and development practices in distributed software teams. Anuradha has published several research papers in refereed journals, book chapters, and national/international conferences. She can be contacted at: A.S.Mathrani@massey.ac.nz.

Dr. Sanjay Mathrani is a Senior Lecturer in the School of Engineering and Advanced Technology at Massey University, Auckland, New Zealand. He holds degrees in Bachelor of Technology in Mechanical Engineering, Masters in Management Sciences, and a PhD in Information Technology from Massey University. He has more than twenty years of product development, manufacturing and global supply chain experience and has held senior positions in various multinationals namely Greaves, Alstom, and Navman. A qualified auditor of quality management systems to ISO 9001:2000, he has assisted organisations in developing quality management practices and implementing process improvement strategies for ISO accreditation endeavours. He has been a practitioner of ERP and business intelligence systems in hi-tech engineering operations and is a chartered professional engineer to New Zealand industry. His research interests are in information and knowledge management, product development, quality, and manufacturing operations, and enterprise service-oriented architectures. He has published several papers in international journals, books, and conferences and is an invited speaker at various international universities. Dr. Mathrani can be contacted at: S.Mathrani@massey.ac.nz.