JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 2003; **15**:9–25 (DOI: 10.1002/smr.264)

Research

Latitudinal and longitudinal process diversity

Nils T Siebel[1,*,†], Stephen Cook[2], Manoranjan Satpathy[2] and Daniel Rodríguez[2]

[1]*Computational Vision Group, Department of Computer Science, The University of Reading, U.K.*
[2]*Applied Software Engineering Research Group, Department of Computer Science, The University of Reading, U.K.*

SUMMARY

Software processes vary across organizations and over time. Managing this process diversity is a delicate balancing act between creative, healthy diversity and chaos. In this paper, we examine a particular aspect of this issue, namely some relationships between diversity in software processes, software evolution and the quality of software products and processes. Our main contribution is to distinguish between two broad kinds of process diversity, which we call *latitudinal* and *longitudinal process diversity*. To illustrate the differences between these two, we examine the case of a medium-sized system (50 000 lines of C++ code) which has undergone major changes during its lifetime of 10 years. The software was originally developed by an individual academic using a research-oriented process to develop a standalone proof-of-concept system. In a current multi-team project, involving three industrial and three academic partners, the software has been adapted for integration as a subsystem of a near-market product. We suggest ways in which the observed process diversity seems to be linked to a change in the software's propensity for evolution, and we discuss the impact of this on both product and process quality. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: process diversity; process improvement; software evolution; software maintenance

1. INTRODUCTION

It is now widely accepted that most software evolves significantly over its lifetime. One of the implications is that there are relationships between software processes and software evolution; that is, the way in which software is developed and maintained may have longterm effects on the software itself.

*Correspondence to: Nils T Siebel, Computational Vision Group, Department of Computer Science, The University of Reading, Reading RG6 6AY, U.K.
†E-mail: nts@ieee.org

Received 28 June 2002
Revised 25 September 2002

In this paper, we examine a particular aspect of this issue, namely some relationships between diversity in software processes, software evolution and the quality of software products and processes. Our main contribution is to distinguish between two broad kinds of process diversity: *latitudinal* and *longitudinal process diversity*. This classification provides a conceptual framework for a better understanding of process diversity and thereby helps managers to apply specifically tailored countermeasures to control the negative effects of a particular type of diversity. We illustrate these concepts with a case study and suggest ways in which they appear to affect product and process quality.

1.1. Process diversity

A *process* is a collection of activities carried out by people and/or machines that are intended to achieve some desired outcome, are related to each other in time, and have inputs and outputs. For management purposes, processes are often recursively decomposed into *subprocesses* to reach what are called *atomic processes*. The possible granularity of processes covers a very wide range, from high-level business processes to finely detailed processes for software maintenance. In order to understand process diversity, we need to situate software processes in the context of larger business processes.

A process is represented by a *process model* and executed within a *process environment* [1]. The process environment links together the people executing the process (e.g. managers and software developers) and any domain-specific tools that may be used. As processes do not have unique representations, and they can be executed within differing process environments, processes invariably differ. In this paper, we consider *process diversity* as it occurs when a project is executed within different process environments, and we study its impact. This diversity can happen either concurrently (e.g. in multi-team projects) or when a project encounters different process environments during its life-cycle. Process diversity can have negative consequences. Two well-known examples are problems during system integration (as project partners might follow differing software processes), and problems arising from software re-use in a new environment.

Diversity in software processes is usually inevitable, and managing it can be a delicate balancing act. Too much process diversity can lead to chaos, but too little may suppress creativity and lead to missed opportunities. Whilst day-to-day process management will be more concerned with deliverables, schedules and budgets, the management of process improvement should periodically review whether a satisfactory balance between uniformity and diversity in software processes is being achieved. Our paper supports this aspect of process review and improvement by proposing a simple high-level model of process diversity that distinguishes two broad categories that have different sources and impacts.

1.2. Related work

The need for quality improvement and cost reduction in software production and maintenance has led to a research emphasis on *process improvement*. The Software Engineering Institute (SEI) has set up a dedicated group of software process improvement networks (SPINs) with the aim of connecting individuals involved in improving software engineering practice. Research and case studies show how much money an improved software process can save a company [2], but also stress the complexities involved in process improvement [3–6]. A number of process models and standards have been developed to analyse and improve software processes, most prominently the capability maturity model (CMM), but also ISO standards such as ISO 9000 and ISO 15504 (see [7] for a review of these and

other standards). Process improvement is usually carried out using a process model, and a measurement framework like the goal/question/metric (G/Q/M) method [8].

The identification of *software evolution* as a research topic originated in Lehman's pioneering studies of the long-term development of IBM systems in the 1960s [9], which resulted in the formulation of Lehman's widely respected 'Laws of Software Evolution' and his S-P-E classification of information systems [10,11], which we will address in Section 2.3. Many of the core concepts and approaches that are currently used can be traced back to Lehman's work.

In Section 2 we describe our concept and classification of process diversity, establishing a link between process diversity and software evolution. Section 3 introduces the object of our case study, which is further analysed in Section 4. Section 5 covers the lessons learned from our analysis and Section 6 concludes the paper.

2. CLASSIFYING AND MODELLING PROCESS DIVERSITY

Diversity in software processes can occur on many different scales, and its impact on product and process quality can also cover a wide range. In order to understand the likely impact of particular examples of diversity, it is helpful to distinguish two high-level categories.

- *Latitudinal process diversity*. This category describes the kind of variation that occurs when diverse processes operate concurrently within the same project. This is often observed in multi-team projects [12].
- *Longitudinal process diversity*. This is the variation that occurs in software processes over time. An example which is often observed is the transition from development to maintenance phases in a project.

Thus latitudinal and longitudinal process diversity are not properties of individual processes; they are properties of the system (in this case, a software development or maintenance system) that provides the context for processes—the *process environment*. Consequently, these properties may be of greatest interest to people who are responsible for managing, designing or improving processes, rather than directly executing them.

2.1. Latitudinal process diversity

We use the term latitudinal process diversity to refer to the phenomenon of diverse software processes operating concurrently. This type of process diversity is often found in projects that span across company boundaries, and it has its most severe impact when software from more than one project partner is to be integrated together [12]. This section proposes a model for understanding its main dimensions.

2.1.1. Role differentiation

In the simplest kind of software project, the roles of customer, developer and user are subsumed in a single person. However, as projects increase in complexity, these roles tend to differentiate in two ways.

1. The various functional roles (customer, developer, maintainer etc.) become more distinct and behave as separate stakeholders in the system, each having characteristic objectives, concerns and priorities.
   The idea that any view of a software system implies the viewpoint of a particular stakeholder has been incorporated into the recent IEEE Standard 1471-2000 [13].
2. Particular roles become shared by individuals, teams or organizations, who may have different notions about how the role should be carried out.

Both kinds of differentiation can lead to latitudinal process diversity. At any moment, the various units (individuals, teams, departments) within an organization may be involved in:

- performing different processes (because the units occupy different roles, e.g. maintainer, product-line architect, customer); and/or
- performing the same process in different ways (because the units have different cultural or professional approaches to the role's responsibilities, e.g. programmers and technical writers producing system documentation).

2.1.2. Cultural diversity

As an engineering product, software is affected by the environment in which it is produced. A variety of sociological factors (e.g. the socio-cultural backgrounds of team members, the structure and business practices of organizations) may lead to differences in the processes that teams use to produce software—see [14] for examples. Thus cultural diversity of various kinds can lead to latitudinal diversity in software processes. This becomes most apparent in multi-team projects, where pieces of software from different teams need to be integrated into one system.

2.2. Longitudinal process diversity

The relationship between process diversity and software evolution can be seen most clearly in the longitudinal case. A fundamental kind of longitudinal process diversity occurs when software changes in its propensity for evolution. For example, a program may be originally developed as a proof-of-concept for some abstract computation with essentially static requirements. Subsequently, the program may be integrated into an information system that supports an evolving business process. This brings the program within the influences of a more dynamically evolving system, and its software process may need to be adapted accordingly.

The concept of longitudinal process diversity implies a long-term viewpoint on software processes. In this view, the detailed definitions of individual steps in a software process are less important than the overall scope and configuration of the process considered as a system. Over the long term, factors such as the sources of change to the software product and the nature of the feedback paths in the process become more influential; these are also important factors in software evolution [15]. Conversely, factors such as the choice of programming language or the use of specific software engineering techniques, like design reviews or code inspections, become less influential. Thus a longitudinal process diversity viewpoint has strong similarities with, and complements, a software evolution viewpoint. They share many concerns (e.g. system maintainability) and modelling techniques (e.g. system dynamics), but differ in whether the viewpoint's focus is on the process or its product.

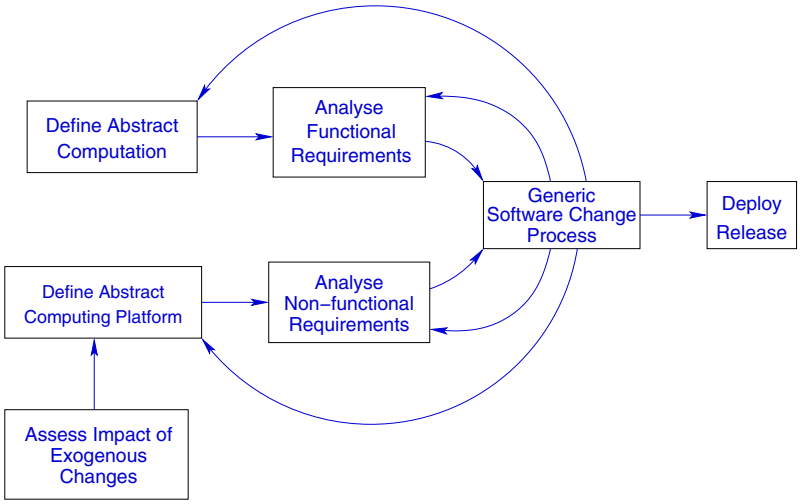


Figure 1. Software process model, Stereotype 1—low propensity for evolution.

2.3. Evolution-oriented models of software processes

To illustrate the relationship between longitudinal process diversity and software evolution, we have developed two stereotypes of the software process that are distinguished by their propensity for software evolution. *Stereotype 1* models a process that is appropriate for software with a low propensity for evolution, *Stereotype 2* models a process for software with a high propensity for evolution. These two represent the extremes of the range that is likely to be found in practice. Longitudinal process diversity can be understood as movement within this range during the lifetime of a software product.

Both stereotypes use the viewpoint described in the previous section to situate the engineering process of changing software in a broader context. Consequently, most of the details (and diversity) of software engineering techniques have been subsumed into a single node *generic software change process*. This is a generic placeholder; it represents a software change process in relation to any life-cycle model (e.g. the *waterfall model* or *agile processes* like XP) [16]. The main features of our process models are illustrated by schematic process diagrams in which the boxes represent (sub)processes and the arrows represent information flow between them. The diagrams are intended to highlight the distinguishing characteristics of each model and not to be complete specifications of software processes.

2.3.1. Stereotype 1: low propensity for software evolution

The process model shown in Figure 1 is intended for software products with a low propensity for evolution. In terms of Lehman's S-P-E taxonomy, they fall into the S-type (specified) and P-type

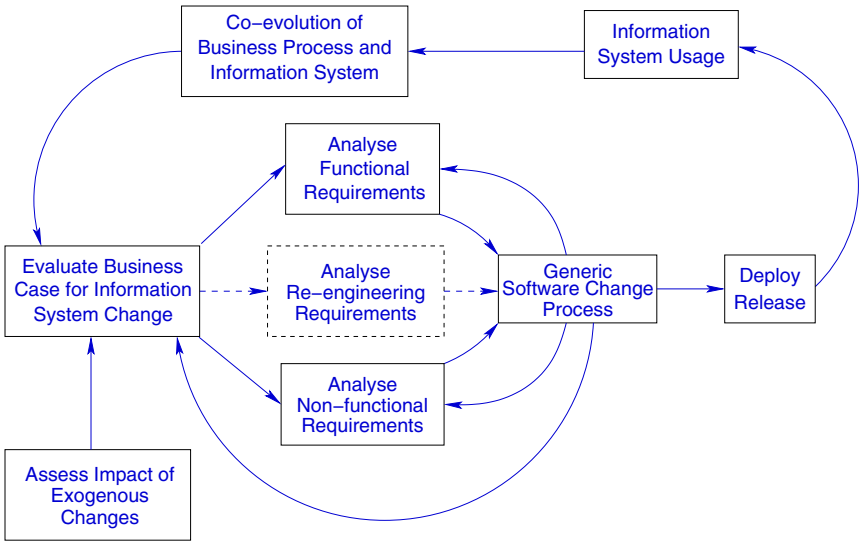


Figure 2. Software process model, Stereotype 2—high propensity for evolution.

(problem-solving) categories. Their functional requirements are likely to evolve slowly, if at all. Implementations of P-type software may need to be adapted occasionally to take account of changes in their technical environments. In the case of an S-type program, by definition, its requirements provide a complete description of the problem to be solved and its implementation does not require any design compromises. Consequently, there are no strong pressures for such programs to evolve. For a P-type program, the first condition is slightly relaxed; its functional requirements may be an abstraction (which can be redefined) from the problem to be solved, but the problem itself is static. A P-type program has significant non-functional requirements that must be reconciled with the functional requirements by the program's design. Changes in the technical environment can require the program to be adapted. For example, if a manufacturer stops supporting particular hardware or compilers, a program which depends on them may have to be ported to a new platform if it is not to become unusable. Conversely, when new technical capabilities become available, they can trigger a reconsideration of design compromises, and possibly a redefinition of the problem abstraction.

2.3.2. Stereotype 2: High propensity for software evolution

When software is embedded in an information system that supports a business (or social) process, its software process inevitably becomes more complex than the model in Stereotype 1. Parnas has used the notion of *software aging* to characterize the additional pressures that affect software and software processes in this situation [17].

One approach to understanding these differences in process can be found in Lehman's 'Laws of Software Evolution' [11] and related studies (e.g. [15]). In Figure 2 we show Stereotype 2, which is a model of the software process that takes account of the findings from Lehman's work that are most relevant to longitudinal process diversity [11]. The software process modelled in Stereotype 2 is intended for programs which fit into Lehman's E-type (embedded or evolving) category [10]. The increase in complexity in this model compared with Stereotype 1 arises from three principal characteristics of E-type programs.

1. Both initial development and subsequent changes to the software are affected by an evolving business case. The business case can change at any time for reasons that ultimately are open-ended.
2. The usage of the system produces business benefits. If the system is successful, this will tend to stimulate co-evolution [18] of the information system and the business process. This will often result in unanticipated changes in the system's requirements.
3. One of the side-effects of the increased propensity for evolution is that periodically it may be necessary for the business case to include some re-engineering of the information system [19]. This path through the process model is shown with dashed lines and boxes in Figure 2.

3. DESCRIPTION OF THE PEOPLE TRACKING SOFTWARE

The software package studied for this article is the people tracking subsystem of ADVISOR[‡], which is an integrated system for automated surveillance of people in underground stations (Figure 3(a)).

ADVISOR is being built as part of a European research project involving three academic and three industrial partners. The task of the people tracking subsystem is to automatically analyse images from one or more camera inputs. A stream of video images is continuously fed to the system, and the subsystem detects all the people in the image and tracks them in realtime. Figure 3(b) shows an example of the output from the people tracker.

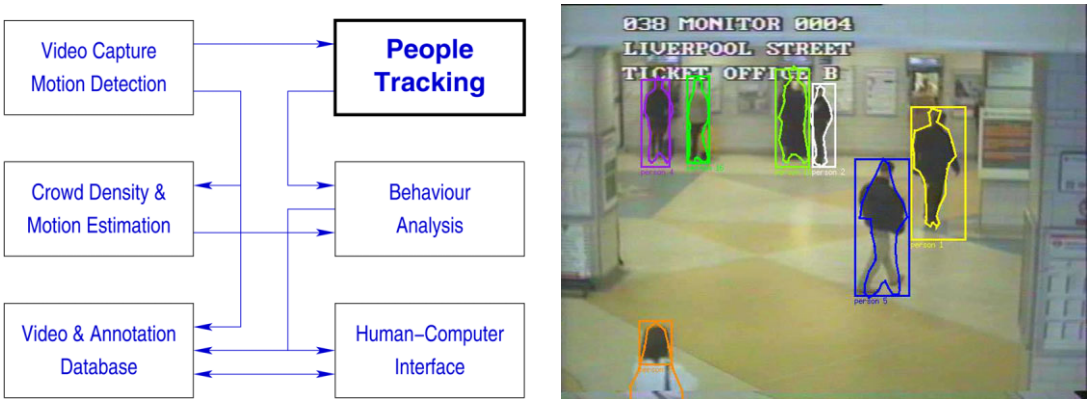
3.1. Brief history

The evolution of the people tracker can be divided into three main phases, extending over 10 years. Figure 4 shows an overview of these phases.

3.1.1. Phase 1: initial development (1993–1995)

The original people tracker was written at the University of Leeds in 1993–1995 using C++ running under IRIX on an sgi platform. It was a research and development system and a proof of concept for a PhD thesis [20]. The main focus during development was on functionality and experimental features which represented the state-of-the-art in people tracking. For code development, a simple process cycle

[‡]ADVISOR—Annotated Digital Video for Intelligent Surveillance and Optimized Retrieval.



(a) (b)

Figure 3. People tracking as one of six subsystems of ADVISOR: (a) ADVISOR system overview; (b) People tracking results.

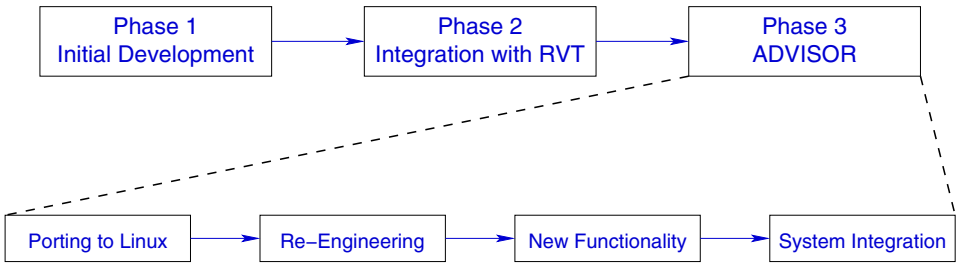


Figure 4. Main development and maintenance phases of the Reading people tracker.

was used to develop the software. The only documentation generated was a short programmer's manual (about five pages) describing how to write an application using the people tracker software.

3.1.2. Phase 2: integration with a vehicle tracker (1995–1998)

In 1995–1998 the code was used in a collaboration between the Universities of Leeds and Reading. The software was changed to inter-operate with the Reading vehicle tracker which ran on a Sun/Solaris platform [21]. Little functionality was changed or added during this time and no new documentation was created. Most of the functional changes made to the People Tracker during this phase were carried out by its original developer.

3.1.3. Phase 3: re-design and integration into the ADVISOR surveillance system (2000–2002)

Since 2000, the people tracker has been adapted for use within the ADVISOR system shown in Figure 3(a). Its planned use within ADVISOR carried many requirements the original implementation could not meet. Most of the difficulties arose from the fact that the people tracker would now have to be part of an integrated system. Using the people tracker in the ADVISOR system also meant moving it 'from the lab to the real world' which necessitated many further changes (e.g. porting it from sgi to a PC running GNU/Linux to make economical system integration feasible).

An analysis of the system revealed that the current implementation was not maintainable enough to be adapted to the new requirements. Therefore it was decided to re-engineer the software. After the re-engineering step, new functionality was incorporated into the software and the missing software artefacts were recovered. Among these is extensive documentation which describes the new software process which is now to be followed for all maintenance work. As the last step within Phase 3, the people tracker was integrated into ADVISOR.

3.2. Current status and outlook

An important result of the re-engineering and documentation process is a new software process which is now in place. This documented software process defines:

- detailed coding standards, including design patterns [22];
- configuration management (includes the use of cvs for concurrent version management);
- a number of 'HowTos' which document how to implement new classes for the most important functionalities (e.g. acquiring images, tracking).

One can compare the software process followed in Phase 1 with the one defined by the capability maturity model (CMM) [23] in 'Level 1', while some *key process areas* are now comparable to those in CMM 'Level 3'. The current (June 2002) status of the people tracker can be summarized as follows.

- The software is fully operational with all necessary functionality implemented.
- It can run either in standalone mode or integrated within ADVISOR.
- The re-engineered people tracker exhibits a high level of maintainability and all software artefacts are consistent with the implementation.
- The code is being maintained under GNU/Linux. For integration purposes within the ADVISOR project, releases of the people tracker subsystem have been compiled and integrated under Microsoft Windows 2000.

The testing phase for the tracker running in standalone mode has been successfully completed. The testing phase for its use in the integrated system is in progress. Within a few months from now, a prototype of the ADVISOR system will be tested in underground stations in Brussels and Barcelona. There are a number of proposals for extending the use of the tracking system. An example is to use it to monitor the activity of a species of rare bat [24].

4. ANALYSIS OF THE PROCESS DIVERSITY FOR THE PEOPLE TRACKER

Over the 10 years that the people tracker software has been maintained and used, a wide range of software processes have been employed. In the following sections, we examine this phenomenon in

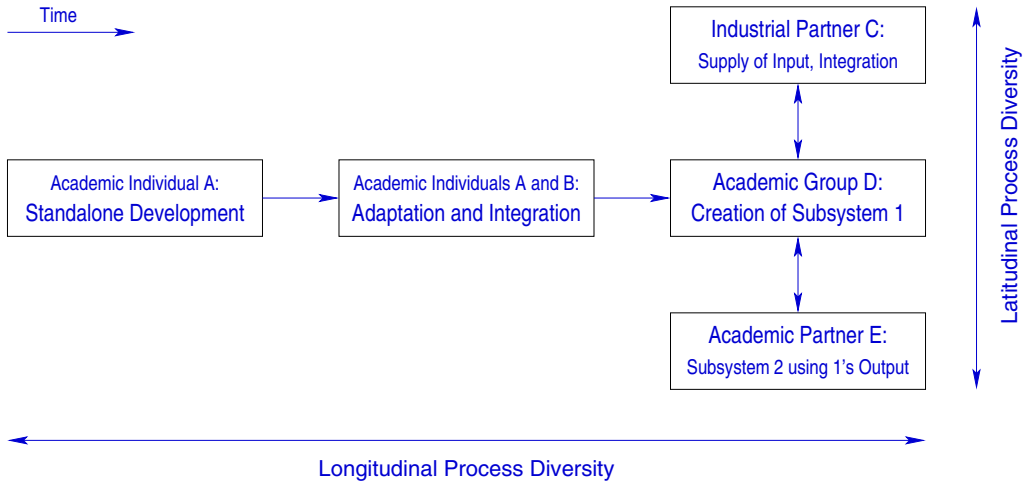


Figure 5. Latitudinal and longitudinal process diversity.

more detail, differentiating between the latitudinal process diversity experienced during Phase 3 and the longitudinal process diversity occurring over all three phases. The relationships between the phases of the people tracker's history and its latitudinal and longitudinal process diversity are summarized in Figure 5. We analyse whether and how the impact of process diversity on the project differs with its type—latitudinal or longitudinal.

4.1. Latitudinal process diversity: diversity between co-operating groups

The use of the people tracker within the ADVISOR project led to increased latitudinal process diversity in Phase 3 as compared with Phases 1 and 2. The principal change was that the academic maintainers, denoted by 'D' in Figure 5, had to co-operate with an industrial partner 'C', and also with an academic partner 'E' in a different country. This meant more actors with distinct roles, as well as different (kinds of) organizations sharing roles, as discussed in Section 2.1.1. In the following description, we will concentrate on these three out of the six groups within ADVISOR, as the others do not directly interface with the people tracking subsystem (cf. Figure 3(a)).

In the collaboration with the industrial partner 'C', the maintainers of the people tracker and their partners experienced many 'cultural' differences. For example, there were differences in the partners' approaches to the use of standards and choices of tools and platforms. The new people tracker subsystem was maintained under GNU/Linux on a PC platform. Academic group *D* adhered to the ISO C/C++ standard, compliant with ISO/IEC 9899:1999(E) [25]. Towards the end of the development, all the design and the implementation were consistent with each other. The module developed by industrial partner 'C', however, was developed under Microsoft Windows 2000 using Microsoft Visual C++.

A consequence of these differences is that the people tracker code, maintained under GNU/Linux, had to be compiled with Microsoft Visual C++. It became clear that Microsoft Visual C++ could not

compile the ISO C code as its compiler is not compatible with that standard. The resolution of this incompatibility required resources that neither partner had budgeted for.

4.2. Longitudinal process diversity and transition between process models

As the people tracker evolved through the three phases described above, its process model changed accordingly. During Phase 1 the process was similar to Stereotype 1 (see Section 2.3.1); during Phase 3 it more closely resembled Stereotype 2 (see Section 2.3.2).

During Phase 1, the main design and programming work was carried out by an academic individual. The goal was to develop a people tracking application which would excel existing applications in performance. During the process cycle, testing and development were closely connected, and both were in the hands of a single person. Being an academic research project, there were also few fixed deadlines for the delivery of code. This meant that all parts of the code could be freely changed at any given time, and many modules were actually changed or re-written many times before the final delivery. Once the software and the PhD thesis were delivered, no functional changes needed to be carried out.

Phase 2 only brought a slight change in the software process. The system was adapted to incorporate a few new requirements, and the interaction with the vehicle tracker module was very localized—the two programs were not even compiled together. The original developer of the software from Phase 1 was involved and carried out many of the functional changes. This reduced the necessity for a change of processes from Phase 1 to Phase 2 as the necessary knowledge about the system was available without the creation of documentation etc.

None of the people involved in Phases 1 and 2 participated in Phase 3. This resulted in Phase 3 being completely de-coupled from Phases 1 and 2, the only linking element over time being the source code. Because there was practically no documentation available for the software, the design methodologies employed during the process were necessarily different from the ones employed before.

The following new requirements necessitated the creation of a process cycle which is close to Stereotype 2 (cf. Figure 2).

- Close integration with another ADVISOR module required interaction with project partners; interfaces needed to be agreed and delivery deadlines had to be kept.
- The significant functional changes necessitated re-engineering analysis and changes through refactoring and re-design. For example, one new requirement was that more than one camera could be used. The use of global variables in the Phase 2 code made this impossible.
- The people tracker subsystem had to be de-coupled from software- and hardware-dependent functions like sgi hardware graphics routines so that it could be maintained under GNU/Linux, but used (within ADVISOR) under Microsoft Windows.
- A strict project schedule with deliverables and milestones meant that a further feedback loop was introduced; based on the deliverables and steering committee decisions, software requirements were changed and adapted to previous results.
- The planned use of the people tracker for related, but different applications (like tracking bats [24]) or the scaling of the system for larger applications inspired the generalization of tracking concepts within the software, keeping in mind possible future uses.

Table I. Metrics of the people tracker over time.

Version	LOC	Classes	Of which instantiated	Methods	Global functions
Pre-Phase 2	20 454	154	71 (46.1%)	1 695	271
Post-Phase 2	19 380	154	71 (46.1%)	1 690	271
Phase 3, a	24 797	191	80 (41.9%)	1 913	286
Phase 3, b	25 306	183	97 (53.0%)	1 714	25
Phase 3, c	23 581	178	85 (47.8%)	1 664	24
End Phase 3	16 472	122	82 (67.2%)	1 231	9

4.3. Metrics and further analysis

Table I shows a brief summary of the characteristics of the people tracker at different stages of its lifetime. The size is given in lines of code (LOC), not counting empty or comment lines.

Code size. The code size as well as the number of classes and methods have varied considerably. Up to the first stage within Phase 3, functionality was simply added without removing much unused functionality. Nevertheless, very little code was added during Phase 2 when the people tracker was changed to interoperate with the Reading Vehicle Tracker.

When the re-engineering stage within Phase 3 started, there was also little change in code size. The explanation the developers gave is that at this stage, unused functionality was identified and marked as such. However, until the end of Phase 3, while new functionality for ADVISOR was added, most of the unused legacy functionality was kept in case it would be needed again. When it was finally removed in the last version examined here, one can see the sharp decrease in code size.

Global functions. The original code contained a large number of global functions. This number was substantially reduced during the re-engineering stage and again when most of the unused functionality was removed.

From our experience of Phase 3, we have inferred several connections between the process changes described earlier and the product characteristics described above.

- The new software process, being closer to Stereotype 2, includes a new design strategy, which observes more dependencies and also considers possible future uses of the software. One of its effects is the reduction of global functions to a minimum.
- One result of the re-engineering step in Phase 3 is that the size of the code was reduced and the proportion of used functionality increased. This can be seen in the percentage of instantiated classes given in Table I. The reduction in redundancy directly improves software maintainability.
- The recovered software artefacts and the redesigned code structure make it easier to understand and change the software. The current maintainers have already noticed the change as they compared the way in which students approached the unknown code when they started to work on the project, before and after the re-engineering process.

5. LESSONS LEARNED

The proposed distinction between latitudinal and longitudinal process diversity is intended to help software managers to analyse process diversity by providing a conceptual framework. Using this framework, corrective or preventive measures can be related to a particular type of process diversity. In this sense, our classification modularizes process diversity.

In process assessment and improvement, e.g. using G/Q/M, it is of great importance to have a detailed analysis of the process. Such knowledge decreases the semantic gap between the high-level G/Q/M goals and the questions which need to be asked in building the 'G/Q/M tree' (which is a directed acyclic graph). Process improvement may be done to control the negative aspects of process diversity. In this case, our classification framework can help in G/Q/M analysis by relating a G/Q/M goal to an improvement action.

A process is executed within a process environment. The environment should provide mechanisms to address the issues related to process diversity. Our classification can provide additional insight into such mechanisms.

In the following, we outline some of the actions which can be taken to counteract the negative impacts of each type of process diversity.

5.1. Latitudinal process diversity

Whenever partners interact, it is important to build a good *co-operative atmosphere* where partners can work together as a 'virtual team', joined by common goals [26]. Participating teams should spend time together at the beginning of a project to understand each other's processes, vocabulary, tools and perspective. For example, one development team could participate in seminars or workshops sponsored by the other team. This will also help to break down barriers stemming from cultural differences, creating a better working environment.

Interface issues should be identified and resolved at the beginning of a project rather than reactively as problems arise. Anderson *et al.* have proposed a similar idea for handling usability aspects at the start of a project [27].

Project partners should be flexible about the detailed definition of processes and should try to tolerate minor differences. For example, one partner's processes cannot usually be imposed on another partner without creating problems. One way to achieve the necessary flexibility could be a *periodic review process for processes*, which should include a method for reaching consensus. This should be compounded by a constant high level of communication, as recommended by Herbsleb and Grinter [12].

5.2. Longitudinal process diversity

When developing software, one should be aware that in the future, it might be re-used in different projects and contexts. In order to minimize problems stemming from longitudinal process diversity we recommend to keep the following points in mind.

- *Scalability*. This mostly affects the software design process. Aim for a good object-oriented structure, avoiding global functions.

- *Portability*. Use of ISO and other standards and minimize the use of non-portable software and hardware functions.
- *Generality*. Use as few prior assumptions on input and output data as possible, and use abstract concepts during design (e.g. when programming a people tracker, find a layer of abstraction for tracking methods—the system might be used to track animals in the future).
- *Interoperability*. Use open standards (e.g. XML) for data exchange. This reduces platform dependencies for data, documents, and configuration information.
- *Maintainability*. Probably the most important point, and connected to several of the above. We recommend the following measures to attain a high level of maintainability:

  - carefully choose, document, and follow a suitable software process to achieve a high *software process consistency over time*;
  - create all software artefacts (e.g. programmer's and user manuals) and keep them synchronized with the implementation;
  - do refactoring [28] 'on the fly'—this can involve incremental reverse engineering using design patterns [22];
  - use metrics periodically to monitor design quality.

6. CONCLUSIONS AND FUTURE WORK

We have presented a way to classify process diversity into latitudinal and longitudinal categories. By studying the case of a project exhibiting both types of process diversity, we have shown how the effects on the quality of products and processes differed with the type of process diversity encountered.

The effects of longitudinal process diversity are most relevant where software components or products are re-used within significantly different processes. This implies that software engineers should try to anticipate the *possibility* of component reuse within different processes, even when the detailed *form* of reuse cannot be predicted.

The effects of latitudinal process diversity are likely to be felt most strongly where 'virtual teams' are created for a specific project, especially if the participants are drawn from contrasting traditions of software development processes.

Understanding the differences between these kinds of process diversity will help software engineers and project managers to assess their implications for process and product quality, and their impact on risk management in projects. Furthermore, relating process diversity to broader concepts of software evolution provides a linkage between fine-grained process improvements and larger-scale organizational models of the co-evolution of software and business processes.

Our current work includes the collection of more data from the early stages of the project. We are working with the project leader of Phase 1 to acquire copies of its source code. With this additional data, we would like to analyse the trends of software metrics further back into Phase 1, thereby covering the whole lifetime of the software.

The classification into latitudinal and longitudinal process diversity has been illustrated with a project strongly exhibiting both types. It would be interesting to see how these two types of process diversity differ in other projects. Further study into projects with only one type of process diversity and projects featuring both types within their lifetime might help to answer this question.

ACKNOWLEDGEMENTS

The authors wish to thank the original developer of the people tracker, Dr Adam Baumberg, and the project leader at that time, Professor David Hogg from the University of Leeds, for their kind support in providing information about Phase 1 of the project. The authors also gratefully acknowledge the comments and suggestions of the anonymous reviewers, which have helped to improve the article.

REFERENCES

1. Doppke JC, Heimbigner D, Wolf AL. Software process modeling and execution within virtual environments. *ACM Transactions on Software Engineering and Methodology* 1998; **7**(1):1–40.
2. Dion R. Process improvement and the corporate balance sheet. *IEEE Software* 1993; **10**(4):28–35.
3. Humphrey W. Introduction to software process improvement. *Technical Report CMU/SEI-94-TR-007*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, August 1992. (Revised version, August 1993.)
4. Herbsleb J, Carleton A, Rozum J, Siegel J, Zubro D. Benefits of CMM-based software process improvement: Initial results. *Technical Report CMU/SEI-94-TR-013*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, August 1994.
5. Cattaneo F, Fuggetta A, Lavazza L. An experience in process assessment. *Proceedings of the 17th International Conference on Software Engineering (ICSE-17)*. ACM Press: New York NY, 1995; 115–121.
6. Sharp H, Woodman M, Hovenden F, Robinson H. The role of 'culture' in successful software process improvement. *Proceedings 25th Euromicro Conference (EUROMICRO '99)*, vol. 2. IEEE Computer Society Press: Los Alamitos CA, 1999; 2170–2176.
7. Paulk MC. Models and standards for software process assessment and improvement. *Software Process Improvement*, ch. 1, Hunter RB, Thayer RH (eds). IEEE Press: Piscataway NJ, 2001; 1–36.
8. van Solingen R, Berghout E. *The Goal/Question/Metric Method*. McGraw-Hill: London, 1999; 200 pp.
9. Lehman MM. The programming process. *IBM Research Report RC 2722*, IBM Research Center, Yorktown Heights, U.S.A., 1969.
10. Lehman MM. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE* 1980; **68**(9):1060–1076.
11. Lehman MM, Perry DE, Ramil JF, Turski WM, Wernick P. Metrics and laws of software evolution—the nineties view. *Proceedings of the Fourth International Conference on Software Metrics (Metrics 97)*. IEEE Press: Los Alamitos CA, 1997; 20–32.
12. Herbsleb JD, Grinter RE. Splitting the organization and integrating the code: Conway's law revisited. *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*. ACM Press: New York NY, 1999; 85–95.
13. IEEE Computer Society. *Recommended Practice for Architectural Description of Software Intensive Systems. IEEE Std-1471-2000*. IEEE Computer Society Press: Los Alamitos CA, 2000.
14. Carmel E. *Global Software Teams: Collaborating Across Borders and Time Zones* (1st edn). Prentice-Hall: Englewood Cliffs NJ, 1999; 200 pp.
15. Chatters BW, Lehman MM, Ramil JF, Wernick P. Modelling a software evolution process: A long-term case study. *Software Process Improvement and Practice* 2000; **5**(2/3):91–102.
16. Beck K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley: Reading MA, 1999; 224 pp.
17. Parnas DL. Software aging. *Proceedings 16th International Conference on Software Engineering (ICSE-16)*. IEEE Computer Society Press: Los Alamitos CA, 1994; 279–287.
18. Warboys BC, Greenwood RM, Kawalek P. Modelling the co-evolution of business processes and IT systems. *Systems Engineering for Business Process Change: Collected Papers from the EPSRC Research Programme*, Henderson P (ed.). Springer: London, 2000; 10–23.
19. Weide BW, Heym WD, Hollingsworth JE. Reverse engineering of legacy code exposed. *Proceedings of the 17th International Conference on Software Engineering (ICSE-17)*. ACM Press: New York NY, 1995; 327–331.
20. Baumberg AM. Learning deformable models for tracking human motion. *PhD Thesis*, School of Computer Studies, University of Leeds: Leeds, U.K., October 1995; xiv+138 pp. ftp://ftp.comp.leeds.ac.uk/comp/doc/theses/baumberg.ps.gz.
21. Remagnino P, Baumberg A, Grove T, Tan T, Hogg D, Baker K, Worrall A. An integrated traffic and pedestrian model-based vision system. *Proceedings of the 8th British Machine Vision Conference (BMVC97)*, Clark A (ed.). BMVA Press: Malvern, UK, 1997; 380–389.
22. Fowler M, Beck K, Brant J, Opdyke W, Roberts D. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley: Reading MA, 1999; 431 pp.
23. Paulk MC, Curtis B, Chrissis MB, Weber CV. Capability maturity model, version 1.1. *IEEE Software* 1993; **10**(4):18–27.

24. Mitchell-Jones AJ. Personal communication, February 2002.
25. International Organization for Standardization. ISO: Programming Languages—C. ISO/IEC 9899:1999(E), 1999.
26. Wells M, Harrison R. The liminal moment. Understanding distributed communication and business processes. *Proceedings of the Conference on Empirical Assessment in Software Engineering (EASE 2001)*, 2001.
27. Anderson J, Fleek F, Garrity K, Drake F. Integrating usability techniques into software development. *IEEE Software* 2001; **18**(1):46–53.
28. Demeyer S, Ducasse S, Nierstrasz O. Finding refactorings via change metrics. *Proceedings of the 2000 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2000)*. ACM Press: New York NY, 2000; 166–177.

AUTHORS' BIOGRAPHIES

Nils T Siebel received his degree in Mathematics from the University of Bremen, Germany. He is currently a Doctoral Candidate for Computer Science at The University of Reading, U.K., where he is researching the computer vision and software integration issues within the people tracking subsystem presented here. In addition to software processes and software maintainability, his research interests are control theory, computer vision and robotics.

Stephen Cook received a BA (Hons.) in Social Science and an MSc in Computing from the Middlesex Polytechnic and the University of Wales, respectively. After more than 10 years of industrial experience as a Senior Scientific Officer and Principal Analyst/Programmer he is now a Research Fellow at The University of Reading, U.K., where he is carrying out research in the areas of software processes and evolution, software architectures and software design.

Manoranjan Satpathy received his undergraduate, postgraduate and PhD degrees, all in Computer Science, from the Indian Institute of Science in Bangalore, the Indian Institute of Technology in Kanpur and the Indian Institute of Technology in Bombay, respectively. Currently he is working as a Lecturer in the Computer Science Department of The University of Reading, U.K. His research interests are software engineering, programming languages and formal methods.

Daniel Rodríguez has a degree in Computer Science from the University of the Basque Country, Spain. Currently, he is a Lecturer and Doctoral Candidate at The University of Reading, U.K. His principal research interest is empirical software engineering, including software assessment and improvement, data mining and the application of evolutionary computation to decision making tasks.