



LarKC

*The Large Knowledge Collider
a platform for large scale integrated reasoning and Web-search*

FP7 – 215535

D7b.1.1b

Iteration evaluation methodology and report template

**Angus Roberts
Hamish Cunningham
Adam Funk**

Document Identifier:	LarKC/2008/D7b.1.1b/V1.1
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	version 1.1
Date:	November 7, 2008
State:	final
Distribution:	internal



EXECUTIVE SUMMARY

This report describes the development methodology that will be used to construct software prototypes for the LarKC carcinogenesis use-case. The report describes an agile, iterative development strategy, in which users give requirements and feedback based on their experience of several prototypes. LarKC functionality will be delivered via a web front end: a wiki intended to integrate into the use-case partner's workflow. Evaluation and feedback will be provided via standard usability techniques, as described in the report. The report concludes by giving an initial set of evaluation reporting templates.

DOCUMENT INFORMATION

IST Project Number	FP7 – 215535	Acronym	LarKC
Full Title	Large Knowledge Collider		
Project URL	http://www.larkc.eu/		
Document URL	http://wiki.larkc.eu/LarkcProject/WP7b/MethodologyDeliverable		
EU Project Officer	Stefano Bertolo		

Deliverable	Number	7b.1.1b	Title	Iteration evaluation methodology and report template
Work Package	Number	7b	Title	Carcinogenesis reference production




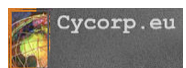








Date of Delivery	Contractual	M6	Actual	31-Sept-08
Status	version 1.1		final <input checked="" type="checkbox"/>	
Nature	prototype <input type="checkbox"/> report <input checked="" type="checkbox"/> dissemination <input type="checkbox"/>			
Dissemination Level	public <input type="checkbox"/> consortium <input checked="" type="checkbox"/>			

Authors (Partner)	Angus Roberts (University of Sheffield), Hamish Cunningham (University of Sheffield), Adam Funk (University of Sheffield)			
Resp. Author	Angus Roberts		E-mail	a.roberts@dcs.shef.ac.uk
	Partner	University of Sheffield	Phone	+44 (114) 222 1917

Abstract (for dissemination)	LarKC is developing the Large Knowledge Collider (LarKC), a platform for massive distributed incomplete reasoning that will remove the scalability barriers of currently existing reasoning systems for the Semantic Web. LarKC will be used in several use-case prototypes, including one to assist with carcinogen research. This report describes the methods that will be used to develop and evaluate the carcinogen research prototype. The report describes an iterative development strategy, in which users give requirements and feedback based on their experience of several prototypes. The report also describes a Wiki that will form the basis of these prototypes, and the standard evaluation methods that will be used to measure user's preferences.
Keywords	Evaluation, Usability, Software Development, Agile Methods, Wikis, CMS

Version Log			
Issue Date	Rev No.	Author	Change
11/09/2008	1	Angus Roberts	Internal QA draft
22/09/2008	2	Angus Roberts	Internal review completed
25/09/2008	3	Angus Roberts	Front matter corrected
05/11/2008	4	Angus Roberts	Correct deliverable number; add bibliography to contents; add conclusion

PROJECT CONSORTIUM INFORMATION

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck http://www.sti-innsbruck.at	STI 	Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
AstraZeneca AB http://www.astrazeneca.com/	ASTRAZENECA 	Bosse Andersson AstraZeneca Lund, Sweden E-mail: bo.h.andersson@astrazeneca.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL 	Emanuele Della Valle CEFRIEL SCRL. Milano, Italy E-mail: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O. http://cyceurope.com/	CYCORP 	Michael Witbrock CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia E-mail: witbrock@cyc.com
Hchstleistungsrechenzentrum, Universitaet Stuttgart http://www.hlrs.de/	HLRS 	Dr. Georgina Gallizo Hchstleistungsrechenzentrum, Universitaet Stuttgart Stuttgart, Germany E-mail : gallizo@hlrs.de
Max-Planck-Institut fr Bildungsforschung http://www.mpib-berlin.mpg.de/index_js.en.htm	MAXPLANCKGESELLSCHAFT 	Michael Schooler, Max-Planck-Institut fr Bildungsforschung Berlin, Germany E-mail: schooler@mpib-berlin.mpg.de
Ontotext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO 	Atanas Kiryakov, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: atanas.kiryakov@sirma.bg
SALT LUX INC. http://www.saltlux.com/EN/main.asp	Saltlux 	Kono Kim SALT LUX INC Seoul, Korea E-mail: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT http://www.siemens.de/	Siemens 	Dr. Volker Tresp SIEMENS AKTIENGESELLSCHAFT Muenchen, Germany E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD http://www.shef.ac.uk/	Sheffield 	Prof. Dr. Hamish Cunningham THE UNIVERSITY OF SHEFFIELD Sheffield, UK E-mail: h.cunningham@dcs.shef.ac.uk
VRIJE UNIVERSITEIT AMSTERDAM http://www.vu.nl/	Amsterdam 	Prof. Dr. Frank van Harmelen VRIJE UNIVERSITEIT AMSTERDAM Amsterdam, Netherlands E-mail: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY http://www.iwici.org/	WICI 	Prof. Dr. Ning Zhong THE INTERNATIONAL WIC INSTITUTE Mabeshi, Japan E-mail: zhong@maebashi-it.ac.jp


<p>INTERNATIONAL AGENCY FOR RESEARCH ON CANCER http://www.iarc.fr/</p>	<p>IARC2</p> 	<p>Dr. Paul Brennan INTERNATIONAL AGENCY FOR RESEARCH ON CANCER Lyon, France E-mail: brennan@iarc.fr</p>
--	--	---

TABLE OF CONTENTS

1	INTRODUCTION	2
1.1	The use case	2
1.2	Developing prototypes for the use case: a methodology	3
1.3	Delivering LarKC functionality	3
1.4	Evaluation to drive requirements gathering	4
1.5	Structure of this document	4
2	DEVELOPMENT METHODS	5
2.1	Agile Software Development	5
2.1.1	Agile Teams and Scrum	6
2.1.2	Supporting the Programmer with XP	7
2.2	Agile Research	8
2.2.1	Process Summary	9
3	SOFTWARE DELIVERY	12
3.1	Introduction	12
3.2	CoW: A Controllable Wiki	13
3.2.1	Philosophy	13
3.2.2	Overview of CoW	14
3.2.3	Why another wiki?	14
3.2.4	Main features	14
3.2.5	Using SVN as a Wiki backend	15
3.2.6	Controlled languages and semantics	15
4	EVALUATION	16
4.1	Methodology	16
4.1.1	Test procedure, tasks and questionnaires	17
4.2	Background	18
4.3	Statistical analysis	18
4.3.1	Statistical measures	18
4.3.2	Sample quality	18
4.3.3	Subjects' suggestions and comments	19
4.4	Questionnaires	20
5	CONCLUSIONS	26
5.1	Prototype development through an iterative methodology	26
5.2	Next steps	26
5.3	Acknowledgements	26
	REFERENCES	28

LIST OF ABBREVIATIONS

CMS	Content Management System
CoW	A Controllable Wiki
GATE	General Architecture for Text Engineering
GWAS	Genome Wide Association Study
IARC	International Agency for Research on Cancer
ICT	Information and Communication Technology
LarKC	The Large knowledge Collider project
SUS	System Usability Scale
SVN	Subversion version control system
XP	Extreme Programming

1 INTRODUCTION

This document describes the methodology that will be used to support prototype software development for LarKC end-users in a specific use-case, carcinogenesis research. This prototype will be used by scientists working for the LarKC partner International Agency for Research on Cancer (IARC). (Note that this document does not describe development of the LarKC platform architecture, nor of plugins for that architecture: these are described by deliverables of the technical workpackages.)

1.1 The use case

The WP7b cancer research usecase starts from two scenarios: assisting IARC scientists in the production of reference works, and assisting IARC epidemiologists in the analysis of gene-disease association study data. The use case is described in full in Deliverable D7b.1.1a. We summarise these requirements below.

Reference Work Production The IARC monographs¹ are a series of volumes evaluating potential carcinogens. They are definitive, and encyclopaedic. Each monograph is based on a review of all pertinent literature. IARC scientists identify all relevant papers for review, and relevant international experts to carry out the review and write the monograph. The literature is large and complex. We aim to support the search and use of this literature by providing richer querying capabilities, over semantically annotated literature, and over the Linked Life Data knowledge base (see Work Package 7a).

Genome wide association studies Genome Wide Association studies (GWAS) examine an entire genome to look for genes that are associated with a particular disease. A population of people with the disease are compared to a population without. Gene microarray chips are used to probe the entire genome of the study participants. If a probe has a statistically significant association with the disease population when compared to the non-disease population, then genes in the region of that probe may have some bearing on the disease. Given the huge number of probes, and the small population sizes, statistically significant probes are hard to find. We aim to support IARC scientists by introducing additional information into the statistical association models, to improve ranking techniques. Additional information will be provided by inference over semantically annotated literature and from existing knowledge bases.

Semantic annotation The cancer research usecases involve semantic annotation of the life science literature against existing biomedical resources. The scale of the literature is huge, in the order of tens of millions of texts. The ontologies against which the literature will be annotated consists of the largest RDF knowledge bases — several in the order of half a billion triples. These resources are heterogeneous, incomplete, and dynamic. Semantic annotation relative to such knowledge bases has not been previously attempted. Both the annotation task, and retrieval, selection and reasoning over the resulting data repository, will certainly be a test of LarKC's scalability.

¹<http://monographs.iarc.fr/>

1.2 Developing prototypes for the use case: a methodology

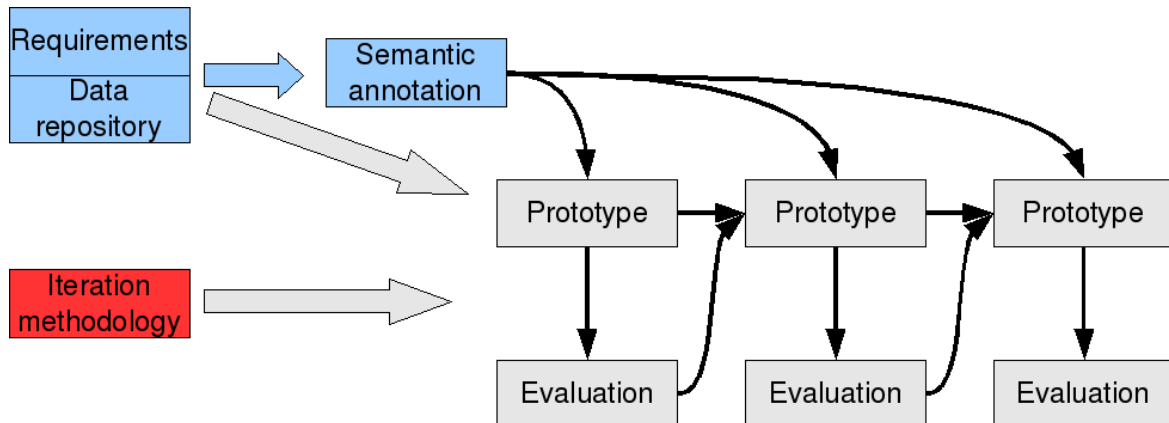


Figure 1.1: The deliverable in context

IARC end users are predominantly interested in retrieving and using the biomedical literature. Two uncertainties are apparent when we consider the software that they will use as LarKC end users:

1. It is unclear what new functionality and ability LarKC will provide
2. It is unclear how users' own requirements will develop and change when they are faced with new technologies. It is likely that they will come up with new and interesting ways of working.

We therefore plan to use an iterative development methodology, which is flexible and responsive to users' changing requirements. We will deliver software over several prototype iterations, each followed by a user-centred evaluation. This methodology deliverable therefore stands in relation to the initial requirements gathering exercise (as described in Deliverable D7b.1.1a), and to future prototype / evaluation iterations. This is shown in Figure 1.1.

1.3 Delivering LarKC functionality

The IARC end user of LarKC needs software to search, navigate, retrieve, and organise the biomedical literature, and with which to integrate the results of their search into evolving documents. Additionally, they need to do this in a collaborative environment. The functionality provided by LarKC needs to be wrapped within an extensible, collaborative, content management environment. We therefore propose to deliver LarKC end-user functionality through a collaborative wiki and content management system (CMS): CoW — a Controllable Wiki. The first iteration of CoW will contain minimal functionality, but as LarKC matures, and further user requirements are exposed, the required functionality will be incorporated into CoW. As a first step, we imagine CoW users being able to query literature databases, the result sets of these queries being dynamically incorporated into shared documents on which the user is working.

1.4 Evaluation to drive requirements gathering

Development of the functionality within CoW will be driven by requirements gathering exercises, and by user evaluations. Each evaluation will compare the functionality of a new system to a previous system. The results of the evaluation, and of user questionnaires, will drive the next stage of prototype development. We will base these user evaluations on the quantitative System Usability Scale (SUS) [4].

We will report three major prototype / evaluation iterations, as shown in Figure 1.2. These three major iterations will be provided as project deliverables. They do not preclude shorter internal cycles, as described in the background to our methodology.

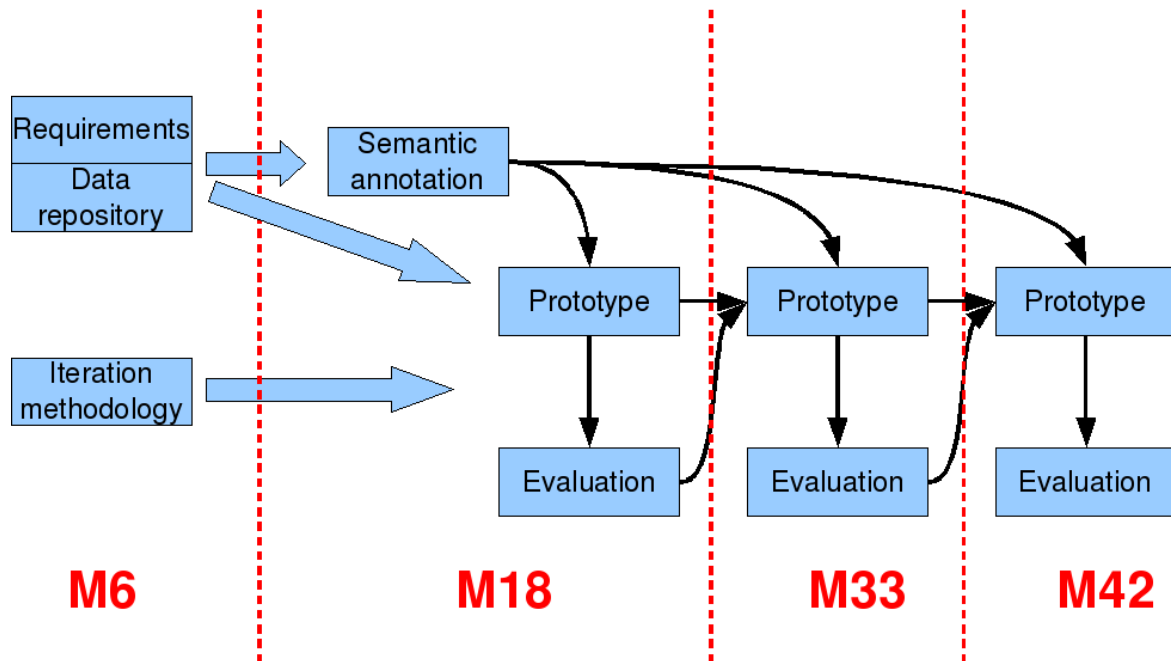


Figure 1.2: The prototype timeline, in the context of other deliverables

1.5 Structure of this document

This document is structured as follows. The next Chapter describes the theoretical background and justification to our development methodology, *Agile software development*. This is followed in Chapter 3 by a description of delivery via the CoW wiki and content management system. Finally, Chapter 4 looks at the evaluation strategy that will be used to drive requirements gathering at each iteration, and provides the reporting templates that will be used for evaluation.

2 DEVELOPMENT METHODS

2.1 Agile Software Development

To start with the obvious, building software is hard. How do people cope with hard construction problems? They hire an architect, who draws a lot of complicated pictures (or, more likely these days, builds a complex computer model), and, for big projects, builds a small scale prototype to give the commissioning customer the feel of the thing. This design process iterates until all appears to be well, and off we go to the cement pourers and the concrete reinforcers.

Software engineering, both as craft and later as discipline, began by making a similar analogy, and so adopted a commission-design-build lifecycle conception of the ideal software development process. The first mature set of methods used techniques like flow charts, entity-relationship modelling and structure diagrams to capture the architectural team's vision of the system to be build (e.g. [15]). Later incarnations merged the data structures with the algorithms and created *object-oriented* design (e.g. [3]). These modelling approaches were then used to produce as comprehensive a design as possible, which was then turned over to implementation teams lower down the food chain (the skilled work was concentrated in architectural hands while the programmer's work taken to be almost as routine as bricklaying). Because of the step-by-step procession from high to low these methods were often called *waterfall* development.

And everyone lived happily ever after. Well, not really. The problem with the architectural metaphor for software engineering is that software and the contexts within which it is constructed have turned out to be rather unlike houses and bridges, and more like evolving viruses or cantankerous animals. This is probably due of a number of factors, including:

- The act of building software changes the needs and the ideas of the people who are going to use it.
- The working systems that software is meant to support or replace are often so complex that no single person possesses a comprehensive understanding of them.
- Software increasingly sits at the bleeding edge of technology's interface with society, and the markets that condition which parts of this interface expand and contract are chaotic and fast-moving. In other words the context is both a strong determinant of success and changing very rapidly.
- Verification of the correct working of software is an extremely difficult problem to solve theoretically. Whereas we can be quite confident of the properties of a wall or joist built to certain well-established principles, the equivalent level of certainty with respect to software is much more difficult to achieve (and in the general case can only be approached via testing and not theoretical proofs¹).

Therefore it has become a truism that the best way to build a particular piece of software is to have built a very similar one before [8]. Obviously this is often

¹Automatic verification is a lively area of research but not yet widely applicable in practice.

impossible, and especially so in an R&D context. The waterfall methods have failed² most often in dynamic contexts; the piles of design artefacts can end up out-of-date before they are finished, and the synchronisation of design and code a never-ending process which makes the nature of the executable system ever more opaque.

Over the last decade or so a new family of software engineering methods have arisen based on the types of insight sketched above: the *agile methods*; below we discuss what we may usefully learn from this experience for our research work in projects like LarkC in general, and WP7b in particular, beginning with a short summary of two of the most popular members of the agile family, Scrum and XP (eXtreme Programming).

2.1.1 Agile Teams and Scrum

In their book *Agile Software Development with Scrum* Ken Schwaber and Mike Beedle [12] recall visiting a DuPont factory and talking with process engineers engaged in managing complex chemical processes. They described the types of activity they routinely encountered in software development and the types of (non-agile) design methods in use. The DuPont engineers laughed: they couldn't see how the methods matched up with the process because the methods were, from their point-of-view, appropriate only in much more defined and predictable circumstances. They suggested that for the chaotic and fast-moving software world a different approach based on constant readjustment and flexibility was more appropriate. Scrum is:

...a process skeleton that includes a set of practices and predefined roles. The main roles in Scrum are the ScrumMaster who maintains the processes and works similar to a project manager, the Product Owner who represents the stakeholders, and the Team which includes the developers.

During each sprint, a 15-30 day period (length decided by the team), the team creates an increment of potential shippable (usable) software. The set of features that go into each sprint come from the product backlog, which is a prioritized set of high level requirements of work to be done. Which backlog items go into the sprint is determined during the sprint planning meeting. During this meeting the Product Owner informs the team of the items in the product backlog that he wants completed. The team then determines how much of this they can commit to complete during the next sprint. During the sprint, no one is able to change the sprint backlog, which means that the requirements are frozen for a sprint.

There are several implementations of systems for managing the Scrum process which range from yellow stickers and white-boards to software pack-

²How to construct a failing software project: option 1: work for the UK government; option 2: define your plan as:

- first we (highly-paid developers) decide what to build (requirements)
- then we (middle-level developers) decide how to build it (design)
- then we (lower-level developers) implement it (coding)
- then we (support staff) deploy it (disaster)

A research equivalent: I hereby promise to invent The Next Big Thing at 3.27 PM on the 23rd of February 2010 in order to produce Deliverable D2,345 revision 63.

ages. One of Scrum's biggest advantages is that it is very easy to learn and requires little effort to start using.

<http://en.wikipedia.org/wiki/Scrum> Wikipedia 2/Sept/2008

Pictorially: see figure 2.1.

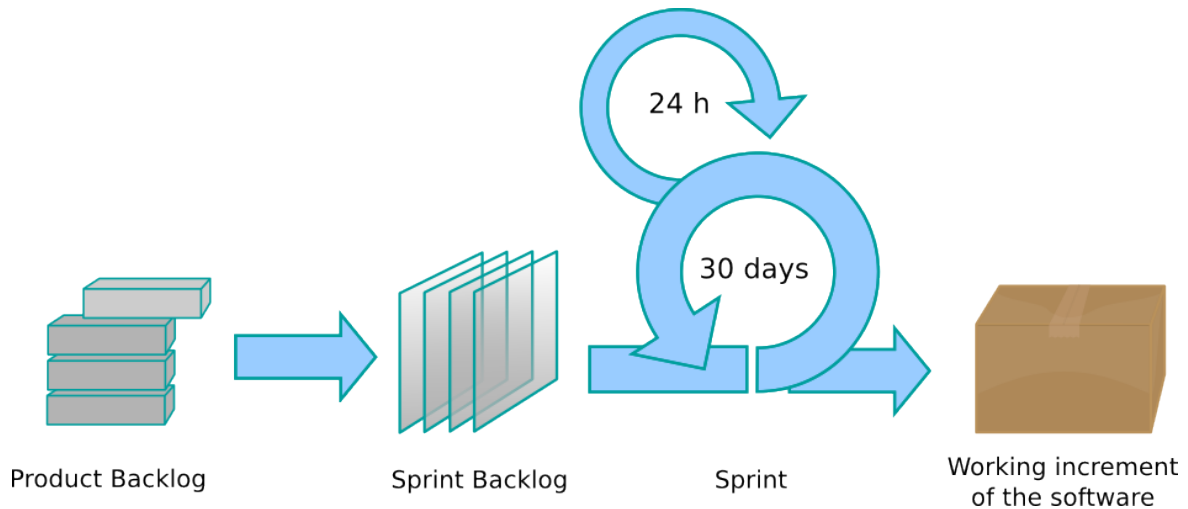


Figure 2.1: Scrum in pictures

2.1.2 Supporting the Programmer with XP

In 1999 Kent Beck wrote a book called *Extreme Programming Explained: Embrace Change* [2] wrote a book that advocated dispensing with most design artefacts and implementing a programmer-centric division of labour. The philosophy was to accept the dynamism and complexity of the software context and to make a virtue out of a weakness by focussing on flexibility and the ability to change course at any point. Key techniques were to keep code quality high (to *refactor* at every opportunity, building libraries of reusable code instead of continually reinventing incrementally different solutions), maximise communication between programmers and between programmers and clients, and to test, test, and test some more as the main method of verifying functionality.

The original list of practises was:

Small Releases: Put a simple system into production quickly, then release new versions on a very short cycle.

Metaphor: Guide all development with a simple shared story of how the whole system works.

Simple Design: The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.

Testing: Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests indicating that features are finished.

Refactoring: Programmers restructure the system without changing its behaviour to remove duplication, improve communication, simplify, or add flexibility.

Pair Programming: All production code is written with two programmers at one workstation.

Collective Ownership: Anyone can change code anywhere in the system at any time.

Continuous Integration: Integrate and build the system many times a day, every time a task is completed.

40-hour Week: Work no more than 40 hours a week as a rule. Never allow overtime for the second week in a row.

On-site Customer: Include a real, live customer on the team, available full-time to answer questions.

Coding Standards: Programmers write all code in accordance with rules emphasizing communication throughout the code.

[2, page 54]

Regarding not working too much overtime, we have refined the protocol as follows:

- on Monday we recover from the weekend
- on Tuesday we get ready to work
- on Wednesday we work
- on Thursday we recover from work
- on Friday we get ready for the weekend

This keeps most of the team happy, although when we first raised the idea one member raised their hand and asked “Does this mean we’re going to work *every* Wednesday?”

Next we look at the promise of the new methods in a research context.

2.2 Agile Research

What of research? Here’s how *not* to do it:

Project Proposal: We love each other. We can work so well together. We can hold workshops on Greek islands together. We will solve all the problems of AI that our predecessors were too stupid to manage.

Analysis and Design: Stop work entirely, for a period of reflection and recuperation following the stress of attending the kick-off meeting in Luxembourg.

Implementation: Each developer partner tries to convince the others that program X that they just happen to have lying around on a dusty disk-drive meets the project objectives exactly and should form the centrepiece of the demonstrator.

Integration and Testing: The lead partner gets desperate and decides to hard-code the results for a small set of examples into the demonstrator, and have a fail-safe crash facility for unknown input (“well, you know, it’s still a prototype...”).

Evaluation: Everyone says how nice it is, how it solves all sorts of terribly hard problems, and how if we had another grant we could go on to transform information processing the World over (or at least the European business travel industry).

[7]

Seriously, there are significant risks hidden under the contractual carpet in modern ICT research, especially in shared-cost collaborative projects with many partners, many agendas, many pieces of background software that theoretically will be integrated and work smoothly together to demonstrate some radical new proposition regarding the possibility of the next high-tech revolution. For obvious reasons of financial probity these projects begin with the authoring of a lengthy annex to the funding contract that specifies several acres of milestones, deliverables and other checkpoints. Conformance to the plans set out in this annex (the *description of work* in current parlance) is measured by periodic expert peer review in the normal (academic) manner.

This is all well and good, but from a software development perspective it does tend to encourage staying with the type of waterfall methods which have been shown to be less than perfect in dynamic environments. How to cope?

In LarKC WP7b we have specified our outputs as a series of three iterations, and we have adopted an agile process for the development of these prototype instances. The rest of the section summarises this process.

2.2.1 Process Summary

Reprising the above, XP is a way to keep focussed on user needs and to ensure stability through testing. Scrum is a way to organise regular deadlines and to encourage teamwork towards clearly defined objectives, while avoiding lock down to inflexible targets. Scrum is less about the code and more about how to split development time into manageable chunks, how to structure implementation iterations, and how to filter and prioritise ideas for new features and technical changes.

This section summarises the agile process in use for LarKC WP7 software development. The main elements of XP and Scrum that we have adopted are:

- Test-driven development.
- Deliver early, deliver often.
- Only document where necessary (minimise design artefacts because they go out of date or cost lots to maintain).
- Maintain our library-based approach and refactor code into the frameworks in and around GATE. (Resist the temptation to reassure bureaucrats and marketing people of the novelty of the work by continually inventing new names.)
- Maintain our continuous integration suite with (minimally) nightly builds.

- Development done by self-organising teams.
- Development done on a bi-weekly or monthly cycle; no new work can be taken on by a team after a cycle starts.
- The list of features, functions, technologies, enhancements and bug fixes that would ideally be performed are prioritised and listed in a “backlog” which represents a snapshot of the ever-changing requirements and plan.
- For each development cycle pick the highest priority items to do next.
- There are no bad ideas for new features, only low priority features.
- Meet every 2 working days to briefly report: what’s been done since last meeting; what is planned for next; what has been a barrier. These are called ”scrums” and should last around 15 minutes.

The process:

- Develop a prioritised list of functionality, technology changes, bug fixes etc.
- Select a month’s work (or a week or a fortnight, depending on external factors).
- Do a month’s work (a “sprint”).
- Review.
- Repeat.

During a sprint new work items cannot be assigned to the developers unless they are of an urgent, show-stopper nature. (New items that are identified are added to the backlog instead.) The team can reduce or increase the amount of work in the sprint if it needs to, but tries to avoid reductions if possible.

As new functionality becomes available it is rolled into the applications and user feedback sought.

Artefacts:

- Applications.
- Backlog. This lists all new and changed functionality that we’ve decided we need to develop to improve the applications. Items at the top of the list should be detailed enough and fine-grained enough to be implemented.
- Test suite. All the backlog items that the team produces are delivered as a set of tests.
- User guide. This has to contain enough information for users to understand what the team outputs and to exploit it effectively.
- Developer guide. This is written relatively informally as a set of notes that point into the test suite javadocs/java2html.
- Documentation process: the sequence is: vision; backlog to-do; backlog done, linked to developer or user documentation.

Meetings:

- First Monday of the month: sprint end review, sprint backlog elements selection. (Changes when a sprint finishes early. Sprints can't finish late: if the work isn't complete when they end we just stop, replan, and start again.) A sprint end review looks at each backlog item that was planned and asks "Do tests exist that cover this item, and do they succeed?" and "Is the customer using this, and if not then when?"
- Monday/Wednesday/Friday bi-weekly and Tuesday/Thursday bi-weekly alternating: team (scrum) meeting. The team meeting is very short; each member summarises what they did since the last meeting, what they plan to do next, and anything that is a problem (technological, organisational, whatever).
- Alternate Thursdays: management meeting (analyse problems raised in team meetings, check backlog priorities, review code).
- As required: user meetings. When we're the users or when the users are remote we have to simulate them. In XP style new development is rolled into the applications as quickly as possible.

3 SOFTWARE DELIVERY

3.1 Introduction

The previous Chapter looked at Agile methods of software development. We plan to apply Agile methods to develop prototype software for the LarKC end-user in the carcinogenesis use-case. This chapter describes a delivery interface for that prototype software. Each iterative development of the prototype will be evaluated as described in the following Chapter, to provide feedback and requirements for further development.

Much of the carcinogenesis use case involves supporting literature search (see deliverable D7b1.1a). The user will not interact directly with the LarKC platform, but through some interface. We envisage the user writing queries and interacting with the platform via a workbench, in which they query and navigate the literature, store their results, and write documents based on their searches. The IARC end user of LarKC will use this workbench to search, navigate, retrieve, and organise the biomedical literature, and to integrate the results of their search into evolving documents. Additionally, the workbench will allow them to do this in a collaborative environment. The functionality provided by LarKC needs to be wrapped within an extensible, collaborative, content management environment. The prototype of this workbench, that we will develop in the next phase of LarKC, is thus a part of our methodology: it is the shell into which user functionality will be iteratively developed, and upon which user studies and evaluations will be performed. This is shown in Figure 3.1.

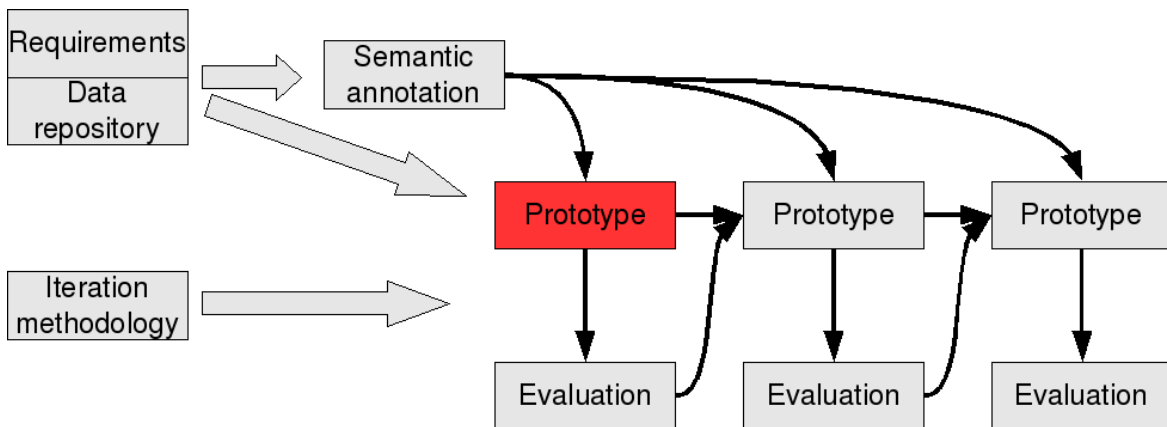


Figure 3.1: The first prototype in context

We propose to deliver LarKC end-user functionality through a collaborative wiki and content management system (CMS): CoW — a Controllable Wiki.

- Wikis are about allowing groups of people to create and edit sets of interlinked web pages with minimal effort and minimal learning of software conventions and features.
- Content Management Systems (CMSs) provide persistence, versioning, metadata management, upload and browsing of sets of documents and other resources.

Initial functionality will be low, perhaps providing some simple file system and organisational functionality to assist with reference sharing. As a first step, we imagine

CoW users being able to query literature databases, the result sets of the queries being dynamically incorporated into shared documents on which the user is working. Functionality will be extended as other requirements are implemented. In this way, users will be able to get used to the environment with little disruption to their daily work.

CoW is an interface through which LarKC software for use by IARC staff will be delivered. Based on their use of this interface, they will provide feedback for further development. As such, it can be seen as part of our use of the Agile methodology for gathering and implementing user requirements. We describe its initial design here.

3.2 CoW: A Controllable Wiki

3.2.1 Philosophy

Files and directories, documents and folders, disks and memory sticks, laptops and games machines, TV time-shifters and corporate IT systems. Data, data everywhere, and never a drop to drink, as the Ancient Mariner could not have dreamt of saying. Wouldn't it be nice to be able to view your filesystem as a website, to be able to edit from multiple machines with and without network connections, to be able to have your own local copy and also share it with friends and colleagues, and to not have to worry about merging it all back together?

CoW is a “Controllable Wiki” and CMS that supports collaborative document creation with asynchronous off-line editing. CoW is designed to make it easy to add interaction to static websites, and to support concurrent editing and off-line working with straightforward synchronisation (using Subversion). The system also serves as a test-bed for experiments in controlled language querying, for knowledge integration, and for round-trip ontology engineering.

Currently CoW includes:

- a wiki language with a JavaCC parser
- a CMS persistence and versioning backend using Subversion ¹;
- simple display and edit of wiki pages and other static content using Grails ²;
- a version of the Grails JSecurity plugin that includes editing of the user data and assigns permissions to wiki areas based on their database ID;
- user registration using the Captcha plugin;
- webtests using Canoo Webtest ³;
- a minimal Hypersonic database to store pointers to wiki areas and user/role/permission data).

The system has two modes, workstation and server; the former does no user management, the latter uses JSecurity.

¹<http://subversion.tigris.org/>

²<http://www.grails.org/>

³<http://webtest.canoo.com/>

3.2.2 Overview of CoW

Wikis are about allowing groups of people to create and edit sets of interlinked web pages with minimal effort and minimal learning of software conventions and features. Typically they achieve this by

- having an edit link from each page that runs a browser-based editor
- allowing pages to be written in simple plain text with a small number of formatting conventions (e.g. **bold**)
- supporting creation of pages by directing links for non-existent pages to a create and edit page
- allowing multiple users to use the system simultaneously (and sometimes to edit the same document simultaneously)

Content Management Systems (CMSs) provide persistence, versioning, metadata management, upload and browsing of sets of documents and other resources.

3.2.3 Why another wiki?

Scratching three itches:

1. adding interaction to a largish static site (15k HTML files, 40k other files)
2. wiki style collaborative document creation with asynchronous off-line editing
3. a test-bed for experiments in querying and controlled languages for retrieval of semantic annotation, and mixed queries across text and knowledge bases.

3.2.4 Main features

CoW is different from other wikis and CMSs because:

- it is designed from the ground up to support concurrent editing and off-line working with straightforward synchronisation using SVN ⁴
- it uses the YAM language, which
 - outputs LaTeX as well as HTML
 - allows paths as links (i.e. does not limit the namespace to a single directory like e.g. JSPWiki does) and consequently allows a tree-structured page store (and later graph-structured navigation via an ontology)
- it allows mixing of all types of files in its page store (which is just an SVN sandbox, in fact)
- it supports versioning and differencing via SVN, and allows other tools that manipulate SVN repositories to be used with the wiki data (e.g. Tortoise, Eclipse, ViewVC, etc.)

⁴SVN: the Subversion version control system.

- it supports embedded query languages, and experiments with applications that store their data in semantic repositories whose schema is user-defined and maintained

3.2.5 Using SVN as a Wiki backend

No available Wiki in Java that we could find has good SVN support. Using SVN as a backend gives us:

- off-line edit — simply checkout the pages and edit to your heart's content while off-line
- edit with other tools, not just the web forms interface
- management of authorship-related metadata such as which lines were added, difference between versions and so on
- a stable and reliable versioning system that's been proved in production use by 000,000s of developers
- concurrent editing

Why not JCR layered on top of SVN? Just a question of team resources and experience, the strength of the SVNKit library and so on.

3.2.6 Controlled languages and semantics

Annotating documents with semantics is not a panacea for all the problems of document retrieval, but can in certain circumstances be beneficial, especially for high value and medium or low volume content.

CoW is partly intended to be an experimental framework for a new type of website in which

- the database is complemented by a knowledgebase storing semantic annotation (using OWLIM ⁵);
- the knowledge base is defined and populated using user queries and controlled languages;
- result sets of user queries are integrated into their wiki pages, allowing them to write new documents around their results;
- concurrent changes are managed as in CVS or SVN (check out, edit, update, merge, etc.).

⁵<http://www.ontotext.com/owlim>

4 EVALUATION

The previous Chapter introduced the CoW wiki, which will be used as a “shell” through which LarKC functionality will be exposed to users. At each iteration, we plan to examine user experience of the prototype, and to adjust the requirements on the prototype, through:

1. a requirements restatement
2. a user evaluation

The evaluation methodology is described in this Chapter. The requirements restatement will take the form of interviews and user group meetings with users at IARC. The user evaluation will provide more quantitative data to back this up. The user evaluation will ask users to carry out a few simple tasks using two of:

1. an existing tool that has similar functionality to the prototype
2. the current version of the prototype
3. a new version of the prototype

The user evaluation is a *user* evaluation against an *evolving prototype*. This means that:

- As the prototype and its functionality evolves through user feedback and pressure, then the evaluation tools themselves will need to evolve. Specific evaluation questions may no longer be relevant, and others may need to be added.
- The evaluation tools described here have been “seeded” with a set of general questions that are likely to apply to the first prototype. These are the questionnaires given in this chapter.
- The evaluation is about *useability* of the prototype, and is used to drive the development of software for the end user. The evaluation is not about measuring the performance of the underlying LarKC platform — this is dealt with by LarKC technology workpackages.

The evaluation methodology is based on a methodology successfully applied within the SEKT project. This is reflected by parts of this Chapter, and the evaluation forms described in later chapters, being based on material in SEKT deliverable D2.2.2 [9].

4.1 Methodology

We have prepared the following documents (given in full in the Chapter 4.4) for the users.

- The pre-test questionnaire (Figure 4.1) asks for background information: how much each subject already knows about literature search. We will score this questionnaire by assigning each answer a value from 0 to 2 (from left to right) and dividing the total by 12 to obtain a score of 0–100.

- The post-test questionnaire for each tool (Figure 4.2) is the *System Usability Scale*, a *de facto* standard for evaluating software usability, which also produces a score of 0–100. [4]
- The comparative questionnaire (Figure 4.4) to measure each user’s preference for one of the two tools. This form will be scored similarly to SUS so that 0 would indicate a total preference for Tool 2, 100 would indicate a total preference for Tool 1, and 50 would result from marking all the questions *neutral*. On the reverse side (Figure 4.5) and in discussion with the facilitators, we will offer each user the opportunity to provide comments and suggestions.

In addition, the user will be provided with:

- Two task definitions, Task A and Task B, asking users to perform a number of relatively straightforward tasks involving search and the use of search results.
- Short manuals for each tool, focussed on the skills needed for the tasks above.

We will recruit volunteers with varying experience levels and asked each subject to complete the pre-test questionnaire, to read the manual, and to carry out each of the tasks with one of the two tools. Half the users will carry out Task A with Tool 1 and then Task B with Tool 2; the other half will carry out Task A with Tool 2 and then Task B with Tool 1.

We will measure each user’s time for each task and in some cases for component subtasks. After each task we will ask the user to complete the SUS questionnaire for the specific tool used, and finally we will ask them to complete the comparative questionnaire.

Section 4.1.1 details the procedure that we will use for the evaluation. Section 4.2 explains the theoretical justification for our methodology; the remainder of this chapter presents the statistical tools that will be used to analyse the user questionnaires, followed by the initial questionnaires themselves.

4.1.1 Test procedure, tasks and questionnaires

1. First we will ask each subject to complete the pre-test questionnaire shown in Figure 4.1.
2. We will then give him either Tool 1 or Tool 2 and ask him to carry out Task A, while we record the time taken to complete sub-tasks.
3. We will then ask the subject to complete the questionnaire shown in Figure 4.2. We will then give the subject the other tool (Tool 2 or Tool 1 respectively), and ask him to carry out the Task B with this second tool.
4. We will then ask the subject to complete the questionnaire shown in Figure 4.2 and then separately the one in Figures 4.4 and 4.5.

4.2 Background

Our methodology constitutes a *repeated-measures, task-based* evaluation: each subject carries out a similar list of tasks on both tools being compared.

We chose the SUS questionnaire as our principal measure of software usability because it is a *de facto* standard in this field. Although it superficially seems subjective and its creator called it “quick and dirty”, it was developed according to the proper techniques for a Likert scale. [4]

Furthermore, researchers at Fidelity Investments carried out a comparative study of SUS, three other published usability questionnaires and an internal questionnaire used at Fidelity, over a population of 123 subjects, to determine the sample sizes required to obtain consistent, accurate results. They found that SUS produced the most reliable results across all sample sizes; they noted a jump in accuracy to 75% at a sample size of 8, but recommended a sample of at least 12–14 subjects. [14]

As a reference for interpreting the results, average SUS scores are usually between 65 and 70. [1] We will consider this to be the baseline for comparison in the next section.

4.3 Statistical analysis

4.3.1 Statistical measures

We will use the following statistical measures.

Descriptive statistics (min, max, mean, median) will be used to look at the general distribution of SUS scores for the two tools.

A *95% confidence interval* calculated from a data sample is a range which is 95% likely to contain the mean score of the whole population which the sample represents. [11]. We will break the scores down according to the tool used and the task (A or B) carried out, and calculate confidence intervals.

A *correlation coefficient* over a set of pairs of numeric data is analogous to the appearance of a scatter graph or X-Y plot. +1 signifies a perfect correlation and corresponds to a graph in which all points lie on a straight line with a positive slope; -1 signifies a perfect inverse correlation (the points lie on a straight line with a negative slope); 0 indicate a complete lack of correlation (random distribution of the points). Values $> +0.7$ and < -0.7 are generally considered to indicate strong correlations.

The formula for Pearson’s coefficients assumes that the two variables are linearly meaningful; physical measurements such as length and temperature are good examples of such variables. The formula for Spearman’s coefficients, on the other hand, stipulates only ordinal significance (ranking) and is often considered more appropriate for subjective measurements (such as many in the social sciences). [6, 10, 11, 5, 13]

We will use correlation coefficients to compare various pairings of SUS scores and task times. We will confirm the coherence of the questionnaire by looking at any correlation of the user’s preferences to the SUS scores.

4.3.2 Sample quality

We will examine sample quality, by establishing the consistency of two partitions of our sample:

by tool order or task-tool assignment: subjects who carried out task A on Tool 1 and then B on Tool 2, in comparison with those who carried out A on Tool 1 then B on Tool 2; and

by sample source: subjects drawn from different groups of users (to allow for biased subjects, in combination with measures of their expertise).

4.3.3 Subjects' suggestions and comments

The test will be completed by a short interview in which comments, problems and suggestions will be discussed. Together with an analysis of test results, this will be used to drive the next development iteration.

4.4 Questionnaires

- | | | | | |
|---|--|--------------------------------|------------------------------------|--------------------------------|
| 1 | I search the biomedical literature as part of my job. | Never <input type="checkbox"/> | Sometimes <input type="checkbox"/> | Often <input type="checkbox"/> |
| 2 | I have worked with PubMed. | Never <input type="checkbox"/> | Sometimes <input type="checkbox"/> | Often <input type="checkbox"/> |
| 3 | I have worked with Web of Knowledge / Web Of Science. | Never <input type="checkbox"/> | Sometimes <input type="checkbox"/> | Often <input type="checkbox"/> |
| 4 | I have worked with other literature databases. | Never <input type="checkbox"/> | Sometimes <input type="checkbox"/> | Often <input type="checkbox"/> |
| 5 | I am familiar with form-based searching tools (e.g. the standard PubMed interface) | No <input type="checkbox"/> | A little <input type="checkbox"/> | Yes <input type="checkbox"/> |
| 6 | I am familiar with textual query syntax (e.g. in PubMed: “protein p53 AND pubmed pmc local[sb]”) | No <input type="checkbox"/> | A little <input type="checkbox"/> | Yes <input type="checkbox"/> |
| 7 | I understand the term “wiki”. | No <input type="checkbox"/> | A little <input type="checkbox"/> | Yes <input type="checkbox"/> |
| 8 | I understand the term “version control”. | No <input type="checkbox"/> | A little <input type="checkbox"/> | Yes <input type="checkbox"/> |

Figure 4.1: Pre-test questionnaire

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1 I think that I would like to use this system frequently.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2 I found the system unnecessarily complex	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3 I thought the system was easy to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4 I think that I would need the support of a technical person to be able to use this system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5 I found the various functions in this system were well integrated	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6 I thought there was too much inconsistency in this system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7 I would imagine that most people would learn to use this system very quickly	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8 I found the system very cumbersome to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9 I felt very confident using the system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10 I needed to learn a lot of things before I could get going with this system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 4.2: Post-test questionnaire for each system

	Tool 1	Tool 1	Tool 2	Tool 2
1 I found one system's documentation easier to understand.	<input type="checkbox"/> much easier	<input type="checkbox"/> neutral	<input type="checkbox"/> easier	<input type="checkbox"/> much easier
2 I particularly disliked using one system.	<input type="checkbox"/> disliked strongly	<input type="checkbox"/> neutral	<input type="checkbox"/> disliked	<input type="checkbox"/> disliked strongly
3 I found one system easier to use.	<input type="checkbox"/> much easier	<input type="checkbox"/> neutral	<input type="checkbox"/> easier	<input type="checkbox"/> much easier
4 One system was harder to learn.	<input type="checkbox"/> much harder	<input type="checkbox"/> neutral	<input type="checkbox"/> harder	<input type="checkbox"/> much harder
5 I would prefer to use one system again.	<input type="checkbox"/> strongly prefer	<input type="checkbox"/> neutral	<input type="checkbox"/> prefer	<input type="checkbox"/> strongly prefer
6 I found one system more complicated.	<input type="checkbox"/> much more complex	<input type="checkbox"/> neutral	<input type="checkbox"/> more complex	<input type="checkbox"/> much more complex

Figure 4.3: Post-test questionnaire comparing the tools

	Tool 1	Tool 1	Tool 2	Tool 2
7 I found it easier to find the literature I was looking for in one system.	<input type="checkbox"/> much easier	<input type="checkbox"/> easier	<input type="checkbox"/> neutral	<input type="checkbox"/> Tool 2 much easier
8 I found that one system supported my work beyond the search task itself.	<input type="checkbox"/> much more support	<input type="checkbox"/> more support	<input type="checkbox"/> neutral	<input type="checkbox"/> Tool 2 much more support
9 I found it easier to write queries in one system.	<input type="checkbox"/> much easier	<input type="checkbox"/> easier	<input type="checkbox"/> neutral	<input type="checkbox"/> Tool 2 much easier
10 I found it easier navigate results in one system.	<input type="checkbox"/> much easier	<input type="checkbox"/> easier	<input type="checkbox"/> neutral	<input type="checkbox"/> Tool 2 much easier
11 I found it difficult to perform searches in one system.	<input type="checkbox"/> much harder	<input type="checkbox"/> harder	<input type="checkbox"/> neutral	<input type="checkbox"/> Tool 2 much harder
12 I found it easier to browse related searches and material in one system.	<input type="checkbox"/> much easier	<input type="checkbox"/> easier	<input type="checkbox"/> neutral	<input type="checkbox"/> Tool 2 much easier
13 I found it easier to save results in one system.	<input type="checkbox"/> much easier	<input type="checkbox"/> easier	<input type="checkbox"/> neutral	<input type="checkbox"/> Tool 2 much easier
14 I found it harder to integrate results with my work in one system.	<input type="checkbox"/> much harder	<input type="checkbox"/> harder	<input type="checkbox"/> neutral	<input type="checkbox"/> Tool 2 much harder

Figure 4.4: Post-test questionnaire comparing the tools

Do you have any comments on either or both systems?
Do you have any specific problems to report?
Do you have any suggestions for improving either system?

Figure 4.5: Post-test questionnaire comparing the tools

5 CONCLUSIONS

5.1 Prototype development through an iterative methodology

LarKC plans to build a platform that will enable reasoning scaled to the large semantic repositories that are now commonplace, such as biomedical knowledge bases and annotated literature. This will be achieved by borrowing techniques from traditional information retrieval and cognitive science to cut down the problem space. Both the approach and the scale will be novel. In demonstrating use cases on this platform, it is likely that we will come across new and unforeseen problems. When end users are faced with software based on the LarKC platform, new ideas and opportunities for its use are likely to emerge. These problems and opportunities will force us, as developers, to respond in flexible and *agile* ways. We hope to foster this by adopting the techniques described in this report:

- an agile methodology for design and development;
- a user-centered approach to software evaluation.

5.2 Next steps

The methodology described in this report will be used to build prototype software that demonstrates the LarKC platform in use, in a life science context. In order to use this methodology to create prototype software, we plan four further steps. These are illustrated in Figure 5.1, as mid gray boxes.

1. The requirements for the use case scenarios will be collected (see in LarKC Deliverable D7b1.1a)
2. These requirements will be used to build and configure annotation software, and to semantically annotate life science corpora (again, see LarKC Deliverable D7b1.1a).
3. The methodology described in this document will be used to form initial prototype software, that:
 - (a) is based on the requirements;
 - (b) uses the semantic annotation.
4. The methodology described in this document will be used to evaluate this prototype.

The results of this evaluation will then be used to inform further development and evaluation iterations.

5.3 Acknowledgements

We would like to thank members of the University of Sheffield GATE team for trialling various agile approaches to software development in recent projects.

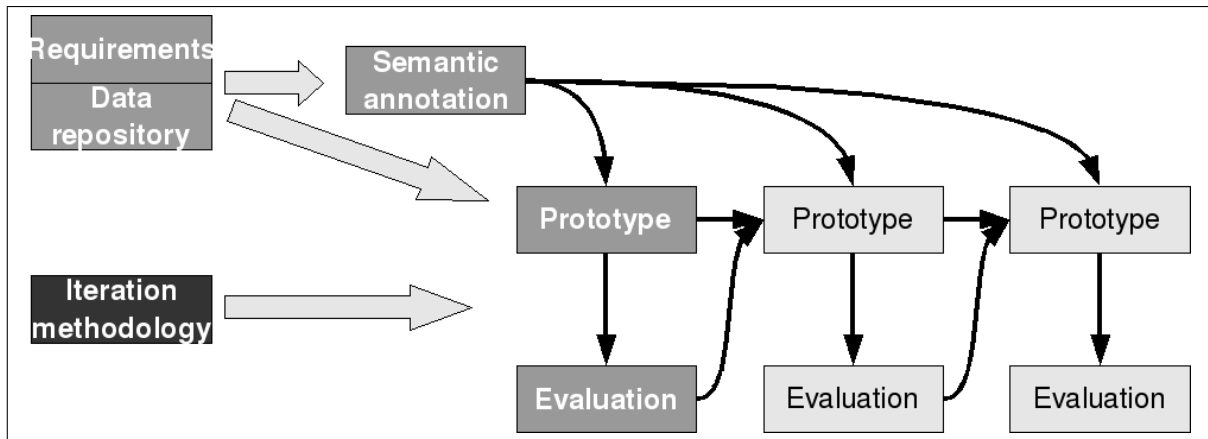


Figure 5.1: Requirements: next steps

REFERENCES

- [1] Bob Bailey. Getting the complete picture with usability testing. Usability updates newsletter, U.S. Department of Health and Human Services, March 2006.
- [2] K. Beck. *eXtreme Programming eXplained*. Addison-Wesley, Upper Saddle River, NJ, USA, 2000.
- [3] G. Booch. *Object-Oriented Analysis and Design 2nd Edn.* Benjamin/Cummings, 1994.
- [4] J. Brooke. SUS: a “quick and dirty” usability scale. In P.W. Jordan, B. Thomas, B.A. Weerdmeester, and A.L. McClelland, editors, *Usability Evaluation in Industry*. Taylor and Francis, London, UK, 1996.
- [5] Judith Calder. Statistical techniques. In Roger Sapsford and Victor Jupp, editors, *Data Collection and Analysis*, chapter 9. Open University, 1996.
- [6] T. G. Connolly and W. Sluckin. *An Introduction to Statistics for the Social Sciences*. Macmillan, third edition, 1971.
- [7] H. Cunningham and K. Bontcheva. Computational Language Systems, Architectures. *Encyclopedia of Language and Linguistics, 2nd Edition*, pages 733–752, 2005.
- [8] Jr. Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1978.
- [9] Adam Funk, Brian Davis, Valentin Tablan, Kalina Bontcheva, and Hamish Cunningham. Controlled language IE components version 2. Deliverable D2.2.2, SEKT, 2006.
- [10] David K. Hildebrand, James D. Laing, and Howard Rosenthal. *Analysis of Ordinal Data*. Quantitative Applications in the Social Sciences. Sage, 1977.
- [11] Jr. John L. Phillips. *How to Think about Statistics*. W. H. Freeman and Company, New York, 1996.
- [12] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
- [13] Steve Simon. Stats: Steve’s attempt to teach statistics. Technical report, Children’s Mercy Hospitals & Clinics, Kansas City, Missouri, November 2006.
- [14] Thomas S. Tullis and Jacqueline N. Stetson. A comparison of questionnaires for assessing website usability. In *Usability Professionals’ Association Conference*, Minneapolis, Minnesota, June 2004.
- [15] E. Yourdon. *Modern Structured Analysis*. Prentice Hall, New York, 1989.