Increasing Understanding of the Modern Testing Perspective in Software Product Development Projects

Maaret Pyhäjärvi
Conformiq Software Ltd.
Stella Terra | Lars Sonckin kaari 16,FIN-02600 Espoo, FINLAND
Maaret.Pyhajarvi@conformiq.com

Kristian Rautiainen and Juha Itkonen
Helsinki University of Technology
Software Business and Engineering Institute
POB 9600, FIN-02015 HUT, FINLAND
firstname.lastname@hut.fi

Abstract

Testing can be difficult to integrate into software development. Approaches to software testing in relation to implementing software are based on the V-model of testing. The software process behind the V-model is the traditional waterfall model, and as such the traditional testing approaches cannot take iterative, incremental and agile approaches to developing software into account well enough.

In this paper, we describe the use of a general iterative and incremental framework defined for controlling product development—4CC—from a modern testing perspective. The framework provides a common language in which the implementation details and pacing as well as testing details and pacing in software product development projects can be communicated. Viewing testing through a general iterative and incremental framework adds to understanding how the testing process should be defined and improved in relation to the software development process. Additionally, best practices for testing are identified.

1. Introduction

The importance of quality assurance (QA) and software testing is recognized in SME's as well as in larger organizations. These activities are an integral part of the software development and releasing a product to the market, and should be included in the software development project from the beginning. However, QA and testing can be difficult to integrate into the software development. They are easily left to just occur at the end of the development project, especially if the resources are scarce, and the pressure is on time-to-market and all effort is focused on implementation.

The essential difference in QA and testing, as understood by testing professionals, is the attitude. When assuring quality, you are building in quality and assuring that the required level of quality is achieved. QA goes more easily hand in hand with development as the approach in both is constructive. In testing, the mindset used is destructive as the goal of testing is to find errors. It has been argued [15] that the attitude towards seeing defects is essential for success in finding the defects and it is the core of modern testing approaches. Traditionally, testing is defined in a narrow sense as "execution of a program in the intent of finding errors" [15]. The modern definition of testing, more easily adopted by people viewing themselves as test professionals, defines testing as "the process of planning, preparation and measuring aimed at establishing the characteristics of an information system and demonstrating the difference between actual and required status" [18]. In the wider context testing and QA activities are converging, QA taking more of a process improvement perspective and testing being part of QA. Both QA and testing are seen as activities starting right from the beginning of the project. In this paper, the focus is on understanding the testing perspective, since in practice—especially in small organizations—testing as a means of finding defects would be the part to start QA related activities from.

In larger organizations, QA and testing are often responsibilities that are organizationally separate from the actual development. Testing is organized as a sub-project within the development project. In a small company testing needs to be more integrated to the development process as there is not a separate testing group due to limited resources. Testing activities are conducted by the same people doing all other tasks, with mere change of role. This is not, however, the same as the developers testing their own code; some level of independence in testing is also aimed for in a small company.

If development and testing within development are separated responsibilities, the project manager would choose a development model applicable for the situation at hand. The project manager quite often understands that testing is important and should be included in the project,

but as the responsibility of testing details is given to a test or quality manager, the lack of testing detail in software process models is emphasized. The test manager bases his testing approach on the V-model as it is the state-of-the-art in testing. The V-model in its turn is based on the waterfall model and thus is difficult to use in communicating both the iterative and incremental nature of a project as well as the need of rework as defects are found in an environment where an incremental software development lifecycle is used.

In the ongoing SEMS (Software Engineering Management System) research project we are studying software engineering in small companies in the software product business. The need to understand and improve the testing process has become evident in the interviews and case work conducted with our pilot companies. When taking the iterative and incremental approach to software development and the V-model of testing, we noticed significant difficulties in understanding the details of the testing perspective in the overall picture without testing-specific expertise. The problems in the use of the V-model have been recognized in the testing community as well [9].

In this article, we discuss the changes needed so that the modern testing perspective could better be understood by all roles involved in a software product development project. We do this using the 4CC framework [20], which provides a structure through which the roles involved can more easily communicate.

2. Using the V-model – Why is it Not Enough?

In software process research, many different software process models have been suggested. Starting with the *code-and-fix* model, adding structure by splitting the process to sequential tasks to form the *waterfall* model [22], noticing the need and cost of change within a development project, resulting in models such as the *spiral* model [4] and different other *iterative and incremental* models, e.g. [5;8]. Latest ones in the field are so called *agile* process models [3;7;23], basing their agility on short increments and intense customer collaboration. Software development models acknowledge testing as an integral activity, but cannot give testers much detail on how to structure their work.

Looking specifically at testing in the perspective of a test manager, testing needs a model that is focused on driving the testing-specific efforts. In situations where it is not applicable to have a subproject for testing and a separate test manager for the subproject, the project manager needs to consider testing in more detail in relation to implementation details. The project manager needs to understand expectations and dependencies between implementation and testing to successfully control the whole project.

In the field of software testing, the V-model is the state-of-the-art taught on practically every course on testing. The V-model—presented in Figure 1—splits the testing process onto levels on which testing is carried out incrementally in conjunction with system implementation. The V-model starts from the smallest pieces possible for testing and moves on to larger pieces, reflecting the different viewpoints of testing in different levels of detail.

Notice that the flow of abstraction in testing is reverse to the flow in implementation where the custom is to start from high abstractions and move towards more and more concrete details. The reason for starting the testing from individual modules (and not, for instance, from user requirements) is the organization of labor. It is much easier to find and fix defects in small units than in large entities, and the testing of large entities can be carried out more systematically if it is known that their sub-units have already been tested. Planning testing should, however, flow in the same order as implementation. The V-model implicitly shows how the testing phase can—and should—be taken into account much before there is some source code to actually be tested.

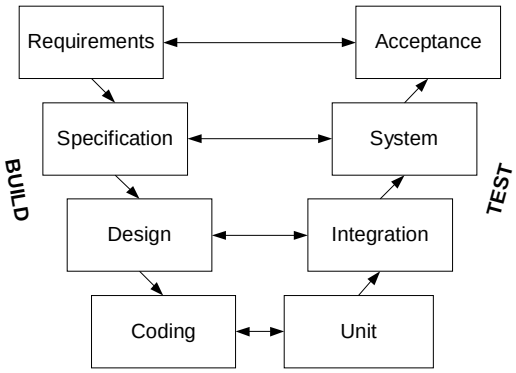

Figure 1 V-model of Testing

The V-model is an extension of the simple waterfall model, where each process phase concerned with implementation has an associated verification and validation phase called *test level*. From a testing point of view, testing on each level should be planned and controlled to avoid overlapping. Traditionally the individual test plans for the test levels are seen as the links between these activities, coordinated with a master test plan.

The V-model is intuitive and easy to explain, even to people who have never heard of a software development process model. This may be the case when persons with specific domain expertise in the use of the system are needed in testing. The V-model essentially brings forth

two important points: testing a smaller part before putting it into a larger system is a good approach and testing efforts can start with planning as soon as the higher level requirements have been identified. The V-model provides a common terminology for testing in the form of test levels.

The V-model as a basis for testing actitivities has been strongly criticized in [14]. The V-model, just as any testing model created as an extension to software development models, ignores the fact that software is developed in a series of handoffs, where each handoff changes the behavior of the previous handoff. The models tend to rely on the existence, accuracy, completeness, and timeliness of development documentation. They assert that a test is designed from a single document, without being modified by later or earlier documents, or assert that tests derived from a single document are all executed together. For example, finishing all module testing prior to moving to integration testing may be implied in the V-model steps, but that is not a good approach.

The V-model as such looks like a tidy process, but communicates change poorly. This is due to the built-in waterfall model assumption. A less experienced test manager may assume that the implementation-related documentation to base testing on is more finalized than it is in practice at time of starting test planning. It may be insufficiently communicated which parts, e.g. requirements, are more finalized than others. This leads to focusing the already scarce testing resources based on outdated information to unproductive work. If implementation documentation has not matured prior to defining test cases to base test execution on, it may not be a good approach to write detailed test cases. Due to changes later on during the project, the test cases could need considerable rework. It has been suggested that the V-model's early test planning approach would help programmers to avoid defects by using detailed test cases testers have written based on first versions of documentation [9]. However, reviews and inspections are likely to be more efficient in order to help correct defects early than relying on pre-writing tests that will never be run [9]. Furthermore, testing is supposed to find defects, and finding a defect may put one back to the requirements definition phase. Defects also need to be verified and corrected, and the software tested for regression, still not re-executing all test cases, which might take too long a time.

Testing activities in the V-model take a document-driven approach not always feasible in practice. For small, co-located teams with little change in team composition the need for documentation is smaller than for large and distributed teams or teams with high staff turnaround. In all cases, the documentation produced should serve an actual need and the need should show in keeping the documentation up-to-date. Especially in agile software process model context, the detailed implementation documentation plays a smaller role but testing still has its place and can take place with the lesser amount of documents.

The V-model tends to emphasize verification (are we building the product right?). However—especially for product business—emphasis on validation (are we building the right product?) in testing has grown [9;24] and testing needs to assess both perspectives.

A manager experienced in organizing testing does not organize testing efforts the way that the V-model may suggest if interpreted strictly. However, it has been our experience that the expertise to avoid the pitfalls may take time to form. Using the V-model as a basis for defining a testing process may create an inflexible process to a place where agile or incremental approaches would be more appropriate. Testing literature and courses mostly rely on the V-model and even imply that the waterfall method would be the most current lifecycle model [12]. However, incremental development is an increasingly popular mode of development [13] and needs to be addressed also in a testing context [6;21]. Basing testing on the expectations set by the V-model in such a context is difficult. Still, the V-model forms the essential basis for any testing activities taking place. Therefore the test manager needs experience on different process models and their implications to testing in order to be able to apply the V-model wisely, usually skipping all details except emphasis on early test planning and the need of test levels. The V-model as development model fits situations in which changes must be managed, for example perhaps with situations in which a fixed cost project is undertaken and any change requests from the customer will carry a price tag.

Understanding how tests should be grown and how constant regression testing is organized between builds is a challenge in practice. Communicating all this requires a more dynamic approach.

3. Using the 4CC from a Testing Perspective

In efforts to understand software product development and how to control it, a framework for managing software product development was introduced, called 4CC (Four Cycles of Control) [20]. With the limitations in the V-model as described above, we suggest that the modern testing perspective can better be communicated through the 4CC framework, which emphasizes pacing that sets the basis for all testing activities, and provides a structure through which the roles involved can more easily communicate. The test levels are continuous flows of activities that need to be structured through setting up a rhythm.

In ongoing research, we are focusing on small software product companies, and working on understanding the

connections of the company's business model (for more information on business models see [19]) to its software product development process. In that context 4CC has been introduced and it is continuously developed in co-operation with Finnish software companies to better understand its operational aspects. 4CC is a high-level iterative and incremental framework, and as such can be applied in many contexts, but our efforts to bring detail to it have been focused on the small product company perspective. The key idea in our research is that different software companies produce different kinds of products for different customer groups, and the approach for creating software should fit the company's business model, and take into account the influences of product perspective and team size. By understanding the possibilities and constraints set on the product development process by the business model, software process improvement can be focused on the essentials from the business perspective and thus improve product quality and profitability.

Testing is one perspective emphasized in our research of small product businesses, as it is viewed as an important area with many challenges in practice by both the researchers and the pilot companies involved in the research. Bringing together development and testing perspectives in product development in a small company context poses challenges, as the traditional approaches of separate test groups presented in testing literature are not applicable as such. We need more thorough understanding of the reasons why the suggestions have been given in order to scope them to a small company context.

We view testing in the broader perspective of maximizing customer satisfaction and providing feedback for process refinement, in addition to just detecting and getting defects corrected in the software. The testing process needs to be examined together with the overall project and product management processes of the firm. Testing activities include planning, management, implementation and support needed from a tester's perspective. Information flow and pacing are important for testing activities. Testing by executing a program needs the program to be implemented to some extent. Test case design relies on having information on the features to be implemented.

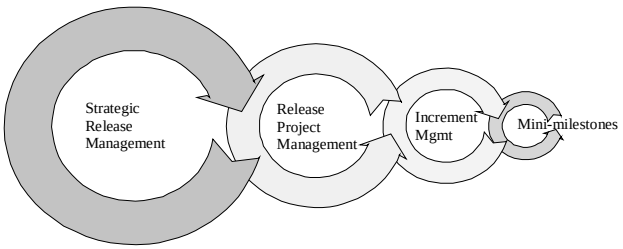

Figure 2 The 4CC Framework

As discussed above in section 2, the details of testing provided in software process models do not help testers to understand their role in relation to the process. A tester's role is to find and report defects and verify that the reported defects have been resolved, either by a programmer fixing them or by management deciding that they will not be fixed for some reason. The 4CC framework helps in understanding testing in relation to other software development activities. It sets four timeframes on which one needs to address certain issues in development. The timeframes—depicted as cycles—are presented in Figure 2. The leftmost cycle, named *Strategic Release Management*, deals with the release project portfolio and is the interface between business management and product development deciding on all ongoing major activities requiring attention from product development. *Release Project Management* deals with issues on the level of individual projects aiming for a product release. *Increment Management* deals with managing individual increments producing a part of a release project's deliverables. *Mini-milestones* deal with structuring and pacing the daily work for different roles participating in the product realization process. Different cycles provide different levels of abstraction to facilitate control and flexibility.

The 4CC model adds an important perspective for testing compared to the V-model. The V-model focuses on a single project and as such, naturally leaves out essential co-operation between projects. Projects following each other in time could, especially in product business, benefit a lot from the results and lessons learned from previous projects. Projects ongoing simultaneously could be managed together for more efficient use of testing resources. It is important to see testing related activities in projects as a portfolio from which all ongoing projects can benefit from through reuse and experiences.

In the testing community, a so called multiple V-model has been applied by consultants in an iterative and incremental context, showing that for testing the number of deliverables to base testing on increases. Using the multiple V-model one draws a V for each iteration and shows time as the horizontal axis. The added detail depicts writing the documents that testing is based on in smaller pieces, but resulting in a presentation that is difficult to communicate and understand and shares the limitations of the V-model.

On the project level, the test lifecycle depicted by a test level is too simplistic to provide support for actual work that is based on small handoffs. The test levels in the V-model depicts in iterative and incremental context a testing effort that start in the beginning of the project with planning and proceed to execution as soon as a part has been implemented. The need for managing all testing levels separately depends on the process used. In extreme programming, two testing levels are applied: unit testing

and acceptance testing, with the first combining goals of the two lower levels and latter combining goals of the two higher levels. In systems with challenging integration the integration level may be managed separately.

Figure 3 describes how testing work is split in iterative and incremental development to various testing levels. The figure depicts relationships between modules A-E and how these are collected together for higher testing levels. Test execution starts as soon as the first modules are implemented. New handoffs which correspond to mini-milestone cycles of the 4CC model trigger new testing activities. Modules A and B are unit tested prior to integrating them together. In integration testing, the focus is on verifying that the added module works together with the current version of the whole system. In system testing, the whole system is verified to planned extent, with focus on the whole system, not just the latest addition.

There are few points that need to be stressed in Figure 3. Implementation and testing of modules D and E are depicted to be separate but partly overlapping. In integration testing, these are presented to be integrated to the latest available baseline that was completed at the time of starting the module. In system testing, the modules D and E are not brought to the system separately, but as a group. A typical situation would be that the two modules are created by separate developers individually. Acceptance testing level is typically the final level of tests, but it essentially is also an ongoing activity. The different levels need to be managed as a whole to avoid unnecessary rework—each level focuses on testing different aspects as described with the V-model. Typically, the different levels in testing would apply different test environments. Rework due to regression testing takes place on the test levels. Change in the tested modules results in need of retesting through the whole pipe effectively.

The test levels just as the V-model defining the levels have their roots in the project business. In project business the acceptance test level is emphasized as it is the customer's perspective in verifying that the software developed fulfills the customer's needs. The essence of acceptance testing is that it is the final testing prior to accepting the software and it should be characterized by relatively small number of defects. The focus on acceptance test level is on fulfilling the customer needs and found defects should be related to that. Essentially at the end of acceptance test, the test is the final check before moving the system into production. In product business the role of acceptance test level is two-fold. First of all, it stresses the user perspective, both usability and applicability, throughout the development. Secondly, it is the final checks that are made for releases.

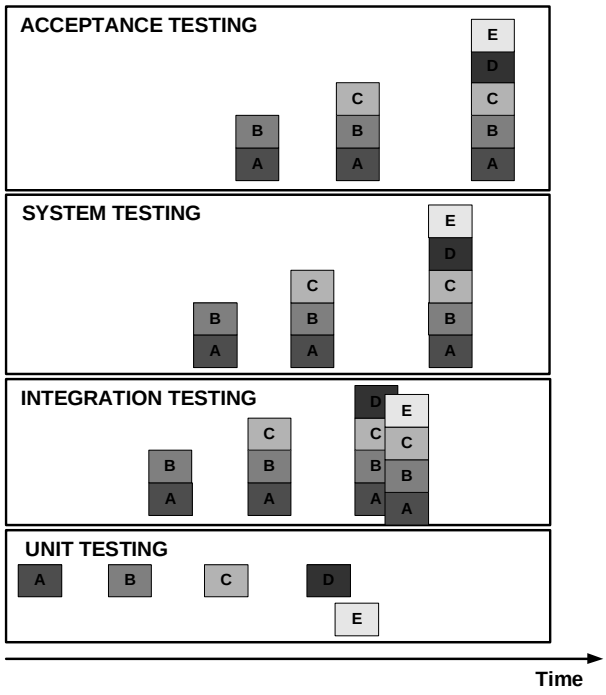

Figure 3 Test Levels in Iterative and Incremental Development

Understanding the modern testing perspective of test levels depicted in Figure 3 results in noting that the system to be tested grows all the time. Very soon in the development, it becomes impossible to re-execute all defined tests on one build, but the tests need to be split on various builds over time. As the pile in the figure grows, managing the testing effort focuses on creation of test suites—collections of test cases—and prioritizing them, as controlling individual tests would result in detail that may distract the overall view on control.

Testing is essentially about feedback to implementation. Testing needs to be managed based on small handoffs, building a larger whole. Testing should be reactive to handoffs. Thus many testing details are best communicated on the mini-milestone level as the daily reaction options. These reactions need to be synchronized to the organization's pacing as well as the developers' pacing. Managing testing in a project needs to build the proper relationship between control and flexibity. How this rhythm has been included in the Microsoft's synch-and-stabilize model is discussed in the following section by describing synch-and-stabilize in 4CC.

4. Modern Testing Best-Practices and Synch-and-Stabilize Testing

Understanding the pacing of development is essential for successful testing. To better understand the modern testing perspective and its implications in managing testing in projects, we have identified the best practices in

modern testing to characterize how testing could be included in the product development lifecycle within the 4CC framework. As a tangible example of the ability to add testing detail in 4CC we have dissected Microsoft's testing approach in the Synch-and-Stabilize process as defined in [5]. A growing trend in the field of testing is a so-called context-driven school of testing, asserting that there are only good practices in context, but no best practices [9]. However, the practices presented here are starting points to tailor the approach for test management in a specific context instead of directly applying assertations in the V-model.

The modern testing perspective can be characterized by its best practices. We have identified five main best practice areas, each with several details, from recent testing literature [10] and mirrored these to the operational test management approaches in case companies[9;11;12;16-18]. The best practices selected are based on the case work conducted at pilot companies. They represent common ways of integrating testing into a software project. *Basing testing on product and business risk* is a main driver behind test efforts. Testing should be based on product and business risks, as exhaustive testing is not feasible as the number of combinations to verify in a non-trivial program is very high. Testing needs to take into account the changing risks as technology and market matures, and needs agility planned in the testing process so that adjusting is possible. The most important risks for the product from the user's perspective should be addressed first. This should be visible both in prioritizing test cases and executing them so that high-priority tests will be run first, as well as in prioritizing different hardware and software platform combinations testing will be conducted on, as all combinations are not possible to test.

Destructive attitude drives the testing effort as the main goal of testing is to find defects as early as possible to facilitate timely release with aimed quality level. In order to include the destructive attitude, there is a need for independence in testing, as one tends to be unable to see one's own mistakes. As defects are fixed, new rounds of previously executed tests need to be executed to find defects that have been caused by defect fixes, which happens easily as the complexity of the code makes it difficult to anticipate all dependencies.

Early involvement of all test levels is important. Each developed feature needs to be tested on all levels from unit to acceptance and the different levels exist concurrently and continuously throughout the project. Reviews and inspections are a part of testing as they help in noticing the defects early. Testing needs to have an emphasis on validation in addition to verification as creation of defect-free software that no one will use is not worth the effort—the software needs to be validated that the features it provides are the ones that the users are willing to use. Testing does not necessarily need to be document-driven, the need for documents as a basis for testing depends on the context.

Pacing test activities is important in order to be able to understand and control testing activities on very short cycles. Testing activities are highly dependent on other project activities and thus the need for agility is high. Testing efforts need to be split over time and on different builds, as execution of all test cases on a certain build is not applicable [2]. The number of test cases is likely to be large and all test cases take significant time to execute. Found defects need to be fixed concurrently with test execution proceeding and corrections released to re-testing. The risk of many corrections integrated into a build after having waited for test round on a previous build to finish increases the risk of noticing side effects of defect corrections late. The test environment should exist separately from the developers environment and should change only in the agreed pace. Daily rhythm through knowing the rules of the game as dependencies and deadlines facilitates efficient testing.

Traceability and maintainability includes ways of connecting testing to requirements as well as considerations on the ability to maintain and grow the tests. Test cases should be grouped into test suites of different priorities, different functionalities and different uses (e.g. smoke test, regression test) to facilitate control. A traceability matrix between the test cases and requirements should be kept up-to-date in order to know if tests need to be updated, as well as what tests need to be updated, to the changing requirements. A light-weight approach to defining test cases is needed. Test case definition should focus on requirements that have matured to the level that they will actually be implemented. The number of test cases needs to be minimized and focus of tests addressed. Quality of testing determines the quality of evaluation on product quality and needs to be assessed continuously. Test reporting should be done on a regular basis but carry only the necessary overhead. The number of test environments is increasing. Testing important user environments is spread over the course of the project and should be focused on selected environments based on environment risk-based priorities. It is important to be able to connect the testing performed, the time the testing was performed and the environments testing was performed on, if e.g. the customer base changes and thus the priorities of environment change.

Looking at Synch-and-Stabilize as defined and detailed in [5], an emphasis on testing in the product development process is evident. Redefinition of Microsoft's development process resulting in definition of synch-and-stabilize started with a "zero-defects memo", pointing out the costs of defects to Microsoft's customers. In synch-and-stabilize, testing exists as a separate function with dedicated testers, integrated into the everyday product

implementation, paced by daily builds. The aim is to find defects as early as possible.

The goal of early test involvement included in the V-model is realized well as testing is integrated to software implementation from the beginning and participates as its own function in planning and scheduling the work. Different levels of testing are also applied, all taking place concurrently and continuously, feature by feature. Unit testing is not formalized and is seen as the testing conducted by the programmer. However, developer testing responsibilities include testing their own features running automated tests created by the testers frequently, usually on a daily basis. Programmers are paired with so called "buddy testers", and these testers do integration testing with the help of the programmer on the feature on a private release before the feature is released to system testing. Buddy testers form a testing organization for the project, coordinated by test leads and test managers that are responsible for system testing iteratively and incrementally. System testing examines the product from six perspectives, namely user perspective, international perspective, hardware compatibility, software compatibility, specification compliance and product stability. Acceptance testing at Microsoft includes usability testing to verify each feature, and beta releases that are employed to better understand dependencies of different customer hardware and software platforms and defects manifesting only in some of these platform combinations. Testing on different levels is not document-driven as the V-model might suggest. The product is outlined in a product vision, its features are written down in a functional specification as the project goes on and technical details are documented in the code using comments and a common style of coding.

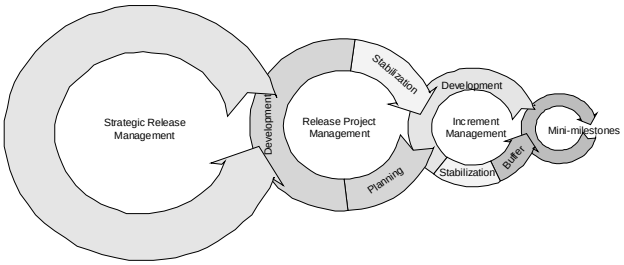

Figure 4 The 4CC Framework with Synch-and-stabilize Details

All test levels are concurrent and continuous activities, and can be better understood by looking at the project on several levels through the 4CC framework presented in Figure 4. On the Strategic Release Management cycle, Microsoft has twice a year highest level scheduling of rolling out new products and setting their budget. Once a year Microsoft updates its three-year product plans and their interdependencies. They use monthly project status reports to highest level management and related projects. Program reviews are conducted quarterly for each project.

The Release Project Management cycle is structured to three themes or phases. The project begins with planning, is continued by 3-4 development subcycles and finished with a stabilization phase. On the Increment Management cycle each of the project phases have different kinds of tasks. Planning focuses on setting a project vision, a skeletal functional specification that sets areas and subareas for features and thus facilitate early identification of test suites, and a master schedule, including testing activities estimated by people conducting the actual work later on. Next 3-4 increments each build and test a selected prioritized set of functionalities of the product. The final increment's theme is stabilization, which includes testing the product as a whole, and finding and fixing defects. The stabilization phase goes hand in hand with beta testing, if one is employed. Increments at Microsoft employ so called buffer time for unexpected delays. The Increment Management cycle is also structured to three phases. The development phase is concerned with the teams developing the deliverables for the increment. At the end, the increment is stabilized to required quality and buffer time is reserved as a contingency for unanticipated problems.

On a mini-milestone level, a tester chooses his task based on the dependent activities in implementation. When preparing for new testing, testers do general reviews on previous project's postmortem reports and reports from other testing groups, talk with product support personnel and customers, review media evaluations, devise special tools or code routines to help them test, study competitor products for new features, develop testing strategy by identifying high-risk areas, and review each other's plans and scripts for completeness. Developers find more of their own defects than testers do, and only developers can prevent errors from happening in the first place. Code that is assumed difficult and code that is produced by new people is reviewed by senior developers. If new functionality has been coded and is to be integrated to the public release, the tester focuses on testing the private release of the "buddy developer". After code has been integrated in the public release, testers execute tests and track defects found in the test release and characterize them by feature area and severity. If coding is ongoing, the tester may focus on defining tests for the functionality as well as automating tests. Online user documentation is tested just as the program itself. If the coding activity is defect-fixing, the tester focuses on verifying fixes as they become available in daily builds. On a weekly basis, a subset of tests is executed on a debug build with testability features helping defect location. System tests go on continuously on daily builds. Pacing of testers

activities is dependent on the programmer's activities. On the other hand, rules apply to the other direction as well. For some products, a rule on having 10 critical open bugs means interrupting development of new features until the critical bugs have been resolved to a level below the agreed limit.

The number of testers Microsoft uses is significantly more than what is traditionally the tester-developer ratio. Especially it is more than what is possible in the small company context. At Microsoft, this has been a compromise in being able to change direction based on market inputs as needed. The number of testers could be reduced if more upfront planning was introduced—more time on architectural planning and detailed design work—or if developers would be made to review their own code more. Reducing the number of testers would reduce the amount of flexibility in evolving features or components incrementally. Testers are deemed relatively inexpensive compared to the cost of recalling and replacing products because of major defects.

5. Managerial Implications

A classic problem in testing is the difficulty of communicating with the project manager when you are taking the role of a test manager [1]. Expectations differ in used process model, produced documentation and readiness level of the documentation at a point in time. This is, at least to some extent, due to the different models applied by the two perspectives. Another reason suggested has been the project manager's lack of knowledge in testing details [10].

In this paper, we have described the use of a general iterative and incremental framework defined for controlling product development—4CC—from a modern testing perspective. The framework provides a common language in which the implementation details and pacing as well as the testing details and pacing can be presented. Based on our experience with our pilot companies, viewing both implementation and testing activities in the 4CC framework helps in understanding dependencies between activities and the scope of time the activity is related to.

Our research focus is small companies and within that context, we have identified best practices for testing. The best practices are summarized in Table 1. The best practices have been selected from testing literature based on our case experience on what kind of approaches are possible in the small product company context. However, the same best practices can be seen in Microsoft's Synch-and-Stabilize development model and its testing details. Essential in synch-and-stabilize is the pacing set for development, facilitating communication and co-operation between implementation and testing. Microsoft's approach employs a 1:1 tester-developer ratio, which is not feasible for a small company. However, the software a small company is developing is probably not the size of Microsoft's products either and the size and complexity are issues to consider for different instantiations of the best practices.

Table 1 Modern Testing Best Practices

Best Practice Area	Details
Basing testing on product and business risk	Gaining understanding of changing product and business risks Prioritizing test cases Prioritizing test environments Testing in order of priority
Destructive attitude	Goal of finding defects Need of independence
Early involvement of all test levels	Reviews and inspections as a means of finding defects early Emphasis of both verification and validation All test levels take place concurrently and continuously Need of document-driven testing needs to be assessed
Pacing test activities	Dependence on other activities Rework due to defect corrections Splitting test cases to builds over time Identifying daily tester tasks Controlled test environment with releases to testing
Traceability and maintainability	Grouping tests into test suites Use of a traceability matrix Light-weight just-in-time approach to writing test cases Test reporting on defined internal releases Scheduling test suites to different environments

The framework described has been created keeping the product development context in mind. However, the three lower cycles describe pacing of a project and could be applied in understanding pacing of projects in project business as well. Lately, the 4CC model has been applied to structure testing in companies other than small as well as other than those in product business. The 4CC framework helps in structuring the complex testing effort on several levels of abstraction, reminding of the connection between these levels. It sets a common vocabulary in the pacing of the development efforts and helps in communicating different kinds of handoffs and their rhythm. Especially in definition and communication of a test strategy 4CC has been effective. Understanding

the forms in which the test strategy presents itself on different levels has helped in defining it.

4CC is a framework for controlling work. It helps in describing logistics in a project for testing as well as other development-related work. The Test Management Approach (TMap) model [18] describes testing in four dimensions: lifecycle, organization, techniques and infrastructure. It has a weakness in its lifecycle as the lifecycle assumes waterfall-like approach. Our experiences point out that the added detail in the pacing in 4CC that is critical for success enables communicating the modern testing perspective better. The other cornerstones, namely organization, techniques and infrastructure, include important testing specific details that are placed on the lifecycle.

Pacing testing in relation to implementation is important. Even though it may seem that there is a lot of change to manage, a word of warning on an approach we have seen in practice. In some cases there has been a daily build cycle but testers use biweekly builds to be able to execute all tests on one build. However, the implementation proceeds meanwhile and even though the full round of tests have been executed, the latest build has changed significantly and would need to be tested for regression. This is one manifestation of interpreting the V-model's levels strictly.

The V-model supports individual projects. With 4CC we depict that it is important to also manage a portfolio of projects. The software product lifecycle outlasts boundaries of projects and when planning for testing, it is important to consider if there would be synergies between the separate testing activities in projects.

6. Discussion and Further Research

We have presented the use of a framework for managing software product development in small companies to increase understanding of the modern testing perspective in software product development projects. The framework and its details are still tentative and our ongoing research both adds detail to it and collects empirical data in using the described details in piloting companies. Test process definition and improvement research work continues and the best practices identified are further tested in companies. To better support testing, 4CC needs to be instantiated to detail in our case companies. The goals of the testing model needed include [14]:

- Force a testing reaction to every code handoff in the project.
- Require the test planner to take explicit, accountable action in response to dropped handoffs, new handoffs, and changes to the contents of handoffs.
- Explicitly encourage the use of sources of information other than project documentation during test design.
- Allow the test effort to be degraded by poor or late project documentation, but prevent it from being blocked entirely.
- Allow individual tests to be designed using information combined from various sources.
- Allow tests to be redesigned as new sources of information appear.
- Include feedback loops so that test design takes into account what's learned by running tests.
- Allow testers to consider the possible savings of deferring test execution.
- Allow tests of a component to be executed before the component is fully assembled

These are also important things for managers to consider when tailoring the best practices in Table 1 to a project-specific or company-specific instantiation.

Test improvement models such as TPI (Test Process Improvement) are attempting to phase test improvement. These models have their roots in CMM and base their testing approach in the V-model, which we argue is not sufficient. Viewing testing through a general iterative and incremental framework adds to understanding how the testing process should be defined and improved in relation to the software process. We are basing the test process on iterative and incremental as well as agile software development processes but also the waterfall model would be applicable as a special case of a project with only one increment. We have conducted a benchmark of 15 Finnish software development organizations' testing with the TPI model and within this benchmark, reflected the results to the 4CC. The results of this benchmarking are currently under processing.

To help with business-focused process improvement and practice selection, we are working on evaluating software development processes from the perspective of business fit and a business-dependent path for software process improvement from the basics. We are also collecting a set of tools to support the instantiation of the framework in companies.

The details in 4CC are focused on small product companies. However, the overall idea of pacing within any project (3 lower cycles) applies just as well. The applicability has been tried in practice at Conformiq Software Ltd.

References

[1] Bach, J., "James Bach on Explaining Testing to Them. Helping Non-testers Understand and Support Your Work," *Software Testing & QUality Engineering*, vol. 3, no. 6, 2001.

[2] Bays, M., *Software Release Methodology*, Prentice-Hall PTR, 1999.

[3] Beck, K., *Extreme Programming Explained*, Addison-Wesley, 2000.

[4] Boehm, B., "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, vol. 21, no. 5, 1988, pp. 61-72.

[5] Cusumano, M. A. and Selby, R. W., *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, Simon & Schuster Inc, 1998.

[6] Cusumano, M. A. and Yoffie, D. B., "Software Development on Internet Time," *IEEE Computer*, vol. 32, no. 10, 1999, pp. 60-69.

[7] Highsmith, I. J., *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House Publishing, 2000.

[8] Jacobson, I., Booch, G., and Rumbaugh, J., *The Unified Software Development Process*, Addison Wesley Longman, Inc., 1999.

[9] Kaner, C., Bach, J., and Pettichord, B., *Lesson Learned in Software Testing - A Context-driven Approach*, Wiley Computer Publishing, 2002.

[10] Kaner, C., Falk, J., and Nguyen, H. Q., *Testing Computer Software*, 2 ed., John Wiley & Sons Inc., 1999.

[11] Kit, E., *Software Testing in the Real World*, Addison-Wesley, 1995.

[12] Koomen, T. and Pol, M., *Test Process Improvement: A Practical Step-by-step Guide to Structured Testing*, ACM Press, 1999.

[13] Marco, I. and MacCormack, A., "Developing Products on Internet Time," *Harvard Business Review*, vol. 75, no. 5, 1997.

[14] Marick, Brian, "New Models for Test Development," *Proceedings of Quality Week 1999*, 1999)

[15] Myers, G., *The Art of Software Testing*, John Wiley & Sons, New York, 1979.

[16] Patton.R., *Software Testing*, Sams Publishing, 2001.

[17] Perry, W., *Effective Methods for Software Testing*, Wiley, 1995.

[18] Pol, M., Teunissen, R., and van Veenendaal, E., *Software Testing - A guide to the TMAP Approach*, Addison-Wesley, 2002.

[19] Rajala, R., Rossi, M., Tuunainen, V., and Korri, S., *Software Business Models: A Framework for Analysing Software Industry*, Tekes, Technology Review 108/2001, 2001.

[20] Rautiainen, K., Lassenius, C., and Sulonen, R., "4CC: A Framework for Managing Software Product Development," *Engineering Management Journal*, vol. 14, no. 2, 2002, pp. 27-32.

[21] Redmill, F., *Software projects: Evolutionary Vs. Big-Bang Delivery*, John Wiley & Sons, New York, 1997.

[22] Royce, W. W., "Managing the Development of Large Software Systems," *Proceedings of Wescon*, 1970, pp. 1-9.

[23] Schwaber, K. and Beedle, M., *Agile Software Development with Scrum*, Prentice Hall, 2002.

[24] Weinberg, G. M., *The Psychology of Computer Programming*, Van Nostrand Reinhold, New York, 1971.