Project no. 644235

REPHRASE

Research & Innovation Action (RIA)
REFACTORING PARALLEL HETEROGENEOUS RESOURCE-AWARE APPLICATIONS – A SOFTWARE ENGINEERING APPROACH

Initial report on applicability of the RePhrase software engineering methodology to the commonly used software development methods
D5.4

Due date of deliverable: April 30^{th}, 2017

Start date of project: April 1^{st}, 2015

Type: Deliverable
WP number: WP5

Responsible institution: UNITO
Editor and editor's address: Marco Aldinucci, UNITO

Version 0.1

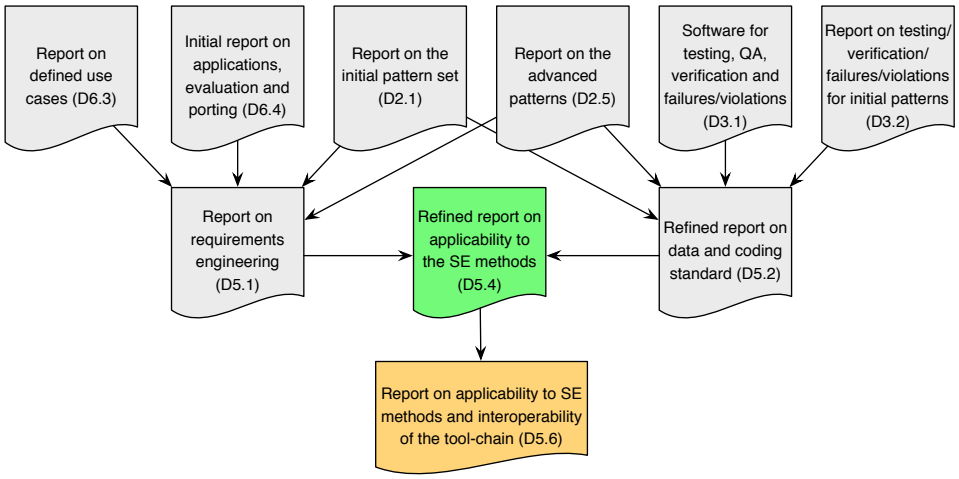
Project co-funded by the European Commission within the Horizon 2020 Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Change Log

Rev.	Date	Who	Site	What
1	15/05/17	Marco Aldinucci	UNITO	Summary, structure and Chapter 1 and 2 done
2	5/06/17	Evgueni Kolossov	PRQA	Added PRQA development model description
3	6/06/17	Javier Garcia-Blas	UC3M	Added CIbersam and UC3M development model description
4	6/06/17	Zsalt Szepessy	evopro	Added evopro development model description
5	10/06/17	Marco Aldinucci	UNITO	Text reviewed, references improved
6	12/06/17	Michael Vinov	IBM	Added IBM development model description
7	12/06/17	Javier Garcia-Blas	UC3M	Fixed encountered typos.
8	62/06/17	Michael Rossbory	SCCH	Added SCCH development model description
9	18/06/17	Marco Aldinucci	UNITO	Cleaning

Executive Summary

In this deliverable, we review different software development models and we make a survey of methodologies and tools currently used by projects' industrial and academic partners. Eventually, we report an initial assessment of the methodology developed within the project to support these development models and their sustainability. In summary, this deliverable reports on the applicability of the RePhrase methodology to commonly used software development methods. Dependencies of the present deliverable D5.4 are reported in the following diagram.



Contents

Chapter 1

Introduction

With the emergence of multi-and many-core systems, software engineering for parallel programming will become a key challenge for next generation systems. The critical issues that will have to be addressed are developing high-level programming models (such as pattern-based programming), tools for automatic or semi-automatic parallelisation of the source (such as refactoring), and automatic monitoring of and tuning application performance. RePhrase aims to produce an integrated software engineering environment that will address all of these issues. Although different software development methods use different approaches to developing software, all of them consider essentially the same development phases.

1. *Requirement Analysis* involves identifying, in conjunction with customers, what an application should do.

2. The *Design phase* involves taking a list of requirements and producing a model of the application that meets these requirements.

3. *Implementation and Debugging* is the part of the development where the actual code is written.

4. *Testing,Verification, and Analysis* deals with ensuring that the code is free of bugs; that it meets the requirements identified during the requirement analysis; and analysing static and dynamic properties of the code in order to discover potential problems.

5. *Deployment* is the process of distributing the software to run on possibly different platforms/architectures.

6. *Maintenance and Evolution* is the process of making an application meet the requirements even if they change.

In the next section, we briefly recap some of the most commonly used software development methods, also referring to the phases above.

Chapter 2

A Survey of Development Models

The most commonly used development models in data intensive applications:

- In a *Waterfall* model, the order of the phases of the software development is followed strictly, and a phase is started only after all of the activities of the previous phase have been finished, and all the deliverables accomplished. Revisiting and revising previous phases is highly discouraged.

- In *Incremental development*, the development starts with initially small, but progressively larger portions of a project (incremental). The software design process is repeated with each new portion of a project. In this way, the hope is that key issues and the overall architecture are discovered early on in the development process.

- *Rapid application development* focuses on rapid prototyping of software at the expense of planing. Planning is interleaved with developing software, which enables dynamic changing of requirements.

- The *Spiral* model is a meta-model that combines any other lifecycle model, but also emphasises risk analysis.

- In the *Code and fix* method, programmers immediately begin producing code, usually omitting the design phase. Testing is left to the end of the development cycle.

- *Agile development* is based on iterative development, but advocates a more people-centric approach to the development, where continuous feedback from customers is incorporated into iterative process in order to suc- cessfully re- fine and improve software. Variants of agile development include Extreme Programming, Dynamic Systems Development and the Scrum method.

2.1 Waterfall model

The waterfall model, firstly formalised by Walker W. Royce in 1970 [13], is a sequential development approach, in which development steadily flows downwards through several phases, like a waterfall.

The basic principles of waterfall development are [7]:

1. Requirements analysis resulting in a software requirements specification;
2. Software design;
3. Implementation;
4. Testing;
5. Integration of subsystems (if any)
6. Installation;
7. Maintenance.

The waterfall model has been widely consider impractical in the case requirements are subject of frequent changes. If any error is occurred in current and previous phase then developer have to be correct it immediately that leads to less chance of error with final product.

2.2 Incremental development

Incremental development aims at mitigating the effect of errors on final product by nesting iteration with a linear model. Various methods are acceptable for combining linear and iterative systems development methodologies, with the primary objective of each being to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.

There are three main variants of incremental development [7]:

1. A series of mini-waterfalls are performed, where all phases of the waterfall are completed for a small part of a system, before proceeding to the next increment;
2. Overall requirements are defined before proceeding to evolutionary, mini-waterfall development of individual increments of a system;
3. The initial software concept, requirements analysis, and design of architecture and system core are defined via waterfall, followed by incremental implementation, which culminates in installing the final version, a working system.

2.3 Rapid application development

Rapid application development (RAD) is a software development methodology, which favors both iterative development and the rapid construction of prototypes instead of large amounts of up-front planning. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.

The rapid development process, introduced by James Martin in 1991 [10], starts with the development of preliminary data models and business process models using structured techniques. Then, requirements are verified using prototyping, finally to refine the data and process models. These stages are iterated; further development results in a combined business requirements and technical design statement to be used for constructing new systems [17].

The basic principles of rapid application development are [7]:

1. Key objective is for fast development and delivery of a high quality system at a relatively low investment cost.

2. Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.

3. Aims to produce high quality systems quickly, primarily via iterative Prototyping (at any stage of development), active user involvement, and computerized development tools.

4. Key emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance.

5. Project control involves prioritizing development and defining delivery deadlines or "timeboxes". If the project starts to slip, emphasis is on reducing requirements to fit the timebox, not in increasing the deadline.

6. Generally includes joint application design, where users are intensely involved in system design.

7. Active user involvement is imperative.

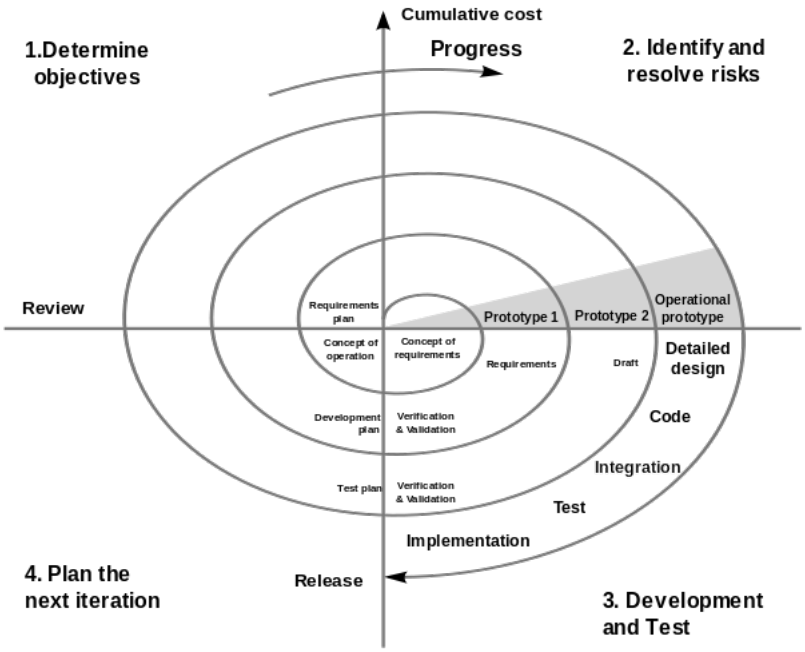
8. Iteratively produces production software, as opposed to a throwaway prototype.

9. Produces documentation necessary to facilitate future development and maintenance.

10. Standard systems analysis and design methods can be fitted into this framework.

2.4 Spiral model

The spiral model, firstly proposed by Barry Boehm in 1988 [5], it combines some aspects of the waterfall model and rapid prototyping methodologies, in an effort to combine advantages of top-down and bottom-up concepts. It provided emphasis in a key area many felt had been neglected by other methodologies: deliberate iterative risk analysis, particularly suited to large-scale complex systems. The spiral model as described in [6]:



The basic principles of the Spiral model are [7]:

1. Focus is on risk assessment and on minimizing project risk by breaking a project into smaller segments and providing more ease-of-change during the development process, as well as providing the opportunity to evaluate risks and weigh consideration of project continuation throughout the life cycle.

2. Each cycle involves a progression through the same sequence of steps, for each part of the product and for each of its levels of elaboration, from an overall concept-of-operation document down to the coding of each individual program.

3. Each trip around the spiral traverses four basic quadrants: (1) determine objectives, alternatives, and constraints of the iteration; evaluate alternatives; Identify and resolve risks; (3) develop and verify deliverables from the iteration; and (4) plan the next iteration.

4. Begin each cycle with an identification of stakeholders and their "win conditions", and end each cycle with review and commitment.

2.5 Agile development

Agile software development describes a set of principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams. It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change. These principles support the definition and continuing evolution of many software development methods. The term agile was adopted by the authors of the Manifesto for Agile Software Development (often referred to as the Agile Manifesto for short) [4].

The Manifesto for Agile Software Development is based on twelve principles:

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Working software is delivered frequently (weeks rather than months).
4. Close, daily cooperation between business people and developers.
5. Projects are built around motivated individuals, who should be trusted.
6. Face-to-face conversation is the best form of communication (co-location).
7. Working software is the principal measure of progress.
8. Sustainable development, able to maintain a constant pace.
9. Continuous attention to technical excellence and good design.
10. Simplicity —the art of maximizing the amount of work not done— is essential.
11. Best architectures, requirements, and designs emerge from self-organizing teams.
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly.

Agile software development methods support a broad range of the software development life cycles. Some focus on the practices (e.g., Extreme programming, Agile modeling), while some focus on managing the flow of work (e.g., Scrum, Kanban). Some support activities for requirements specification and development (e.g., Feature-driven development), while some seek to cover the full development life cycle (e.g. Dynamic systems development method). Popular agile software development frameworks include:

- *Adaptive software development (ASD)* replaces the traditional waterfall cycle with a repeating series of speculate, collaborate, and learn cycles. This dynamic cycle provides for continuous learning and adaptation to the emergent state of the project. The characteristics of an ASD life cycle are that it is mission focused, feature based, iterative, timeboxed, risk driven, and change tolerant [9].

- *Agile modeling* is a methodology for modeling and documenting software systems based on best practices. It is a collection of values and principles, that can be applied on an (agile) software development project. This methodology is more flexible than traditional modeling methods, making it a better fit in a fast changing environment [1].

- *Disciplined agile delivery (DAD)* is a process decision framework that enables simplified process decisions around incremental and iterative solution delivery. DAD builds on the many practices espoused by advocates of agile software development, including Scrum, agile modeling, lean software development, and others [2].

- *Extreme programming (XP)* advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted. Other elements of extreme programming include: programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, code simplicity and clarity, expecting changes in the customer's requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers. The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels [8].

- *Feature-driven development (FDD)* blends a number of industry-recognized best practices into a cohesive whole. These practices are all driven from a client-valued functionality (feature) perspective. Its main purpose is to deliver tangible, working software repeatedly in a timely manner [11].

- *Kanban* is a method for managing knowledge work which balances demands for work with the available capacity for new work. Work items are visualized to give participants a view of progress and process, from task definition to customer delivery. Team members "pull" work as capacity permits, rather than work being "pushed" into the process when requested. In software development, for example, Kanban provides a visual process-management system which aids decision-making about what, when and how much to produce. Although the method originated in software development and IT, it may be applied to any professional service whose work outcome is intangible rather than physical [3].

- *Scrum* is an iterative and incremental agile software development framework for managing product development. It defines a flexible, holistic product development strategy where a development team works as a unit to reach a common goal, challenges assumptions of the traditional, sequential approach to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved. A key principle of Scrum is the dual recognition that customers will change their minds about what they want or need (often called requirements volatility) and that there will be unpredictable challenges, for which a predictive or planned approach is not suited. As such, Scrum adopts an evidence-based empirical approach, accepting that the problem cannot be fully understood or defined up front, and instead focusing on how to maximize the team's ability to deliver quickly, to respond to emerging requirements, and to adapt to evolving technologies and changes in market conditions [15, 16].

- *Dynamic systems development method (DSDM)* is an agile project delivery framework, initially used as a software development method, then revised and became a generic approach to project management and solution delivery rather than being focused specifically on software development. The DSDM Agile Project Framework is an iterative and incremental approach including continuous user/customer involvement. It fixes costs, quality and time at the outset and uses the MoSCoW prioritisation of scope into *musts*, *shoulds*, *coulds* and won't haves to adjust the project deliverable to meet the stated time constraint [12].

2.6 Code and fix method

Code and fix is an anti-pattern. Development is not done through a deliberate strategy or methodology. It is often the result of schedule pressure on the software development team. Without much of a design in the way, programmers immediately begin producing code. At some point, testing begins (often late in the development cycle), and the unavoidable bugs must then be fixed - or at least, the most important ones must be fixed - before the product can be shipped. Programmers have autonomy over the development process. This includes control of the project's schedule, languages, algorithms, tools, frameworks and coding style.

Chapter 3

Development models currently used by RePhrase industrial partners

In this section the background knowledge of **RePhrase** on development models is briefly described. This is of specific interest of industrial partners of **RePhrase**, which are: IBM, SCCH, EviPro, and Cybersam.

3.1 IBM

We believe all of the software development methodologies described in this document have been used by various teams across IBM. The Agile software development model has recently become one of the major development methodologies for IBM software projects. IBM is probably one of the largest practitioners of this development practice. Two technologies developed and used by the **RePhrase** project – ExpliSat and Focus – are developed based on this software model. Agile development focuses on developing software in iterations of usually two weeks. For our development we have used longer iterations of 1.5 months. The Agile development approach allows us for more flexibility than the traditional waterfall methodologies.

A key factor is that IBM helps customers and its own teams with applying agile complex environments, facing such issues as application complexity, scalability, and compliance issues.

There are many development approaches that can help implement a major Agile objective of continuous delivery. Examples include test driven development, extreme programming, the use of containers, microservices, behavior driven development, design for testability, design for failure, social coding and component based development. Many of these approaches are complimentary to each other and are most effective when used together in an overall framework.

Our development process is based on very basic Agile principles such as: think, practice, focus on the outcome, collaborate and be innovative. The major goal of this process is to shorten development cycle and increase customer value. Benefits of applying this approach are highly appreciated by the development teams as well as by our customers. The benefits are improved code quality, more frequent releases and hence more satisfied customers, fewer mistakes.

3.2 SCCH

The main type of projects conducted at SCCH are research projects. Key characteristics of this kind of projects are the high complexity, novelty, volatility and unpredictability. Additionally there is often a close collaboration with the development department of our customers. Constantly intermediate results are reported, reviewed and the next steps planned. For these reasons traditional project management processes like V-model or waterfall model do not fit for this kind of projects. Instead an iterative, agile method is needed.

Figure 3.1 depicts the project development process implemented at SCCH. It is an iterative approach based on SCRUM including specialisation for the needs of research projects.

The management process consists of the following phases:

- **Research Problem Analysis** At the very beginning of the project or even before start the overall project vision has to clarified with all stakeholders. The result of this first phase is description of the coarse projects goals that will we refined through feedback loops in later iterations.

- **State of the Art Analysis** Since novelty is a key characteristic of research projects, state of the art analysis is an important step at the beginning of the project. The result of this phase is a description of the state of the art, the new development will be built on.

- **Iteration Planning** This phase consists of two tasks: Problem analysis and task planning. During problem analysis the project manager and stakeholders identify requirements and problems and prioritize them. The result is a requirements catalogue that is documented in the project backlog and a risk plan. Actual task planing is done by the project manager and the project members. Items are selected from the project backlog by priority, refined to fit into the iteration and effort for these items is estimated. Usual iterations are 2 - 4 weeks long.

- **Iteration Retrospective** The retrospective of an iteration is done by the project manager and the researchers working on the project. Positive and negative aspects of an iteration are analysed to continuously improve the iteration planning process.

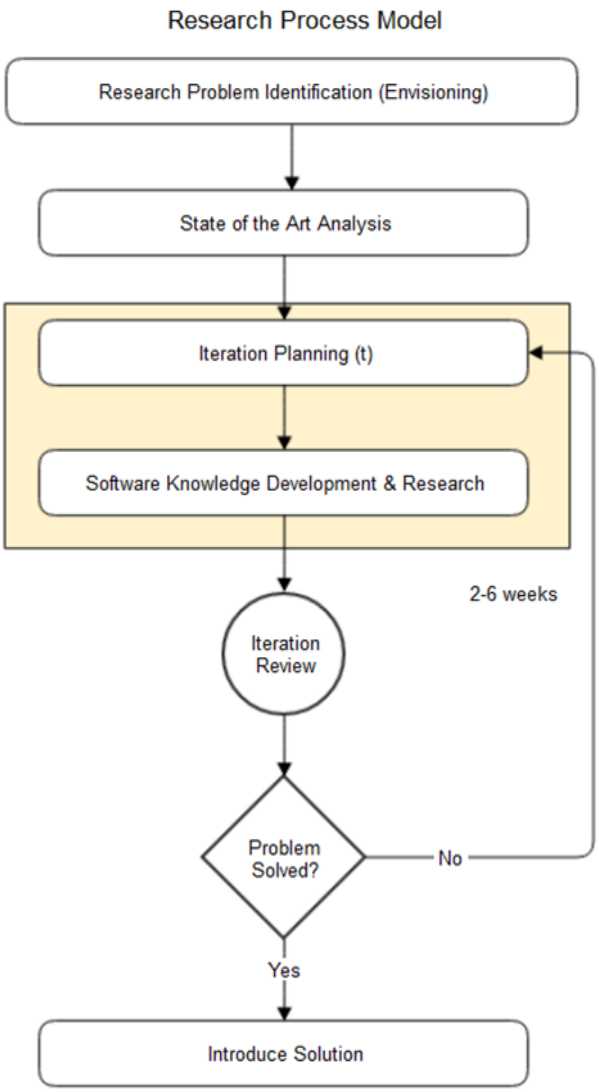


Figure 3.1: Research Project Management Process

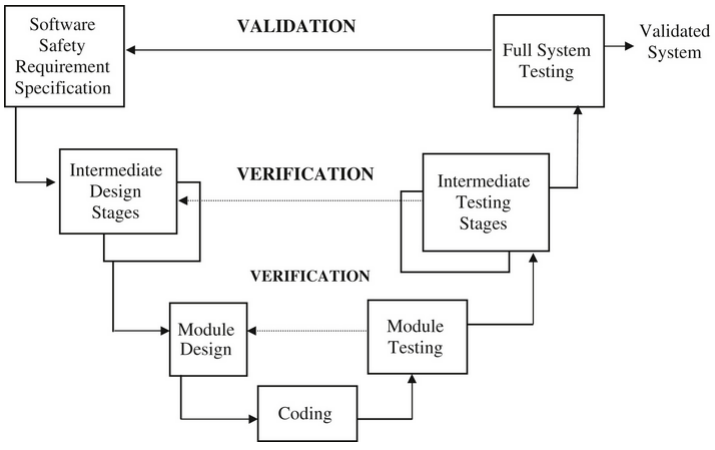
- **Iteration Review** During iteration review the project manager presents the iteration results to the project stakeholders, usually during the planning meeting for the upcoming iteration.
- **Introduce Solution** If the problem identified at the beginning of the project has been solved to the customers satisfaction knowledge is transferred to the customer and the prototype or pre-product is introduced at customers site. An important task within a research project is to write publications about the research done in the project. If not already done during the project results should be published in this task phase.

Parallelisation is a fundamental aspect that has to be considered from the beginning of the project, starting already in the requirements capturing phase as described in D5.1. In the project development process coarse parallelisation aspects have to be treated from the first phase, the research problem analysis, together with the customer, since adding parallelisation later in the project needs much more additional effort. Identified needs have to be refined in early project iterations based on the requirements.

3.3 EvoPro

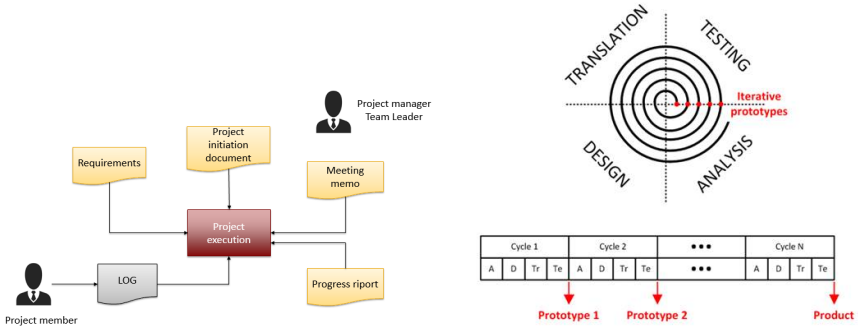
evopro Innovation uses 3 types of project management process for software development, they are:

1. V model: large projects for large organizations, when software project is part of a larger project or Functional Safety is required (IEC61508). In the traditional V model the process covers the following stages



2. Iterative process: this process is used when the specification and the requirements are still detailed but the management expect changes during implementations or the implementation has steps or tasks which are challenging,

have risks or tasks the team is not familiar with. In this process the software product is created as a consecutive series of prototypes. The process is built of stages. Each stage results in a prototype. Each phase contain the four steps: Analysis, Design, Translation, Testing. We used to utilize the PRINCE 2 document structure and corresponding management procedures, but it was too heavyweight for our practice, thus a simplified set of documents was defined. These documents are: Project Initiation Document, Requirements (SysML models), Meeting Memorandum List, Progress Report and Project Log. The Project Log contains Events, Risks, Issues, Changes, and Actions.



3. Scrum. The Scrum is used in mobile application development, loosely specified software projects when customer is not sure or confident int he detailed specification. The applied Scrum process is built around the Team of Product Owner (usually delegated by customer), Scrum Master and team members. The process is organized in Sprint of 2 weeks. Each Sprint starts with a Sprint Meeting of 4 hours. The product backlog contains the items that the team has to implement. Each item is specified by a one-page storycard. The progress is monitored by the compound burndown chart. We offer a transparent monitoring of the status for our customers.

evopro Innovation also implements the following meta-models for evaluating, comparing, and improving the adopted processes:

- Quality Management: ISO9001
- System Engineering: ISO15288
- System and Software Engineering: ISO12207
- Requirement Engineering: IEEE830

3.4 PRL

As described already in this document there are many methodologies that can be used for software development. Agile is the latest one and growing very quickly

because it concentrating on delivery of software on time and budget. But even in Agile there are many flavours. When PRL have been selecting the methodology we have been discussing all of them. To select the right methodology we need to understand the following:

1. Agile is not a *process* but a *methodology*. To implement Agile successfully the *process* needs to be created around the *methodology*.
2. For different types of projects different Agile flavours are favourable.
3. Different teams prefer different Agile flavours.

So, the main challenge has been to:

1. Implement the process which can support different Agile flavours.
2. Avoid re-invention of the wheel;
3. Utilize good practice we already have;
4. Make transition as less painful as possible.

The Disciplined Agile Delivery (DAD) process framework is a people-first, learning-oriented hybrid agile approach to IT solution delivery. It has a risk-value life cycle, is goal-driven, and is enterprise aware. DAD actually the only one Agile flavour that has process framework built into it. DAD process framework has several important characteristics. These characteristics are:

1. People-first;
2. Goal-driven;
3. Risk and value driven;
4. Enterprise aware;
5. Learning-oriented;
6. Scalable.

DAD addresses the project lifecycle from the point of initiating the project through construction to the point of releasing the solution into production. DAD explicitly observe that each iteration is NOT the same. Projects do evolve and the work emphasis changes as we move through the lifecycle. To make this clear, DAD carve the project into phases with light-weight milestones to ensure that we are focused on the right things at the right time. Such areas of focus include initial visioning, architectural modelling, risk management and deployment planning. This differs from mainstream agile methods, which typically focus only on the construction aspects of the lifecycle.

The disciplined agile delivery lifecycle expands upon the Scrum construction lifecycle in three important ways:

- It has explicit project phases, recognizing that agile delivery is really iterative in the small and serial in the large.
- It includes a full range of practices. This includes initial requirements and architecture envisioning at the beginning of the project and iteration to increase the chance of building the right product in the right manner, as well as system release practices
- It includes more robust practices. The lifecycle of DAD explicitly reworks the product backlog into the more accurate concept of a ranked work item list. Not only do agile delivery teams implement functional requirements, they must also fix defects (found through independent testing or by users of existing versions in production), provide feedback on work from other teams, take knowledge transfers, and so on.

Based on these advantages PRL have selected DAD as our development methodology and process have been build around it using project management tool ProjeQtOr (ttp://www.projeqtor.org/en/). We are using DAD-based process for over 1 year with a great success on delivery of our projects on time and budget.

3.5 Cibersam

The software development model employed by Cibersam during the **RePhrase** project is the spiral model. Given the nature of the development process based on Matlab, this model is the most adequate. The development process consists of the following steps:

1. The initial design and implementation is done in Matlab due to its fast development and prototyping.
2. Once the Matlab code is implemented as a DSL, we translate it by using the Armadillo library [14]. This library enables the line-to-line transformation of the Matlab code, maintaining the legibility and traceability of the source code.
3. Next, the code is tested and evaluated, comparing the obtained outputs with the results obtained using Matlab.
4. Finally, in case of success, new functionalities and features are implemented in Matlab.

This model allows us to develop source code that requires especial attention, given the importance of the medical use case. The spiral model permits us to carefully control the quality of the resulting data in each step of the lifecycle. We use the SNR metric (signal-noise ratio) to validate the resulting data in each stage. This metric enables a quick and accurate validation, reducing significantly the implementation time.

Chapter 4

The Support for Development Models in RePhrase

The goal of the **RePhrase** project is to produce tools and technologies that will ease the development of high-performance parallel software, addressing every stage of the typical software development process from requirements capturing to the deployment and maintenance. Table 4.1 gives an overview of how different technologies developed in the **RePhrase** project fit with the software development stages, as well as the deliverables in which they are described.
One of the main goals of **RePhrase** is to ensure interoperability between different tools in the overall toolchain (show in Figure 4.1). While we are not aiming at the overall integration of the system in a sense that output of one tool could readily be used as an input to the next tool in the tool chain, we aim at enabling the tools to work together, possibly with some modifications of the outputs/inputs. Interoperability has been described for dynamic adaptivity tools in D5.3, and the interoperability of the overall tool chain will be described in D5.6 (M36). This, in particular, makes the **RePhrase** technologies usable in different software development methodologies, as it is very easy to switch between different phases or to repeat a certain phase. In the next sections, we give more details about the **RePhrase** technologies used in each phase and about the ways in which these tools can be used together in software development methodologies.

4.1 Requirements Analysis

We have investigated the specific issues that appear in the process of requirement capturing for parallel applications and produced a set of guidelines that can help the users make the right decisions at design and implementation time, in term of, for example, parallel patterns used in the application. Requirement analysis is usually independent of the other phases and is most of the time fully manual process, not suitable for automatisation via tools. Furthermore, in most of the software development methodologies, requirements capturing and analysis is done only once,

Software Development Phase	**RePhrase Tools**
Requirements Capture	**RePhrase** Requirements Engineering (D5.1)
Design	**RePhrase** Parallel Patterns (D2.1, D2.4, D2.5), **RePhrase** Pattern Language (D2.2), Pattern discovery mechanisms (D2.3), **RePhrase** refactoring tool (D2.2, D2.6)
Implementation and Debugging	**RePhrase** Parallel Patterns (D2.1, D2.4, D2.5), **RePhrase** Refactoring Tool (D2.2, D2.6), Race condition detection (D3.1, D3.2), Extra-functional properties debugging (D3.1, D3.2)
Testing, Verification and Analysis	**IBM** ExpliSAT (D3.1, D3.2), **IBM** Focus (D3.1, D3.2), **PRL** QA-Verify (D3.1, D3.2, D3.3), Race condition detection (D3.1, D3.2), Extra-functinal properties debugging (D3.1, D3.2)
Deployment	Static Mapping Infrastructure (D4.1, D4.2), PaRLSched Dynamic Scheduling Library (D4.1, D4.2), **IBM** Dynamic Compiler (D4.1, D4.2), Performance Monitoring (D4.1, D4.2)
Maintenance and Evolution	**RePhrase** Refactoring Tool (D2.2, D2.6), Parallel Patterns (D2.1, D2.4, D2.5), Mapping and Scheduling (D4.1, D4.2), **PRL** QA-Verify (D3.1, D3.2, D3.3)

Table 4.1: **RePhrase** technologies and parallel software development phases.

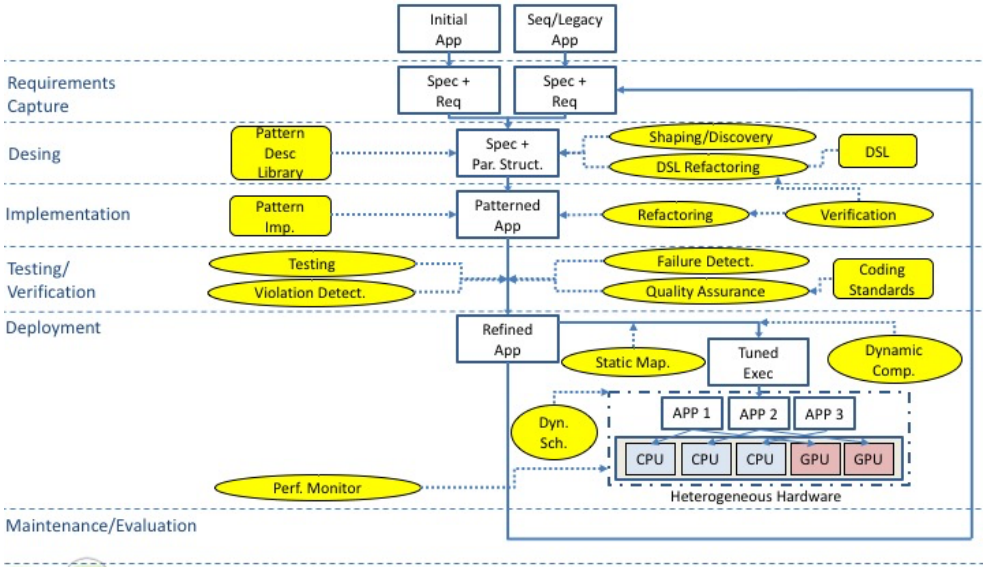


Figure 4.1: **RePhrase** Tool Chain.

with possible refinements later. Therefore, this stage does not require machnie-readable input from tools that deal with other phases, and its output is also not usually machine-readable.

4.2 Design

The main feature of the **RePhrase** methodology for the design phase of the software development is the *RePhrase Pattern Language* (RPL). This language enables the programmer to design the parallel application at very high level, in terms of composition of parallel patterns. RPL also allows specification of some implementation decisions related to the given pattern structure. The **RePhrase** Refactoring Tool supports automatic rewriting of the RPL expressions, enabling changes in the existing design of the parallel application, which might be requirement for *incremental*, *rapid* and *agile* software development models. Furthermore, there is a link between the changes in the application parallel structure, expressed in RPL, and the corresponding changes in the implementation - refactoring of the RPL is tightly linked with the refactoring of the resulting parallel C++ code, which enables two-way linking between design and implementation phases of the software development. Finally, pattern discovery mechanisms can be used to extract the basic structure of the application to give the initial parallel structure in RPL.

4.3 Implementation and Debugging

The **RePhrase** refactoring tool is the main point of interaction to the programmer for the implementation phase of the software development. It allows semi-automatic generation of the parallel code based on the RPL constructs from the design phase, as well as direct parallelisation of the existing sequential applications. The mechanisms for discovering the most common concurrency bugs, such as race conditions, as well as the mechanisms for detection violation of extra-functional requirements, are well integrated into the tool, enabling transparent interleaving of debugging and implementation. The output of the implementation phase is the parallel C++ code which can readily be used by the **RePhrase** testing and verification mechanisms.

4.4 Testing, Verification and Analysis

In addition to the race condition detection and extra-functional properties violation detection tools, also used in the implementation/debugging phase, for the testing, verification and analysis stage of software development we utilise the **IBM** *Focus* and *ExpliSAT* tools. Focus generates the test cases for complete coverage of the code and ExpliSAT verifies the equivalence between two versions of the code (in our case, between sequential and parallelised version). Both of these tools have

been extended in the project to deal with parallel code, as well as sequential. They receive as input sequential or parallel C++ code, as produced by the **RePhrase** Refactoring Tool. We are currently working on providing the feedback from these tools back to the refactoring tool, pointing to the potential problems in the code. We are also using **PRL** *QA-Verify* tool to verify that the resulting parallel code conforms to the required standards and suggest appropriate changes. The output of this phase of development is also C++ code, which can be readily used by the technologies for the deployment phase. As we have mentioned before, the same code can be used again by the tools for the implementation/debugging phase, if testing, verification and analysis identifies problems. This means we can freely get to and from the implementation/phase, a requirement for the use of *incremental*, *agile* and *rapid* development methods.

4.5 Deployment

The **RePhrase** tools for support with the deployment of parallel applications, which include static mapping, *PaRLSChed* dynamic scheduler, **IBM**'s dynamic compilation infrastructure and performance monitoring, work on generic C++ code and execute it on parallel environment. Therefore, the output of previous phases of development can readily be used at this stage. Furthermore, performance monitoring can provide feedback to the **RePhrase** Refactoring tool, allowing us to go back to the previous stages and make necessary modifications to the code, if problems are identified at this stage.

4.6 Maintenance

The **RePhrase** refactoring tool serves as the main point of contact of the programmer with the rest of the system, and most of the tools and technologies link directly to this tool. It is also well known that refactoring methodology produces the code that is easier to maintain that the code generated by hand. All this makes for easier maintenance of the parallel code by the **RePhrase** methodology, since most of the changes to the code can be performed semi-automatically. Dynamic adaptivity tools aimed for the deployment development phase ensure adaptivity of the code to the new architectures without significant changes in the source code, as well as adaptivity to possibly changing extra-functional requirements such as performance and energy. Therefore, in many cases (such as change in the architecture on which the application is executed), there will be no need for modification and maintenance of the application, as the application will be able to adapt to the new environment automatically.

5. Conclusion

In this deliverable, we surveyed most used software development methods and tools currently, and we presented how projects' industrial and academic partners use them. Eventually, we reported an initial assessment of the methodology developed within the project to support these development models and their sustainability.

Bibliography

[1] Scott Ambler. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[2] Scott W. Ambler and Mark Lines. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise (IBM Press)*. IBM Press, 1 edition, June 2012.

[3] David J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

[4] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mallor, Ken Shwaber, and Jeff Sutherland. The Agile Manifesto. Technical report, The Agile Alliance, 2001.

[5] Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, May 1988.

[6] Barry W. Boehm. Spiral Development: Experience, Principles, and Refinements. In Wilfried J. Hansen, editor, *Spiral Development Workshop*, Pittsburgh, 2000.

[7] R. N. Burns and A. R. Dennis. Selecting the appropriate application development methodology. *SIGMIS Database*, 17(1):19–23, September 1985.

[8] Lee Copeland. Extreme programming. *ComputerWorld*, December 2001.

[9] James A. Highsmith, III. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House Publishing Co., Inc., New York, NY, USA, 2000.

[10] James Martin. *Rapid Application Development*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1991.

[11] Stephen R. Palmer and John M. Felsing. *A Practical Guide to Feature-Driven Development*. Prentice-Hall, 2002.

[12] Keith Richards. *Agile Project Management: Running PRINCE2 Projects with DSDM Atern Running Prince2 Projects with DSDM Atern*. TSO (The Stationery Office), UK, 2007.

[13] Walker W. Royce. Managing the development of large software systems: concepts and techniques. *Proc. IEEE WESTCON, Los Angeles*, pages 1–9, August 1970. Reprinted in *Proceedings* of the Ninth International Conference on Software Engineering, March 1987, pp. 328–338.

[14] Conrad Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. 2010.

[15] Ken Schwaber. *SCRUM Development Process*, pages 117–134. Springer London, London, 1997.

[16] Hirotaka Takeuchi and Ikujiro Nonaka. New new product development game. *Harvard Business Review*, 1986.

[17] Jeffrey L. Whitten and Lonnie D. Bentley. *Systems Analysis and Design Methods*. McGraw-Hill, Inc., New York, NY, USA, 7 edition, 2007.