Managing embedded software project team knowledge

P. Kettunen

Abstract: In modern large new product development (NPD) organisations the embedded software teams do not work in isolation but develop software in complex interrelationships with the other teams of the product development. Specifically the software team must interface with the system and hardware developments. Ambitious industrial product development programs require typically concurrent engineering of the various components of the system product, making it even more complicated for the software development to capture all the relevant information on time. Furthermore, multiple software projects may be working concurrently on different release versions of the same product family and platform core assets. In such a cross-functional environment, building and managing the embedded software team body of knowledge is not trivial. The paper proposes methods for managing such knowledge with a systematic modelling approach. The methods help in capturing project-specific context-driven key information. The empirical background of the study is in certain business units of Nokia Networks developing embedded software for telecommunications equipment products.

1 Introduction

Modern market-driven new product development (NPD) projects are often characterised by ambitious time-to-market goals while working in turbulent environments. Typically the whole product concept may still be under evolution even though the product engineering is already well on its way.

The embedded software projects responsible for the software components of such products face many new challenges compared to traditional placid contract-driven environments [1]. In particular, capturing all the necessary project-specific knowledge and product information introduces new problems in such cross-functional multiparty product development (PD) environments. The software team must often manage incomplete information over time. Frequent changes are inherent rather than exceptions [2].

What are the key sources of information for the embedded software project team in such environments? What kind of information is needed from those different sources? When is each piece of information needed relative to the product life-cycle? This paper addresses those questions by proposing systematic methods for capturing and managing embedded software project team knowledge. The goal is to provide a framework for the project manager to manage the project-specific key information. The empirical background of the study is in certain business units of Nokia Networks developing embedded software for telecommunications equipment.

IEE Proceedings online no. 20031173

doi: 10.1049/ip-sen:20031173

Paper first received 7th May and in revised form 8th October 2003

The author is with Nokia Group, P.O. Box 301, 00045 Nokia Group, Finland

2 Background and related work

2.1 Characteristics of embedded software product development

Industrial development of new products faces many challenges: often intense competition, rapid technological advances, and changing customer needs and expectations [3]. The product development must then be responsive and able to release new products in a timely fashion, yet with a good-enough quality for customer satisfaction. Because of the time pressure, product development decisions must often be made quickly with only partial information available. Uncertainty is inherent.

The speed of new product development is an increasingly important success factor [4]. A basic way of accelerating product development is to compress the workflows by performing sequential activities to some extent in parallel (overlapping), or even skipping some intermediate steps. The basic difficulty with such concurrency is that subsequent activities must often be based on partially incomplete information. Managing such concurrent engineering is more difficult than strictly sequential development. In particular, managing the flows of information is one of the key issues.

Software product development shares many of the characteristics of NPD in general. However, traditionally the NPD research has emphasised organisational and human issues while software engineering has focused on development processes [5]. Interteam knowledge transfer is an important enabler of accelerated product development, and interproject learning facilitates larger-scale productivity improvements within the PD organisation. In turbulent environments flexibility of the PD process, i.e. the ability to accommodate frequent and even very late changes, is a key success factor [6–8].

The development of large complicated systems products includes typically many concurrent activities for different technology areas. Systems engineering interconnects these different areas, of which the embedded software is one component. Concurrent engineering (CE) is a modern way of developing such combined hardware–software system products. For instance, the recent CMMI process model recognises such an integrated product development approach [9].

In such a complex environment the overall product development work is typically managed as a program comprising various projects. In particular, the embedded software development is usually one project of the product program. Large complex system products (e.g. telecommunications equipment) are typically developed during a long period of time (several years) as a series of releases. We must then take into account the life-cycle of the product. Many projects may be developing the software releases over time, possibly even concurrently. An embedded software release is then a software (sub)system package for the product supporting defined system features with a certain corresponding hardware release. Here we are focusing on market-driven rather than contract software development. Each project may be responsible for one particular release of the software product providing new features. Multiproject and project portfolio management are then issues [3]. Furthermore, often a class of similar products is developed as a product family utilising reusable software core assets and platforms. In such cases there are actually two types of software projects: application development and platform development [10]. This paper focuses on the former.

2.2 *Software project teams and knowledge*

Knowledge management (KM) is the overall concept of collecting data, formulating information and using that knowledge, i.e. knowledge is more than just information. In general there are two types of knowledge: explicit knowledge and tacit knowledge [11]. The knowledge assets can be tangible (e.g. documents) or intangible (e.g. experience).

In the software engineering context KM means in particular learning, capturing and reusing experience [12]. Typical software development knowledge management problems in practice are misunderstandings and imperfect communication caused by out-of-date documents, incomplete terminology definitions, undocumented information and unclear authoritative sources [13].

Product data management (PDM) is an integrated approach to combine all the information of complex products consisting of various parts and components into a consistently managed and accessible system [14]. Embedded software is one element of such product data.

Over the years there have been some studies about how large software organisations work and what factors affect software project team performance. A seminal investigation was conducted by Curtis *et al.* in the late 1980s [15]. Lack of and insufficient spread of domain knowledge and requirements related gaps were found to be major problem factors. A further key observation was that software project teams do not work in isolation, but the larger organisational context (individual, team, project, company and business milieu) must also be understood. A project interacts with the company, market and macro environment [3]. In large dispersed organisations multisite considerations may cause additional complexities [16].

In general, team learning has been recognised to be an important success factor for NPD [17]. Not only is technical knowledge needed but the project team must know their shared vision and common objectives. Various forms of informal and formal knowledge networks are useful within PD organisations [18]. It is important to understand that tacit knowledge sharing takes place in interactions between people [11]. Necessary enabling conditions must be satisfied to make this happen dynamically. For example, the emerging agile software development methods encourage such intensive communication with less formal documentation. Team proximity is then an issue.

In industrial PD project environments there is often a tradeoff between documentation completeness and the effort required to develop and maintain the documents. The key is to find a practical balance so that the risk caused by the partial or incomplete information is justified by the resource expenditure. Risk-driven specification acknowledges this [19]. Change management must be taken into account [20]. Agile software development methods try to avoid possibly obsolete intermediate documentation.

The key characteristic of embedded software development knowledge is often complex interconnections to the special-purpose target hardware. The hardware environments may be even highly exotic, in particular in so-called hybrid systems, and there are often stringent design constraints (e.g. performance). The software development team must then understand not only the general operation of the target hardware but also the overall functionality of the combined hardware–software system. Hardware/software codevelopment (cospecification, codesign) attempts to build this kind of shared system-level knowledge. There have been some approaches to create CASE tool environments, for instance, for such developments [21].

In our previous studies we have found requirements and specifications related problems to be among the main causes of trouble for telecommunications equipment development [22, 23]. Incomplete software requirements and specifications of the system are troublesome for the embedded software projects. It is important for the software development team to have sufficient knowledge of the system and hardware behaviour in order to be able to create the software part of the system product successfully.

3 Managing embedded system software project knowledge

3.1 *Contextual framework*

Considering products in general, there are three different levels of embedded software development knowledge to be managed: basic embedded software engineering knowledge and the base software (e.g. operating system) information, the special-purpose target hardware context, and the product system context including the related external systems.

With respect to product development projects there are different domains of knowledge as depicted by Fig. 1, showing an intersection set view. Basically both product and process knowledge are involved within the project scope.

As a team the software project must master all this knowledge in order to be able to develop the software successfully. But not every member of the team has to know everything.

We can start characterising the knowledge base of an embedded software project team by modelling the software development process workflow. Figure 2 is such a very high-level flow diagram.

However, this generic top-level diagram must be extended and refined in order to be useful for industrial product development embedded software project purposes. To begin with we must take the system and hardware engineering interdependencies into account. In addition the model must capture the key information sources of the product development environment.

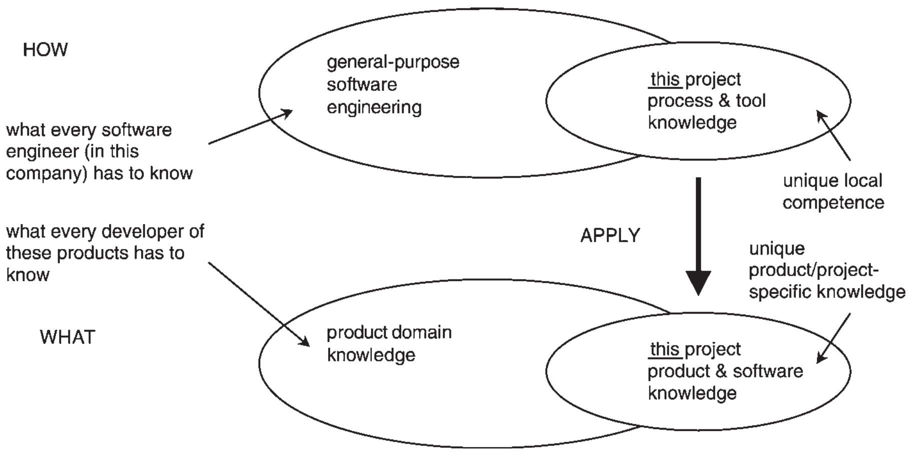
HOW
what every software engineer (in this company) has to know
general-purpose software engineering
this project process & tool knowledge
unique local competence
APPLY
what every developer of these products has to know
unique product/project-specific knowledge
WHAT
product domain knowledge
this project product & software knowledge

Fig. 1 *Embedded software project knowledge domains*

Figure 3 is a synthesis of all those considerations. The software development is guided by the system development function. The software project team must then acquire information about the system design (Fig. 3, item 9). The software requirements comprise the system requirements allocated to software typically accompanied by external specifications (10, e.g. telecommunications standards) and information about the related systems (5) in the product use context (6). It is anyway important that the software team understands the real needs (11).

The software development must follow the related hardware development (8, e.g. hardware/software interfaces). Also there are often hardware-specific test support needs (7). A large software product is hardly ever created totally from scratch but usually there are various reusable assets and platform components involved – both in-house (2) and externally developed (3). The software project team must then learn to incorporate and possibly also modify them. Unless the project is creating the very first release of the product, there is an existing base software version (1). The team must learn it in enough detail before they can start modifying and extending it. In turbulent cases the base software and some of the external components (e.g. software platform) may in addition be under development concurrently with the product software.

Furthermore, it is not enough to have the product knowledge but also the process knowledge is needed (4). Note that the software development team must have a general understanding of the related systems and hardware engineering processes, for instance, to be able to deal with the interface specifications. A common terminology base helps to avoid misunderstandings [24, Chap. 3]. Finally, the experience and skills of the project people are valuable – sometimes even critical – inputs to the software development.

Like the inputs, the software development output knowledge is in practice a complex set of pieces of information. Figure 4 depicts this knowledge space.

The output knowledge comprises not only the software product related information, but also project and process related items as well as product development organisational elements. Note in particular that often in practice some design knowledge remains outside the formal documentation (e.g. private notes) and even undocumented. In embedded software development this tends to be the case with the software engineer's understanding of the specific target hardware behaviour learned during the development. Finally, during the software project new process and tool developments may have been carried out. Notably the project team has gained new personal experiences and skills.

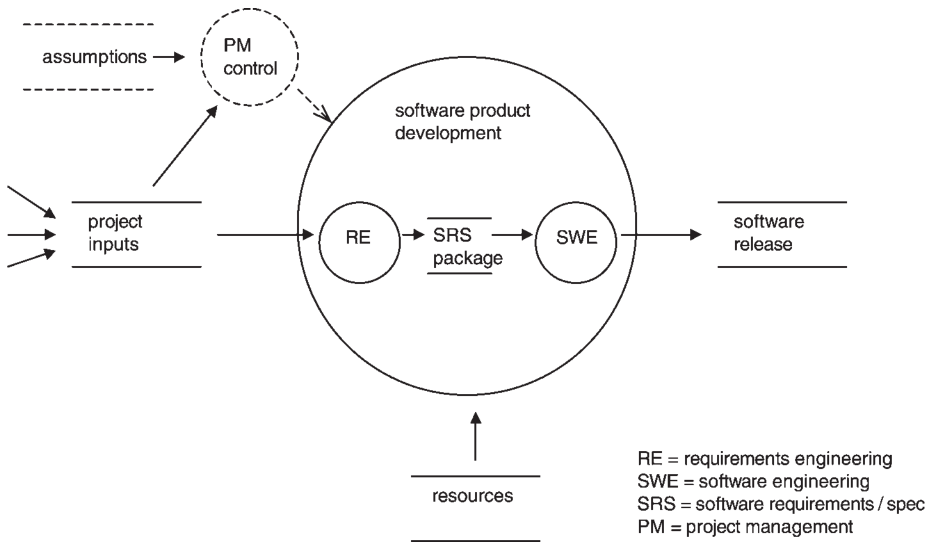
assumptions
PM control
software product development
project inputs
RE
SRS package
SWE
software release
resources
RE = requirements engineering
SWE = software engineering
SRS = software requirements / spec
PM = project management

Fig. 2 *Software product development process model*

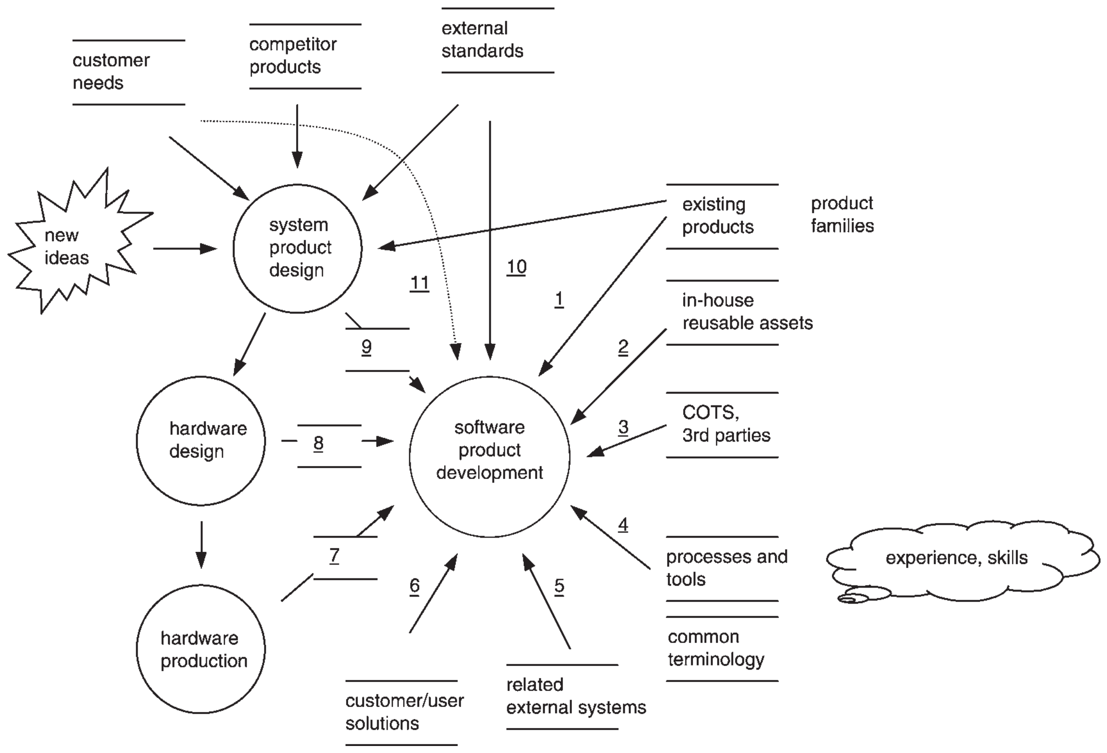

Fig. 3 *Embedded software project knowledge context model (inputs)*

It is important to understand that in large product development organisations the embedded software development projects do not work in isolation. We must actually consider not only individual software projects but also their organisational interplay over time:

• The current project gets all the necessary information efficiently.
• The next projects get the information, ensuring that the project knowledge is not lost.
• Organisational learning, ensuring that the knowledge accumulated during the product development is not lost but reused and shared.

In particular, in the case of complex system products released in upgrade steps over a long period of time we have a chain of interdependent projects, as illustrated by Fig. 5. Note also that even the very first release project hardly ever starts from scratch but there may even be a long history of preceding product development projects more

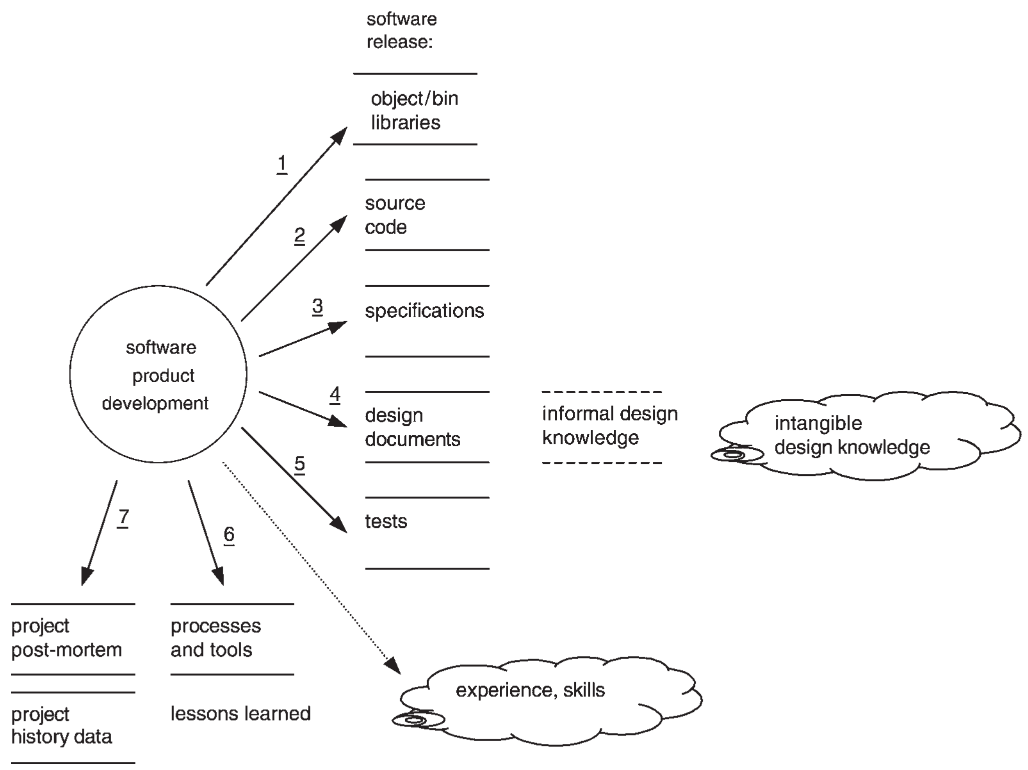

Fig. 4 *Embedded software project knowledge context model (outputs)*

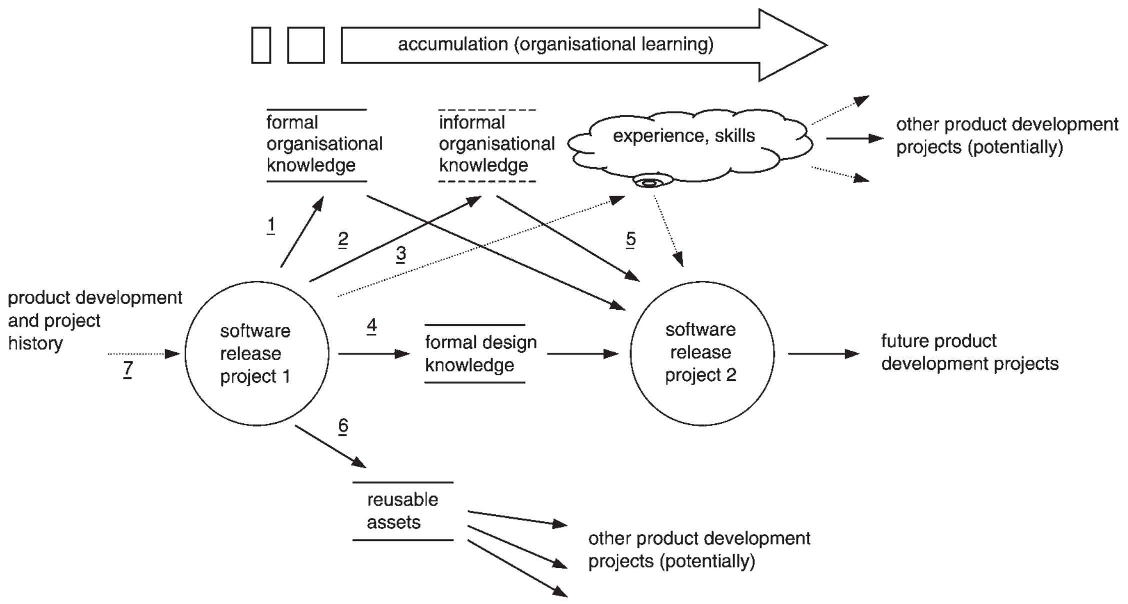
accumulation (organisational learning)
formal organisational knowledge
informal organisational knowledge
experience, skills
other product development projects (potentially)
1
2
3
5
product development and project history
7
software release project 1
4
formal design knowledge
software release project 2
future product development projects
6
reusable assets
other product development projects (potentially)

Fig. 5 *Software project knowledge accumulation/sharing model*

or less related to the current product. Furthermore, the current development may generate new reusable assets to other related product developments creating a complex network of knowledge sharing and accumulation.

3.2 Managing the knowledge

We now propose two pragmatic methods for managing embedded software development knowledge in practice with respect to the conceptual framework described in Section 3.1. These methods have been developed based on our earlier empirical experience in industrial embedded software production for telecommunications network equipment (see Section 4.1).

Our first method (Method A) helps to identify and consequently provide the necessary knowledge of the project team. Basically we have a checklist for systematic management of each software development process area. Table 1 is an outline excerpt of such a table. The items (rows) are in this case grouped according to the ISO/IEC 15504 reference model (SPICE) which covers all the relevant process areas [25]. Each process area is then accompanied by a set of questions concerning the related knowledge elements. Note that the knowledge items in Table 1 can be mapped to Figs. 3 and 4. The idea is to go through the questions and answer them from the viewpoint of each member of the project team. The answers are then recorded for each person (columns). The answer range is in the simplest case just binary (Yes/No), indicating whether that person needs to know that particular item or not. This simple form can be extended for instance by considering not only the plain needs of the persons but also the right time when that knowledge is needed relative to the project life-cycle. The table can further be augmented by defining where exactly the information for each question item can be found in practice (e.g. document identifiers, web-links).

Overall this knowledge management method is most useful during the early phases of the project. It can be used for staffing and project team training purposes as a planning aid, but it can also be used more generally for organisational software process improvement (SPI) purposes by identifying possible lack of competencies, missing work instructions and other imperfect process areas.

Our second method (Method B) helps to actually use the knowledge during the software project life-cycle. Communication gaps and missing information have been recognised to be typical project failure factors in large-scale project work. What we need then is a method of ensuring that the critical information is both produced by the right persons at the right time, and utilised by all the relevant parties. Change management is also intrinsic to this process.

Responsibility charts are well known tools for project planning in general. For example, Andersen has proposed so-called project responsibility charts for systematic identification of project milestone responsibilities [26]. We can now adopt similar ideas to embedded software project knowledge management by defining the producers and consumers of the key information and mapping them together as a chart. Table 2 illustrates such a project knowledge sharing chart.

With this tabular method we can clearly see the knowledge item dependencies of each member of the project team. Note that for a typical product development software project there are many project external connections. For embedded software development it is in particular important to define the hardware and systems engineering dependencies (c.f. Fig. 3). Both tangible and intangible knowledge items are included here since often in practice not all useful information is tangible (e.g. informal previous project experiences). Some types of knowledge are not easy to disseminate on paper alone, for example, but need in addition face-to-face communication. Not only product related information, but also process knowledge is important to share. We can extend the basic method shown in Table 2, for instance by defining when exactly that piece of information is needed. The same table could furthermore be used for status tracking of the knowledge item production indicating when each item is produced and needed.

4 Evaluation

4.1 Reflecting earlier empirical observations

We have been observing and analysing certain real-life telecommunications product development embedded software projects over the years within Nokia Networks. These observations form the empirical background on which our knowledge management approaches are based. At the time of this writing we are not ready to present empirical validation data of our new proposals. However, we can use

Table 1: Method A: Embedded software project knowledge management planning template

	P. Kettunen (PM)	N.N. (Designer)
Customer–supplier process cat (CUS):		
Acquisition		
Supply		
Who are our customers (external and internal)?		
Requirements elicitation		
What do the customers really want from us?		
Who is responsible for the elicitation of the customer requirements?		
Operation		
Engineering process cat (ENG):		
System requirements analysis and design		
Where do I get my system requirements?		
How do I know the software architecture (and system design)?		
Software requirements analysis		
Which items (documents) comprise my software requirements package?		
How are the requirements managed (changes)?		
Software design		
What design methods and tools do I use?		
How do I change the component/subsystem *external interfaces?*		
Where can I find the hardware data sheets (if any)?		
Software construction		
What compilers etc. tools do I use?		
What implementation rules do I have to obey (e.g. coding standards)?		
Software integration		
What kind of integration and testing should I do?		
Software testing		
Where do I get the target test hardware?		
System integration and testing		
How do I interface with the SW integration and system I&V?		
System and Software maintenance		
What software platform dependencies do I have to follow?		
Support process cat (SUP):		
Documentation		
How do I manage (e.g. version) my documents?		

The following is a list of software product development process areas based on the ISO/IEC 15504 Reference Model. The accompanying questions (in italics) are supposed to help identify the practical information needs of those areas. Each member of the project team should know those things from their point of view
For each question, answer considering the actual needs of that person:
-n/a no need to know
-Yes: Need to know. When is the knowledge needed?

Table 2: Method B: Embedded software project knowledge sharing chart

	Software project internal		Software project external		
Knowledge items\actors	P. Kettunen (PM)	N.N. (Designer)	(System specifier)	(Hardware manager)	(Quality manager)
Previous projects history	User	n/a	n/a	n/a	Provider
Software specification A	Author	Reader	n/a	n/a	n/a
System specification B	Reader	n/a	Responsible	Contributor	n/a
Hardware data sheet	n/a	Reader	Reviewer	Responsible	n/a
ASIC hardware behaviour	n/a	User	n/a	Provider	n/a
Standard operating procedure	Reader	Reader	n/a	n/a	Responsible
User’s Guide	n/a	Author	n/a	Reviewer	Reviewer
Test process experience	Provider	User	n/a	n/a	n/a

Table 3: History data based evaluation of the constructs and methods

Case no.	Old study observations	New construct reflections
1	• underspecified system requirements	• Fig. 3, item 9; Method B
	• lack of hardware behaviour knowledge	• Fig. 3, item 8; Method A
2	• unclear, changing requirements	• Fig. 2 (SRS); Method B
	• architectural complexities	• Fig. 3, item 9
	• testing	• Fig. 4, item 5
	• multisite organisations, communication	• Method B
3	• project scoping	• Fig. 2 (PM); Method A
	• product release planning	• None!

those historical records here to evaluate the proposed constructs and methods hypothetically to show how our ideas could tackle similar situations in the future.

Our first study (Case 1) analysed one particular embedded software development area of a transmission product line (no longer with Nokia) [22]. This software subarea has close connections with the special-purpose hardware (ASIC circuits). By analysing the software development defect data, we concluded that the technical functional requirements are sometimes underspecified at the system level. The correct understanding of the underlying special-purpose hardware behaviour should be ensured.

Our second study (Case 2) investigated embedded software project success and problem factors with respect to the software development cycle-time within a radio network element product development [23]. A group of project managers rated a given set of project factors according to their perceived importance. Unclear, changing requirements specifications and architectural complexities were rated to be the most difficult problems. Also testing tended to be a troublesome area. Multisite organisations and communication gaps caused problems.

Finally, our third study (Case 3) surveyed project problem areas within a unit developing embedded software for various transmission devices [27]. The project managers were asked to evaluate whether their observed project problems were related more on project management or software engineering technical issues. This survey concluded that overall project management issues (e.g. project scoping) appeared to be more troublesome, and more problems originate for project external reasons (e.g. product release scheduling).

Table 3 summarises our historical case study findings, and links them to the proposed constructs. Most of the problem observations seem to have a direct correspondence, indicating that we may have managed to capture the essential knowledge factors at least within this particular PD context, and the new proposals could probably have helped to avoid some of those problems.

4.2 Comparing and contrasting the constructs and methods

Thorough understanding of the product domain knowledge is a well known success factor of embedded software development as pointed out by Curtis *et al.*, for instance [15]. Figure 1 reflects this.

Software requirements engineering (RE) of market-driven embedded software product development has certain inherent difficulties. In particular, the user needs and requirements apply to the product as a system, and not directly to the embedded software part. There is evidence that in terms of overall productivity it is often advantageous to put more effort on the requirements and specifications [28]. Note that the 'SRS package' in Fig. 2 comprises the explicit requirements documents as well as all other relevant supplementary information items, and even tacit knowledge. The observation that in practice product information – in particular the user requirements – tends to evolve during the development has been acknowledged by many studies [29–31: Chap. 16]. For those reasons we have emphasised the RE activity in Fig. 2.

Figures 3 and 4 refine these principles in more detail. We have taken into account the hardware and systems engineering dependencies. Note also that the various knowledge items outlined in Figs. 3 and 4 are not static, but they have evolution life-cycles during the projects [32, Chap. 6]. The data-flow notation expresses this naturally.

It is important to realise that the embedded software development is actually a continuous process in the case of long-term development of large complex system products (Fig. 5). Systematic project and organisational memories are then needed. Large-scale reuse could potentially increase the productivity and reduce cycle times significantly, in particular in product line environments. However, often many nontechnical obstacles tend to hinder this [33]. Organised management of the reusable assets knowledge (Fig. 5, item 6) could help this, for instance by ensuring that all the developers are aware of the available assets in the first place. Often a major problem is the initial asset codification and contribution to the organisational repository [34].

For our knowledge management Method A (Table 1), the choice of the SPICE reference model is not critical. In fact, other suitable groupings could be based, for example, on the CMMI [9]. A key point is to recognise the special areas of embedded software development. This is also reflected in Method B (Table 2). Organisational structures and possible communication barriers can be overcome by making the knowledge transfer needs explicit, as shown in Table 2. In general, project communication plans serve similar purposes [35]. In particular, disseminating tacit knowledge tends to require face-to-face discussions [36–38]. RaPiD7 is a document production method based on bringing the key personnel together in a planned way [39].

It is a well known observation that software development is human-intensive and 'soft' factors contribute greatly to project success [40]. Even some 'creative chaos' is a prerequisite for innovation [11]. In that respect our attempts to systematise project knowledge management do not have very much novel to offer. However, even in highly flexible innovative environments certain mechanistic routines are useful. From that point of view our proposals help to build a more productive project infrastructure.

Although our empirical background is in telecommunications products embedded software development, most of these constructs should be generally applicable. Other product types may require certain domain-specific knowledge items but the methods and abstract information models remain basically the same.

5 Conclusions and further research

Managing the intellectual capital (IC) is becoming increasingly important in knowledge-intensive organisations. Consequently, knowledge management of the software projects should be an integral part of the activities of the modern efficient product development organisations working in turbulent business environments. The key is to (re)combine the knowledge and routines in efficient ways according to the current situation [41].

In this paper we have outlined the basic knowledge management needs of embedded software projects in product development environments. The key is to realise the software project external dependencies with the related systems, hardware and other context-specific sources and sinks of information. Both tangible and intangible knowledge items must be covered. Even tacit knowledge needs to be considered. Not only product knowledge but also the related process knowledge is important for the embedded software project team. Experience of the team is a valuable organisational asset.

We have proposed some pragmatic methods for managing embedded software project team knowledge based on our earlier empirical experiences. The study does not contribute major theoretical novelties, but the emphasis is rather in recognising the real-life problems and practical combination and application of basic methods. However, often in industrial NPD environments the key challenge is really to find some effective yet easily applicable ways of implementing process improvements.

Further work remains to be done, however:

- Table 1 and 2: empirical validation of the proposed methods in future projects. How useful are they really? A measure could, for example, be the number of defects having the root cause in the knowledge management category
- Figures 3 and 4: more detailed characterisation of the information models and their change management
- adding measurements (e.g. how well a project team masters the relevant knowledge)
- Figure 5: managing tacit knowledge (e.g. project team experience) more systematically
- taking multisite project knowledge management needs into account [16].

The future trend of embedded software product development is to increasingly use – when feasible and cost-effective – standard hardware modules and COTS. Also new horizontal component markets are emerging. The product release cycle-times have become ever shorter, demanding more agile and flexible software development. Active knowledge management is then likely to be an increasingly important part of embedded software development practice.

6 References

1 White, S.F., Melhart, B.E., and Lawson, H.W.: 'Engineering computer-based systems: meeting the challenge', *Computer*, 2001, **34**, (11), pp. 39–43
2 De Meyer, A., Loch, C.H., and Pich, M.T.: 'Managing project uncertainty: from variation to chaos', *MIT Sloan Manage. Rev.*, 2002, (Winter), pp. 60–67
3 Ulrich, K.T., and Eppinger, S.D.: 'Product design and development' (McGraw-Hill, USA, 2000)
4 Smith, P.G., and Reinertsen, D.G.: 'Developing products in half the time: new rules, new tools' (John Wiley & Sons, USA, 1998)
5 Nambisan, S., and Wilemon, D.: 'Software development and new product development: potentials for cross-domain knowledge sharing', *IEEE Trans. Eng. Manage.*, 2000, **47**, (2), pp. 211–220
6 Mellis, W.: 'Software quality management in turbulent times – are there alternatives to process oriented software quality management?' *Softw. Qual. J.*, 1998, **7**, (3), pp. 277–295
7 Mikkonen, T., and Pruuden, P.: 'Flexibility as a design driver', *Computer*, 2001, **34**, (11), pp. 52–56
8 Klein, B., and Meckling, W.: 'Application of operations research to development decisions', *Oper. Res.*, 1958, **6**, pp. 352–363
9 http://www.sei.cmu.edu/cmmi/ accessed 6 October 2003
10 Jaaksi, A., Aalto, J.-M., Aalto, A., and Vättö, K.: 'Tried & true object development' (Cambridge University Press, UK, 1999)
11 Nonaka, I., and Takeuchi, H.: 'The knowledge-creating company: how Japanese companies create the dynamics of innovation' (Oxford University Press, USA, 1995)
12 Rus, I., and Lindvall, M.: 'Knowledge management in software engineering', *IEEE Softw.*, 2002, **19**, (3), pp. 26–38
13 Skuce, D.: 'Knowledge management in software design: a tool and a trial', *Softw. Eng. J.*, 1995, **10**, (5), pp. 183–193
14 Lindeman, D.D., and Moore, D.: PDM: 'An enabling technology for integrated product development'. Proc. IEEE Annual Symp. on Reliability and Maintainability (RAMS), Anaheim, CA, January 1994, pp. 320–326
15 Curtis, B., Krasner, H., and Iscoe, N.: 'A field study of the software design process for large systems', *Commun. ACM*, 1988, **31**, (11), pp. 1268–1287
16 Desouza, K., and Evaristo, R.: 'Global knowledge management strategies', *Eur. Manage. J.*, 2003, **21**, (1), pp. 62–67
17 Lynn, G.S., Reilly, R.R., and Akgün, A.E.: 'Knowledge management in new product teams: practices and outcomes', *IEEE Trans. Eng. Manage.*, 2000, **47**, (2), pp. 221–231
18 Bűchel, B., and Raub, S.: 'Building knowledge-creating value networks', *Eur. Manage. J.*, 2002, **20**, (6), pp. 587–596
19 Boehm, B.W.: 'A spiral model of software development and enhancement', *Computer*, 1988, **21**, (5), pp. 61–72
20 Mäkäräinen, M.: 'Software change management processes in the development of embedded software'. Dissertation, University of Oulu, Finland, 2000 (Technical Research Centre of Finland, VTT Publications 416)
21 Heikkinen, M.: 'A concurrent engineering process for embedded systems development'. Licentiate thesis, University of Oulu, Finland, 1997 (Technical Research Centre of Finland, VTT Publications 313)
22 Kettunen, P.: 'Software requirements engineering for an embedded system development'. Licentiate thesis, Helsinki University of Technology, Finland, 2000
23 Kettunen, P.: 'Towards rapid development of embedded telecommunications system software products'. Individual study paper (unpublished), Helsinki University of Technology, Finland, 2001
24 Forsberg, K., Mooz, H., and Cotterman, H.: 'Visualizing project management – a model for business and technical success' (John Wiley & Sons, USA, 2000)
25 'Information technology – software process assessment – Part 2: A reference model for processes and process capability'. Technical report ISO/IEC TR 15504-2, 1998
26 Andersen, E.S.: 'Warning: activity planning is hazardous to your project's health!' *Int. J. Proj. Manage.*, 1996, **14**, pp. 89–94
27 Kettunen, P.: 'Project QP survey'. Internal study (unpublished), Nokia, 2002
28 Blackburn, J.D., Scudder, G.D., and Van Wassenhove, L.N.: 'Improving speed and productivity of software development: a global survey of software developers', *IEEE Trans. Softw. Eng.*, 1996, **22**, (12), pp. 875–885
29 Rowen, R.B.: 'Software project management under incomplete and ambiguous specifications', *IEEE Trans. Eng. Manage.*, 1990, **37**, (1), pp. 10–21
30 Boehm, B.W.: 'Seven basic principles of software engineering', *J. Syst. Softw.*, 1983, **3**, (1), pp. 3–24
31 Weinberg, G.M.: 'Quality software management, Vol. 4: Anticipating change' (Dorset House Publishing, USA, 1997)
32 Royce, W.: 'Software project management' (Addison-Wesley, USA, 1998)
33 Boehm, B.W.: 'Managing software productivity and reuse', *Computer*, 1999, **32**, (9), pp. 111–113
34 Davenport, T.H., Thomas, R.J., and Desouza, K.C.: 'Reusing intellectual assets', *Ind. Manage.*, 2003, **45**, (3), pp. 12–17
35 'A guide to the project management body of knowledge (PMBOK guide)'. Project Management Institute, USA, 2000
36 Desouza, K.C.: 'Barriers to effective use of knowledge management systems in software engineering', *Commun. ACM*, 2003, **46**, (1), pp. 99–101
37 Desouza, K.C.: 'Facilitating tacit knowledge exchange', *Commun. ACM*, 2003, **46**, (6), pp. 85–88
38 Koskinen, K.U., Pihlanto, P., and Vanharanta, H.: 'Tacit knowledge acquisition and sharing in a project work context', *Int. J. Proj. Manage.*, 2003, **21**, pp. 281–290
39 Kylmäkoski, R.: 'Efficient authoring of software documentation using RaPiD7'. Proc. ICSE, 2003, pp. 255–261
40 McConnell, S.: 'Quantifying soft factors', *IEEE Softw.*, 2000, **17**, (6), pp. 9–11
41 Ståhle, P., and Grönroos, M.: 'Dynamic intellectual capital' (Werner Söderström Corporation, Finland, 2000)