# On the Use of Handover Checkpoints to Manage the Global Software Development Process

Frank Salger

Capgemini sd&m, Carl-Wery-Straße 42, 81739 Munich, Germany
frank.salger@capgemini-sdm.com

**Abstract.** Despite the fact that global software development (GSD) is steadily becoming the standard engineering mode in the software industry, commercial projects still struggle with how to effectively manage it. Recent research and our own experiences from numerous GSD projects at Capgemini sd&m indicate that staging the development process with handover checkpoints is a promising practice in order to tackle many of the encountered problems in practice. In this paper we discuss typical management problems in GSD. We describe how handover checkpoints are used at Capgemini sd&m to control and safely manage large GSD projects. We show how these handover checkpoints and the use of cohesive and self-contained work packages effectively mitigate the discussed management problems. We are continuously refining and improving our handover checkpoint approach by applying it within large scale commercial GSD projects. We thus believe that the presented results can serve the practitioner as a fundament for implementing and customizing handover checkpoints within his own organisation.

## 1 Introduction

It is well known that the global distribution of software development processes poses new ones and intensifies existing challenges to project management as compared to collocated software development [1-5].

A number of practices have been proposed in order to tackle these problems. One of the most promising one seems to be the use of clearly defied synchronization and handover points [1, 4-6]. But although much research emphasis the relevance of such handover checkpoints for GSD, the description of these checkpoints is often not elaborated enough to serve as basis for integrating them into commercial GSD projects.

The main contribution of this paper is a description of the structure and content of the handover checkpoints used at Capgemini sd&m for managing globally distributed, large scale custom software development projects of business information systems. We provide detailed descriptions on our notion of 'work package' as well as on the structure and content of our handover checkpoints. Further, we show how the handover checkpoints complement the 'quality gates', already used at Capgemini sd&m. Finally, we explain how these results can serve as a basis for practitioners to integrate handover checkpoints in their own organizations.

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 267–276, 2009.

The notion for ‘work package’ and the definition of handover checkpoints given in this paper where developed at Capgemini sd&m, and are specifically tailored to GSD of large business information systems. Capgemini sd&m is the Technology Services unit of the Capgemini Group in Germany and Switzerland and offers its customers end-to-end process and software solutions that have a crucial impact on their competitiveness. The Capgemini Group exploits captive centers for offshoring software development in several countries.

The rest of the paper is organized as follows. In section 2, we discuss the most prevalent problems we encountered when managing GSD project for large business information systems at Capgemini sd&m. In section 3, we introduce and discuss our notion of ‘work package’ and the structure and content of the handover checkpoints. The benefits we obtained by using handover checkpoints in our GSD projects are described in section 4. We draw our conclusions in section 5 and sketch our agenda for future research.

## 2 Prevalent Problems in GSD

In this section, we discuss some of the most prevalent problems we encountered in numerous globally distributed projects for the development of large scale and custom build business information systems. Doing so, we point out related work, thereby affirming our experiences as well as the cited research results.

**1. Too Little Process Structure**
The distributed process is often not structured enough in order to be effectively manageable. Work units (in particular software requirements specifications) are not defined precisely enough [2, 6, 7]. Dependencies between different work units as well as dependencies between different tasks are often little understood [8]. There are no defined points, where work units and the responsibility therefore are handed too and fro between distributed teams.

*Resulting Problems:* Without intermediate milestones or checkpoints, the overall process becomes fragile. Especially in GSD, the distributed teams must be able to relay on stable and complete baselines from which they can safely start their work.

**2. Missing Concept to Bundle Work Units**
The distribution of the software engineering process negatively affects the ability to coordinate and control work [1-6, 8]. It is thus important to bundle work units which can be handled together without further need to communicate too much about them. But defining work packages only form a ‘module point of view’ seems to be insufficient: It is important to define a work package not only from a logical viewpoint but also from a management and risk oriented viewpoint.

*Resulting problems:* Cohesiveness from a logical point of view is not the only important dimension for devising modules. If for example, modules are chosen too small, project management will eventually be screwed up in micro-management and frequent re-planning due to inevitable changes which happen frequently on the fine grained task levels. If the on the other hand work packages are devised too big from a risk management point of view, then problems with such work packages will surface

too late and only until they already have a critical impact on the overall schedule and effort. Finally, if work packages are chosen to be too big from a business logic point of view, they will bundle lots of non-cohesive business logic. This will force the development team of such work packages to know too much about the overall business logic. This leads to a communication overkill, which in turn results in lots of effort and time needed to develop such work packages.

**3. Undefined Acceptance Criteria and Quality Objectives**
Often, acceptance criteria and quality objectives for work units are not defined precisely enough, or are missing altogether. This holds for requirements specifications, system specifications or source code alike.

*Resulting problems:* The fundamental problem with weak acceptance criteria is that it is not clear, when a deliverable is really finished. This makes meaningful earned value analysis impossible, which in turn prohibits controlling of real progress. But rigorously monitoring progress is especially important in GSD [2, 6].

**4. Missing Alignment of Engineering Disciplines**
GSD poses additional challenges to all software engineering disciplines. In such stormy environments, it is natural that the managers, business analysts and software architects concentrate on getting their own job done. This however proves to be especially dangerous in GSD, where tight cooperation and awareness between different roles is paramount [9].

*Resulting problems:* Divergence of disciplines can happen in various ways. Probably the most common one is, when actual effort needed for the implementation of work units is not reported to the management in a disciplined manner. It then becomes impossible to readjust the plan and adapt the projections for overall effort and schedule. Moreover, systematic estimation errors are not detected.

**5. Insufficient Knowledge Transfer**
It is well known that global distribution usually necessitates a more intensive knowledge transfer as compared in collocated development settings [2, 6]. It is simply not possible to walk into the colleagues cube and to ask him for the missing information. Cultural differences might further add to this problem, when colleagues find it impolite to ask clarifying questions.

*Resulting problems:* Onsite and offshore software engineers often have little or none project experience in common. Thus, they have less common ground with regards to processes and practices. Intensive knowledge transfer and training is paramount to tackle this issue.

## 3 The Use of Work Packages and Handover Checkpoints in GSD

In this section, we first describe our notion of ‘work package’. We then present the structure of our handover checkpoints and show, how they are applied in GSD.

### 3.1 Work Packages

Two organizations that work together but are geographically separated require a clear concept for bundling and executing work. Often, 'modules' (in the sense of Parnas [10]) are used as 'work packages'. As the organizational structure and the product structure influence each other, modules have to be very carefully devised in GSD [8, 11, 12, 13]. At Capgemini sd&m, we defined 'work packages' to consist of three kinds of artifacts:

1. *Software requirement specification (SRS) artifacts.* A cohesive set of system use cases, user interface descriptions, domain objects and other requirement artifacts. We call such sets 'conceptual components' [14]. Additional specification-related information is bundled into a work package: a) 'Reuse lists', which contain information about SRS artifacts that can be reused and have already been realized during development of other work packages (like certain business rules). b) 'Tracing matrices' which are used to trace the SRS artifacts back to the high level requirements. c) Specifications of functional test cases.
2. *Design artifacts.* Usually, a subsystem comprising a couple of modules will realize the conceptual component of a work package. Included into a work package will be the external interfaces of the modules (i.e., the external technical view on the subsystem which realizes the conceptual component) and the internal high level design of the modules, like e.g. which layers will be used.
3. *Project management artifacts.* The engineering artifacts as mentioned under 1) and 2) are complemented by bundling management-oriented artifacts into a work package: These are the list of work units which are bundled into the work package, the schedule for the work package, and the definition of quality objectives and acceptance criteria for the work units.

This notion of work package allow us to define

- clear artefacts to be delivered by both parties
- clear handover points and handover processes
- clear acceptance criteria for work units of both parties

Some further comments on the notion of work package might be needed: It might seem that a work package is bundled by the onsite team and then 'thrown over the wall' to the offshore development team for implementation. This is definitively not the case.

In contrary, key offshore roles are involved in the bundling work packages throughout the project live-cycle: First, the offshore business analysts already join the onsite team early on from requirements engineering, and are thus involved in structuring the business domain and 'cutting' it down into sub-domains. These sub-domains usually become the first candidates for conceptual components.

During the inception and elaboration phase, the onsite and offshore architects together devise the high level architecture, where the conceptual components are further refined into modules. This usually happens incrementally and in some cases might involve the redefinition of some conceptual components.

Step by step, reasonably stable components (modulo requirements change) are defined in that way, i.e., the content of the work packages gets fixed. For these fixed

work packages, the onsite and the offshore project manager then jointly create the precise list of work units, devise a work package schedule and allocate the necessary developers to the work package. They also break the overall quality objectives down to the concrete work units.

### 3.2 Handover Checkpoints

Before a work package is cleared for realization, the 'Work Package Handover Checkpoint' (WPHC) is applied jointly by onsite and the offshore team members. After the construction of a work package, the result is checked in the 'Result Handover Checkpoint' (RHC). Please note that in between these two checkpoints, comprehensive code quality assurance is conducted, like continuous integrations, systematic code reviews, and automatic checks for architecture compliance.

One important point to observe is that not the work products themselves are handed over (in the sense of 'sending a parcel'). They reside in shared repositories anyway. What actually is handed over, is the *responsibility* for certain activities and tasks conducted on work packages.

A positive outcome of the WPHC means that the onsite and offshore team members agree on the 'transfer' of the work package: The offshore developers basically declare that the work package contains all information they need in order to construct the work package.

Analogously, a positive outcome of the RHC means that the team members jointly agree on transferring the responsibility for the result of the work package construction to the integration team.

Figure 1 sketches, how the two checkpoints are used in GSD. The thick arrows indicate a transfer of responsibility. In this example, the sequential realization of two work packages is shown.

Usually, multiple work packages will be constructed in a rather overlapping or concurrent fashion. In the example, the integration of the work package results is done onsite as indicated by the assembled pieces of the puzzle top right in the figure.

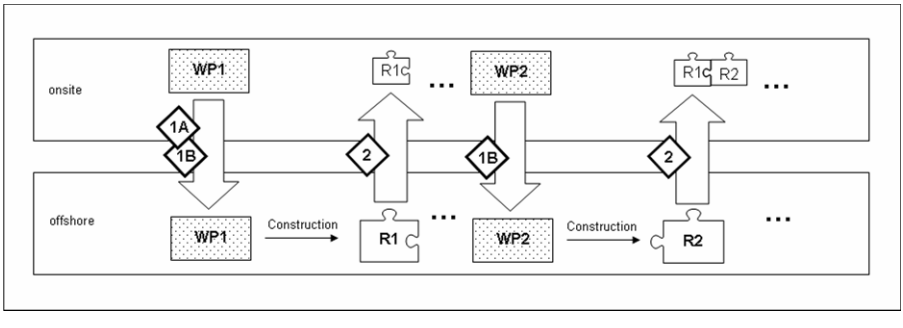


**Fig. 1.** Application of handover checkpoints (simplified example)

**Checkpoint 1: Work Package Handover Checkpoint (WPHC)**
The WPHC encompasses two kinds of evaluations: 1A) Initial pre-construction check and 1B) work package check. We now describe these two checks in more detail.

*1A) Initial pre-construction check:*

- Overall goal: Ensure that the 'production means' are in place and mature. This check is not applied to single work packages, but to the 'production means' which will be used to construct work packages.
- Sub-goals: a) Determine that the high-level requirements and the basic specification artefacts are precisely defined and a baseline is established. b) Ensure that the high-level architecture and development means are in place and mature. c) Ensure that the necessary processes are defined and agreed on.

*1B) Work package check:*

- Overall goal: Ensure that construction of a specific work package can safely start. This check is applied to each single work package.
- Sub-goals: a) Ensure that the content which will be constructed within this work package is clearly defined. b) Determine that the internal design of the modules in the work package is defined. c) Ensure that the realization of the work package is planned, scheduled and staffed.

**Checkpoint 2: Result Handover Checkpoint (RHC)**
The RHC encompasses only one kind of evaluation.

- Overall goal: Determine whether the result of work package construction is finished and completed according to the defined quality objectives.
- Sub-goals: a) Ensure that the result complies with the specification, the design and the architecture. b) Ensure that the result is complete and does comply with the quality objectives. c) Determine whether the estimation for the work package correct was correct.

We refined the sub-goals of the checkpoints into concrete checklist questions. These questions are augmented with concrete benchmarks derived from our completed GSD projects. These checklists are the central tool when we apply our handover checkpoints.

Using handover checkpoints fits well to the use of agile methods: The onsite and offshore team members need to synchronize on the handover checkpoints but can use their own processes in between the checkpoints. This supports fast cooperation ramp up, as not team has to learn the processes of the other team in every detail [15]. Obviously this can be especially supportive when working with sub-contractors (e.g. in an offshore outsourcing mode).

When using Scrum for example, the scope of a 'sprint' resembles to the scope of a work package. In the 'sprint meeting', the precise list of work units which are bundled by the work package and the schedule for the construction of the work package would be defined. The sprint meeting would be finished by the application of the WPHC. In turn, the RHC could be applied during the 'retrospective meeting' after completion of a sprint. Applying agile methods in GSD is however a challenging field and requires further research [16].

We believe that the discussed overall goals and the sub-goals of the checkpoints will look very similar even for different companies engaged in offshore custom software development. The concrete questions and benchmarks however might differ:

This is the tailoring point, where a practitioner could derive his own questions from the sub-goals according to the needs of his company.

At Capgemini sd&m, we already use so called 'quality gates' in order to assess the content of software requirements specifications [14, 18] and software architectures [19]. These quality gates however have the main goal to find major problems which put project success at risk, like for example the ignorance of non-functional requirements or missing alignment of the architecture to the business and user requirements. As such, they concentrate on the 'higher-levels' of software architecture and software requirements specifications.

We apply the quality gates in GSD as well, but complement them by the finer grained checkpoints which concentrate on the specifics of GSD. This is also necessary due to the fact that in GSD, problems with single work packages get the attention of the management less easily than in collocated development settings: Awareness and insight into status is impeded by high distribution [9, 17].

## 4 Benefits from Using Work Packages Handover Checkpoints

We now describe the benefits we obtained by using work packages and handover checkpoints in our GSD projects by showing, how our approach addresses the problems discussed in section 2.

**1. Mitigation of problem *'Too little process structure'***
The handover checkpoints effectively stage the development process. They allow the transfer of responsibility in a defined manner. The comprehensive checklists used in the checkpoints ensure that the 'receiving' teams can safely start their work.

**2. Mitigation of problem *'Missing concept to bundle work units'***
The definition of 'work package' helps to bundle cohesive and self-contained sets of work units. This helps to regulate the communication between distributed teams, as it reduces the need for repeatedly inquiring missing information.

**3. Mitigation of the problem *'Undefined acceptance criteria and quality objectives'*.** In the WPHC, the use of precisely defined acceptance criteria and quality objectives is checked. This makes clear, when a single work unit or a complete work package is 'finished', and allows effective earned value analysis. In the RHC, compliance of the result to the defined criteria and objectives is checked.

**4. Mitigation of problem *'Missing alignment of engineering disciplines'***
In the RHC it is checked whether work packages results where constructed within time and budget, while meeting the defined quality objectives. It is also checked, how deviations to the plan are considered by the project management, and whether their impact on the overall schedule is estimated. This continuously re-aligns the engineering disciplines with the management disciplines.

**5. Mitigation of problem *'Insufficient knowledge transfer'***
In both handover checkpoints it is assessed, how knowledge transfer was carried out and maintained. The checkpoints also mandate the controlling of the knowledge transfer, i.e., the project must implement mechanisms to ensure the effectiveness of the knowledge transfer.

Taken together, the notions of 'handover checkpoints' and 'work packages' supports clear handovers of responsibilities based on precisely defined work packages, allow earned value analysis, align the engineering disciplines and ensure effective knowledge transfer in GSD.

## 5 Conclusion and Future Work

We discussed some of the most prevalent problems encountered in managing global software development (GSD). These were

- Too little process structure
- Missing concept to bundle work units
- Undefined acceptance criteria and quality objectives
- Missing alignment of engineering disciplines
- Insufficient knowledge transfer.

Based on this discussion, we motivated the need for handover checkpoints. We then presented an overview on the handover checkpoints used at Capgemini sd&m to safely manage GSD.

Finally, we reported on the benefits already gained at Capgemini sd&m from applying our notion of work packages and handover checkpoints. These are:

- Precisely defined, cohesive and self-contained sets of work units
- Clear handover of responsibilities
- Support for management discipline (e.g. by supporting earned value analysis)
- Alignment of the management with the engineering disciplines
- Effective knowledge transfer

We are continuously applying the notion of work packages and the handover checkpoints within our GSD projects very successfully.

In short, the use of work packages and handover checkpoints allow clear handovers of responsibilities based on precisely defined work packages. This in turn allows earned value analysis, aligns the engineering disciplines and ensures effective knowledge transfer for large GSD projects.

Further research is however necessary in order to economically optimize our approach:

- How can the application of handover checkpoints be supported by tools? Here, some promising results are already available (see e.g. [20]).
- How can we avoid isolating the distributed teams from each other if we further decrease communication by using strong modularization? How can we avoid, that the teams only concentrate on their work package while neglecting to consider the overall big picture of the system (see also [21, 22])?
- How do handover checkpoints fit into different process models like the Rational Unified Process [23] or Scrum [16]?
- How do handover checkpoints fit to other models of work distribution (see, e.g., [24]).

## References


1. Cusumano, M.A.: Managing Software Development in Globally Distributed Teams. Communications of the ACM 51(2), 15–17 (2008)
2. Sangwan, R., Bass, M., Mullik, N., Paulish, D., Kazmeier, J.: Global Software Development Handbook. Auerbach Publications (2006)
3. Battin, R.D., Crocker, R., Kreidler, J., Subramanian, K.: Leveraging Resources in Global Software Development. IEEE Software 18(2), 70–72 (2001)
4. Cusick, J., Prasad, A.: A Practical Management and Engineering Approach to Offshore Collaboration. IEEE Software 123(5), 20–29 (2006)
5. Kommeren, R., Parviainen, P.: Philips Experiences in Global Distributed Software Development. Empirical Software Engineering 2(6), 647–660 (2007)
6. Hussey, J.M., Hall, S.E.: Managing Global Development Risks. Auerbach Publications (2007)
7. Smite, D.: Requirements Management in Distributed Projects. Journal of Universal Knowledge Management, J.UKM 1(2), 69–76 (2006)
8. Herbsleb, J.D.: Global Software Engineering: The Future of Socio-technical Coordination. In: Proc. of Future of Software Engineering, pp. 188–198. IEEE Computer Society Press, Los Alamitos (2007)
9. Damian, D., Chisan, J., Allen, P., Corrie, B.: Awareness meets requirements management: awareness needs in global software development. In: Proc. of the International Workshop on Global Software Development, International Conference on Software Engineering, pp. 7–11 (2003)
10. Parnas, D.L.: On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM 15(12), 1053–1058 (1972)
11. Conway, M.: How Do Committees Invent? Datamation 14(4), 28–31 (1968)
12. Herbsleb, J.D., Grinter, R.E.: Architectures, Coordination, and Distance: Conway's Law and Beyond. IEEE Software 16(5), 63–70 (1999)
13. Herbsleb, J.D., Grinter, R.E.: Splitting the Organisation and Integrating the Code: Conway's Law Revisited. In: Proc. of the 21th International Conference on Software Engineering, pp. 85–95. IEEE Computer Society Press, Los Alamitos (1999)
14. Salger, F., Sauer, S., Engels, G.: An Integrated Quality Assurance Framework for Specifying Business Information Systems. In: Proc. of the Forum at the 21th International Conference on Advanced Information Systems, pp. 25–30. CEUR (2009)
15. Paasivaara, M., Lassenius, C.: Collaboration Practices in Global Inter-organizational Software Development Projects. Software Process Improvement and Practice 8, 183–199 (2004)
16. Hossain, E., Babar, M.A., Paik, H.: Using Scrum in Global Software Development: A Systematic Literatur Review. In: Proc. of the 4th International Conference on Global Software Engineering. IEEE Press, Los Alamitos (2009)
17. Herbsleb, J.D., Paulish, D.J., Bass, M.: Global Software Development at Siemens. Experience from Nine Projects. In: Proc. of the 27th International Conference on Software Engineering, pp. 524–533. IEEE Press, Los Alamitos (2005)
18. Salger, F., Engels, G., Hofmann, A.: Inspection Effectiveness for Different Quality Attributes of Software Requirement Specifications: An Industrial Case Study. In: Proc. of the 7th ICSE Workshop on Software Quality, pp. 15–21. IEEE Computer Society Press, Los Alamitos (2009)

19. Salger, F., Bennicke, M., Engels, G., Lewerentz, C.: Comprehensive Architecture Evaluation and Management in Large Software-Systems. In: Becker, S., Plasil, F., Reussner, R. (eds.) QoSA 2008. LNCS, vol. 5281, pp. 205–219. Springer, Heidelberg (2008)
20. Gupta, A.: The 24-Hours Knowledge Factory: Can it Replace the Graveyard Shift? Computer 42(1) (2009)
21. Cataldo, M., Bass, M., Herbsleb, J.D., Bass, L.: Managing Complexity in Collaborative Software Developoment: On the Limits of Modularity. In: Proc. of the Workshop on Supporting the Social Side of Large-Scale Software Development, Conference on Computer Supported Cooperative Work 2006. ACM, New York (2006)
22. Ebert, C., Neve, P.: Surviving Global Software Development. IEEE Software 18(2), 62–69 (2001)
23. Rational Unified Process. IBM Coporation. Version 7.0.1. 2000 (2007)
24. Carmel, E.: Global Software Teams. Prentice Hall PTR, Englewood Cliffs (1999)