Volume 79, issue 11, November 2006
ISSN 0164-1212
ELSEVIER
Special Issue:
Software Cybernetics
The Journal of
Systems and Software

Available online at www.sciencedirect.com

ScienceDirect

The Journal of Systems and Software 79 (2006) 1504–1513

The Journal of Systems and Software

www.elsevier.com/locate/jss

Requirement process establishment and improvement from the viewpoint of cybernetics ☆

Hong Xu [a], Pete Sawyer [b,*], Ian Sommerville [b]

[a] School of Computer Science and Engineering, BeiHang University, Beijing 100083, China
[b] Computing Department, Lancaster University, Lancaster, LA1 4WA, UK

Available online 16 June 2006

Abstract

As a branch of engineering cybernetics, automatic control theory has been extensively applied to improve products, increase productivity and rationalize management. This paper adapts the principles of automatic control theory to the field of software process improvement. In particular, the work described uses control theory to define a requirement engineering (RE) process control system, its dynamic and steady-state performance, and the steps in designing, analyzing and improving such a system. The work has highlighted the need for process activities relating to measuring elements, including those in feedback compensation and organizational support. The results of this research can be used to guide the establishment and improvement of RE processes, compare different requirement process solutions quantitatively, develop methods for evaluating benefits from process improvements, and structure the application of knowledge about RE.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Software cybernetics; Software process improvement; Requirements engineering; Control theory

1. Introduction

The Requirements Engineering Good Practice Guide (REGPG) (Sawyer et al., 1999; Sommerville and Sawyer, 1997) was the first public-domain software process improvement (SPI) model that explicitly focused on the incremental, systematic improvement of Requirements Engineering (RE) processes. It describes 66 RE good practices organized according to a number of RE process areas, an assessment method and a three-level improvement model.

Since the REGPG's publication in 1997 it has been subject to an extensive empirical analysis (Sommerville and Ransom, 2005; Kauppinen et al., 2002; Kauppinen and Kujala, 2001) designed to improve its efficacy as an RE process improvement model. It has also been the subject of, or contributed to, several research projects investigating RE process improvement for particular domains, such as packaged software development (Regnell et al., 1998), and organisational contexts, particularly SMEs (Nikula and Sajaniemi, 2002; Nikula, 2003).

These studies, complement the eight years of experience and deployment with the REGPG in many companies world-wide. They have shown that among the REGPG's most important strengths are that it provides practitioners with an overall view of RE concepts and principles, is effective at raising personnel awareness of RE and motivating them for RE process improvement, is useful in identifying process improvements across a range of different types of companies and includes relevant requirements practices for different kinds of application domains.

However, the REGPG offers only a limited and general set of process improvement guidelines. Moreover it lacks sufficient guidance for selecting a realistic set of practical

☆ This is an extended version of the paper of the same name presented at the 2nd International Workshop on Software Cybernetics (IWSC'05).
* Corresponding author. Tel.: +44 1524 10320; fax: +44 1524 10492.
E-mail addresses: xh@buaa.edu.cn (H. Xu), sawyer@comp.lancs.ac.uk (P. Sawyer), is@comp.lancs.ac.uk (I. Sommerville).

0164-1212/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2006.03.050

RE improvement actions to meet specific business goals with available resources.

We have exploited the understanding we have acquired through experience with, and validation of, the REGPG to investigate what insights into RE process improvement might accrue from the application of cybernetics. Our work represents the first attempt to apply classical control theory (Ogata, 1970) to RE process improvement in what we call a Requirements Process Control (RPC) system. We characterize RE problems as deviations from an RE process's ideal performance. The RPC system's role is to damp these deviations to a tolerable level thus making the RE processes more repeatable.

In Section 2 we map the REGPG's 66 good practices to different types of control system components and develop the mechanism of collecting process data for them in Section 3. We define the dynamic and steady-state performance of an RPC system in Section 4 and propose steps in designing and improving an RPC in Section 5. In Section 6 we demonstrate how the framework presented in this paper works in RE process establishment and improvement.

2. An RPC system

We use (I, O, C, T) to define an RPC system, where:

- I refers to the input of the RPC system i.e. the stakeholders' requirements to be implemented;
- O refers to the object to be controlled i.e. the requirement specification of the application. "Requirements specification" here is a proxy not only for the requirements specification document(s) but for the set of requirements artifacts that have to be managed by an RE process;
- C refers to the output of the RPC system i.e. the quality of O produced, and;
- T refers to the task of this RPC system i.e. controlling O to be produced within a given time and budget representing I and of the specified quality.

Fig. 1 shows the logical structure of an RPC system, and its basic components. These are:

- Actuators, which here are the requirements analysts who fulfill the control task T, representing the idea that different control strategies are embodied by the people performing process activities.
- Measuring elements, which measure C, and may also detect an error by comparing C with I.
- The requirements specification O.
- Compensators, which may act as actuators or measuring elements but in the process of application system requirement development to help improve the performance of the RPC system.

From these basic components we have classified the 66 good practices in the REGPG according to which components they address. Table 1 shows how the RE process areas map to the RPC components. Appendix A adds more

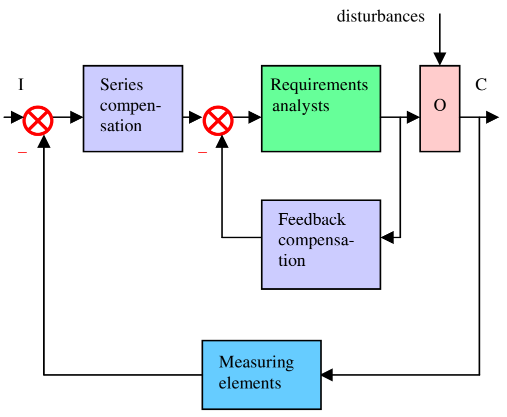

Fig. 1. Block diagram of an RPC system.

Table 1
The 66 key practice areas viewed from the perspective of different RPC system components

RPC system components	RE process area	Good practice[a]
Actuators	Feasibility study (AC4)	4.1
	Requirements elicitation (AC4)	4.3, 4.5–4.7, 4.9, 4.11 and 4.12
	Requirements description (AC6) (System modeling)	6 Describe requirements 7.1–7.3
	Requirements management (AC9)	9 Manage requirements
Series compensation	Feasibility study (SC4)	10.3
	Requirements elicitation (SC4)	4.10, 4.13 and 10.4
	Requirements management (SC9)	9.5
Measuring elements	Requirements validation (ME8)	8.1, 8.2, 8.5–8.7
Feedback compensation	Requirements elicitation (FC4)	4.10
	Requirements analysis (and negotiation) (FC5)	5.1, 5.2, 5.4–5.8, and 10.5
	Requirements description (FC6) (System modeling)	7.1, 7.2, 7.6 and 10.6
	Requirements validation (FC8)	8.3, 10.2, 8.8 and 10.5
Organizational support	Requirements documentation (OS3)	3.1–3.8
	Requirements elicitation (OS4)	4.2, 4.4, 4.8–4.11, and 4.13
	Requirements analysis and negotiation (OS5)	5.2, 5.5–5.8
	Requirements description (OS6) (System modeling)	6.1–6.5, 7.1–7.6, and 10.6
	Requirements validation (OS8)	8.2, 8.4, 10.1, 8.5, 8.7 and 8.8
	Requirements management (OS9)	9.1–9.9
	To all	10.3, 10.7–10.9

[a] The number reflects the section in a chapter in Sommerville and Sawyer (1997) describing the practice.

detail, listing the corresponding good practices that rather than just their reference numbers as shown in the last column of Table 1. It should be noted that

- An RPC system consists of different system components, which comprise different process activities. However, there is a many-to-many relationship between RPC components and process areas.
- Compensators acting as series compensation can be viewed as actuators.
- Feedback compensation relating to the requirements validation area can be viewed as measuring elements.
- Through the establishment of standards and policies, process institutionalization and staff training programs, an organization provides the culture, context and resources for an RPC system.

3. Mechanism of process data collection for RPC system components

Based on the work of (Kitchenham et al., 2001), we use a unified data set model {PA, AU, MP, ST} to measure the process and document process activities in an RPC system so as to ensure that the data collected are reliable, consistent, repeatable, comparable and interchangeable, and that the resulting analysis is valid. Here

- The object to be measured PA = {P_{er}, A_{ct}, P_{rj}}, where P_{er} is the person participating in the development of the requirements specification; A_{ct} = {C_{yc}, P_{ha}, T_{as}, G_{ra}}, a process activity performed by P_{er}, and is defined by the current development cycle C_{yc} of the current project P_{rj}, the current development phase P_{ha} in the software process followed, the project task T_{as} undertaken and the granularity G_{ra} of the resulting product such as modules or sub-systems; and P_{rj} = {N_{am}, P_{de}, D_{om}, T_{yp}, P_{op}}, where N_{am} is the project name, P_{de} the development platform, D_{om} the application domain, T_{yp} the application type (such as interactive or embedded systems), and P_{op} the operation platform.
- The basic set of measurement values AU = {AU$_1$, AU$_2$, AU$_3$}

  $$AU_i = \{A_i, U_i\}, \quad A_i = \{A_{iA}, A_{iE}\} \quad i = 1, 2, 3$$

  where A_1, A_2 and A_3 are three basic attributes to be measured on PA; A_1 consists of the actual time A_{1A} spent on A_{ct} and the estimated one A_{1E}, which are measured by minutes specified by U_1; A_2 consists of the actual size A_{2A} of the application system requirement specification produced by A_{ct} and the estimated one A_{2E}, which are measured by pages specified by U_2; A_3 consists of the actual number of defects A_{3A} found in the application system requirement specification produced by A_{ct} and the estimated one A_{3E}, which are measured by numbers specified by U_3; and A_{1E}, A_{2E} and A_{3E} are represented as a most likely value together with upper and lower bounds such as 95% or 99% confidence limits.
- The set of measurement protocols

  $$\begin{aligned}&\mathrm{MP} = \{\mathrm{MP}_1, \mathrm{MP}_2, \mathrm{MP}_3\}\\ &\mathrm{MP}_i = \{\mathrm{MP}_{iA}, \mathrm{MP}_{iE}\}\\ &\mathrm{MP}_{iA} = \{T_{iA}, P_{iA}, M_{iA}\}, \quad \mathrm{MP}_{iE} = \{T_{iE}, P_{iE}, M_{iE}\}\\ &\quad i = 1, 2, 3\end{aligned}$$

  which are defined in the context of PA, and where T_{iA} and T_{iE} specify when to measure A_i in the development process, P_{iA} and P_{iE} who are responsible for the measurement, and M_{iA} and M_{iE} the tools or methods used to extract, record and store the data values such as those described in the Team Software Process (TSP) (Humphrey, 2000). See Table 2 for details.
- The set of scale type specifications ST = {ST$_1$, ST$_2$, ST$_3$}, where in C_{yc} of P_{rj}, $0 < \mathrm{ST}_1 \leqslant$ the actual or estimated time of P_{ha} as to A_{1A} or A_{1E}, $0 < \mathrm{ST}_2 \leqslant$ the actual or estimated size of the application system requirement specification as to A_{2A} or A_{2E}, $0 < \mathrm{ST}_3 \leqslant$ the found or estimated number of defects in the application system requirement specification as to A_{3A} or A_{3E}.

It should be noticed that

- There can be more than one PA in a project, and one P_{er} may perform one A_{ct} or more.

Table 2
The measurement protocol definitions relating to the basic attributes to be measured

A_i		MP$_i$	Definition
A_1	A_{1A}	T_{1A}	T_{1A} = {T_{1AS}, T_{1AF}}, where T_{1AS} and T_{1AF} are the time at which A_{ct} starts and ends
		P_{1A}	P_{er}
		M_{1A}	Recommend those in TSP, and $A_{1A} = T_{1AF} - T_{1AS}$
	A_{1E}	T_{1E}	The time when the planning phase of P_{rj} ends
		P_{1E}	P_{er}, or the person who is responsible for project planning
		M_{1E}	Recommend those in TSP, and $T_{1EF} = T_{1ES} + A_{1E}$, where T_{1ES} is the planned time at which A_{ct} will start, and T_{1EF} the estimated time at which A_{ct} will end
A_2	A_{2A}	T_{2A}	$T_{2A} = T_{1AF}$
		P_{2A}	P_{er}, or the person who is responsible for measuring the size of a product
		M_{2A}	Recommend those in TSP
	A_{2E}	T_{2E}	The time when the planning phase of P_{rj} ends
		P_{2E}	P_{er}, or the person who is responsible for project planning
		M_{2E}	Recommend those in TSP
A_3	A_{3A}	T_{3A}	$T_{3A} \in [T_{3AR}, T_{3AO}]$, where T_{3AR} is the time when requirement inspection ends, T_{3AO} the time at which the number of defects is no longer counted after the system delivery
		P_{3A}	P_{er}
		M_{3A}	Recommend those in TSP
	A_{3E}	T_{3E}	The time when the planning phase of P_{rj} ends
		P_{3E}	P_{er}, or the person who is responsible for project planning
		M_{3E}	Recommend those in TSP

- The granularity of A_{ct} should be determined by the business goals of the software organization within its specific development context, the specific factors for the project, and the feasibility of collecting the process data needed.
- One RPC system manages and controls the development of one requirements specification.
- One component of an RPC system may be one A_{ct} or a composition of A_{ct}s, depending on the type of analysis that is required.

4. Definition of dynamic performance and steady-state error of an RPC system

Assume that there are r_1 ($r_1 \geqslant 1$) A_{ct}s acting as actuators, series compensators and feedback compensators (but excepting those in the requirements validation area), and r_2 ($r_2 \geqslant 0$) A_{ct}s acting as measuring elements including the feedback compensators in the requirements validation area.

Assume that the unit-step response $c(t)$ of an RPC system is represented by the curve shown in Fig. 2 in the context of the software development process. Here, the x-axis t represents the working days of P_{rj}, the y-axis the accumulated progress of P_{rj} measured or estimated by some technique, and the origin the starting point of P_{rj} at which the first process activity A_{ct1} starts. In other words, $T_{1S1} = 0$ and the progress of P_{rj} is 0. We define the accumulated progress of P_{rj} as 1 when the last process activity $A_{ct(r_1+r_2)}$ ends at $T_{1F(r_1+r_2)}$. We then have dynamic performance indicators of the RPC system:

- Delay time t_d, where $c(t_d) = 0.5$,
- Rise time t_r, where $t_r = t_{2r} - t_{1r}$, $c(t_{2r}) = 0.9$, $c(t_{1r}) = 0.1$, and
- Settling time t_s, where $t_s = T_{1F(r_1+r_2)}$; or time constant T, where $c(T) = 0.632$ if $\log|c(t) - c(t_s)|$ takes the form of a line (Ogata, 1970).

The actual dynamic performance of the RPC system can be represented by actual t_d, t_r and t_s, or T calculated by using A_{1Ai}, T_{1ASi} and T_{1AFi}, and the estimated dynamic performance by estimated t_d, t_r and t_s, or T by using A_{1Ei}, T_{1ESi} and T_{1EFi}, where $i = 1, \ldots, r_1 + r_2$.

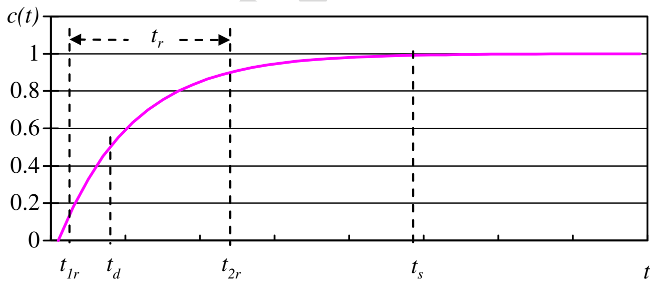

Fig. 2. The accumulated progress of P_{rj} controlled by an RPC system.

Under the assumption that the actual defects found are all fixed, the actual steady-state error e_{ssA} of an RPC system is defined as $\sum_{i=1}^{r_1}(A_{3Ai}|_{t=T_{3A}} - A_{3Ai}|_{t=T_{3AR}})$, which is also the actual measuring error e_{mA} of the system. We define the estimated steady-state error e_{ssE}, and the estimated measuring error e_{mE}, as $a \times T_{1EF(r_1)} \times (1 - b)$, where at $T_{1EF(r_1)}$ all the r_1 process activities will end, a is the estimated number of defects to be injected in the application system requirement specification per hour, and b is the estimated percent of defects to be fixed (Humphrey, 2000).

5. Steps of designing an RPC system

In general, we assume the performance specification of an RPC system is expressed as the cost constraint C_G, the given time T_G and the required quality Q_G of the requirements specification together with their lowest and highest acceptable values. We also assume that we have historical measurement values needed to derive possible RPC system components, which have been collected and documented by using the unified process data set model described in Section 3. Through synthesis, simulation, and trial and error, we can determine the RPC system components needed and organize them in a way that all the process activities are performed with the expected dynamic performance and within the specified steady-state error, and as a result of this, the established or improved RPC system satisfies the given specification. The general steps for the RPC system design are:

(1) Map the specification of the RPC system to be designed to its expected dynamic performance and acceptable steady-state error. Here, let the targeted settling time $t_{sG} = T_G$, and if Q_G is represented by the tolerable number of defects to be found per page of the resulting application system requirement specification, let acceptable steady-state error

$$e_{ssG} = Q_G \times \sum_{i=1}^{r_1} A_{2Ei}.$$

(2) Identify RPC system components, relationships among them and between them and the whole system. As we have done for the REGPG in Table 1, this involves identifying the activities that comprise the requirements process and mapping them to corresponding RPC system components.

(3) According to the result of step 2, historical process data sets, and the definitions in Section 4, calculate the estimated delay time t_{dE}, rise time t_{rE}, settling time t_{sE} and steady-state error e_{ssE} of the designed RPC system. The people performing process activities in the system, the software and hardware needed constitute the main part of the estimated cost C_E of this RPC system solution. The availability of historical process data requires that progress measurement is

systematically collected by the organization and this acts as a prerequisite.

(4) Compare t_{sE}, e_{ssE} and C_E with t_{sG}, e_{ssG} and C_G. If the specification of the RPC system is satisfied, then go on with step 6, otherwise execute step 5.

(5) Partly, or as a whole, adjust the structure of the designed system (i.e. the relationships among system components and/or between the system components and the whole system), and/or add, remove and replace system components. Selection of which components to adjust, replace, etc. is informed by knowledge about which requirements process activities represent which RPC components, the strengths and weaknesses of the practices used to enact these activities and the costs and benefits of alternative practices. Return to step 3.

(6) If there is more than one design satisfying t_{sG}, e_{ssG} and C_G, we may further compare their t_{dE}s and t_{rE}s, and select the solution with quicker transient-response and less steady-state error.

It should be noticed that:

- Although development techniques, available software tools, staff experience, the characteristics of computer hardware, application system size and complexity differ in software organizations and projects in an organization, many remain unchanged in the context of one project. However, because of the effect of the learning curve, people performing process activities may have different work experience and familiarity with the project. This may manifest itself in different measurement values in the process data sets collected. In general, people with more development experience and familiarity with the project at hand work more effectively and their products are of better quality. While designing the RPC system to be established or improved, it is important to achieve a compromise between the dynamic performance and the steady-state behavior of the resulting system. In general the performance of the designed system should satisfy the specification, but is not necessarily better than that or optimized.
- When identifying the compensators in the system, it is important to take into consideration the specification of the RPC system, the characteristics of the information in the system, the components available for compensator selection, the convenience of management, the requirements of adaptation and cost-effectiveness, the context of the project, and the experience of the people designing the RPC system. This is in order that the compensated system can produce the requirements specification of the specified Q_G representing the stakeholders' requirements within T_G and C_G.
- When identifying the measuring elements in the RPC system to be designed, it is important to select those with more types of defects found, less time delays, and higher percent of defects detected since the upper bound of the static accuracy of the RPC system is determined by the measuring elements.
- The actual performance of the RPC system, which has been designed to satisfy all the specifications, may deviate from the expectation in operation. However, step 3 can still be used to predict the future behavior of the RPC system. When needed, different control strategies can be evaluated by steps 1–6 with updated specification and input of the RPC system and adjustments in the system components and their relationships.

6. The REGPG as an RPC system

From the RPC system component point of view, Fig. 3 shows the 66 RE process activities, the subset of these that the REGPG terms the 'top 10' guidelines recommended for any organization commencing basic RE process improvement, and the subset that are specifically recommended for critical systems (Sommerville and Sawyer, 1997). Fig. 4 shows the good practices selected as the RE improve-

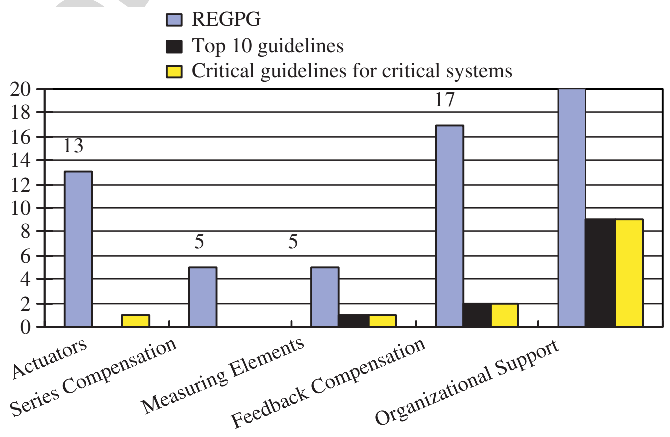

Fig. 3. The distribution of RPC system components recommended in the REGPG.

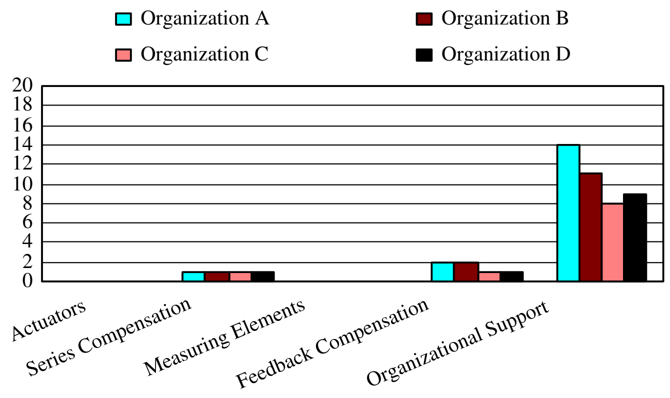

Fig. 4. The distribution of RPC system components selected in (Kauppinen et al., 2002).

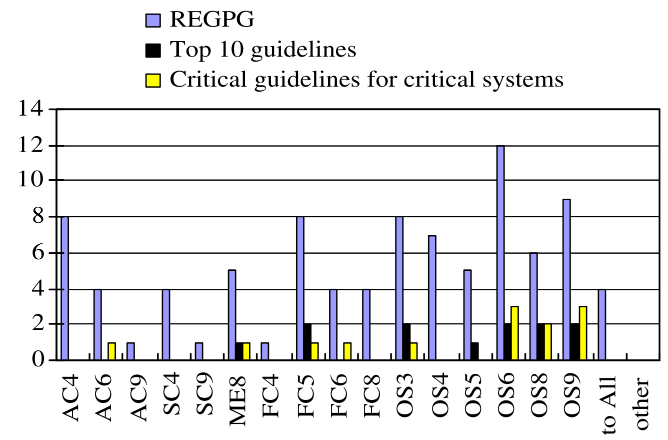

Fig. 5. Detailed recommendation of RPC system components in the REGPG.

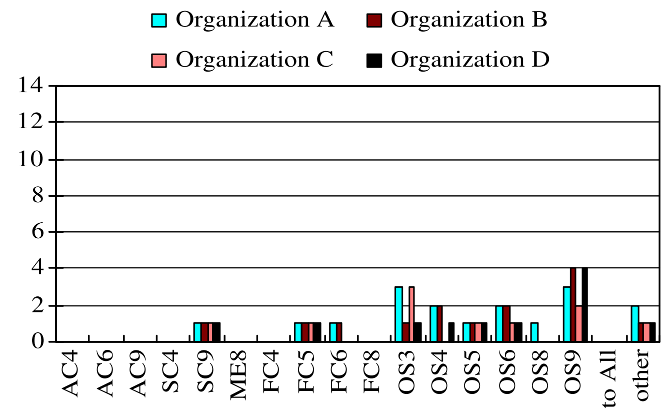

Fig. 6. Detailed selection of RPC system components in (Kauppinen et al., 2002).

ment actions by the four Finnish organizations described in (Kauppinen et al., 2002). The detailed RE process areas in the RPC system components are shown in Figs. 5 and 6 respectively. It can be seen that:

- There are only five good RE practices that have been recommended in the REGPG in the category of measuring elements with 18 in actuators and series compensation, 17 in feedback compensation, and 51 in organizational support. Practices in the requirements validation area, which help improve the performance of measuring elements, account for 23.5% in the category of feedback compensation and 11.8% (or 15.7% considering those two practices relating to collecting and learning from incident experience) in organizational support.
- In the case of top 10 basic guidelines and those particularly important for critical systems, there is one practice in the category of measuring elements and two in the requirements validation area in organizational support. These account for 8.3% and 16.7%, respectively of the top 10, and 7.7% and 15.4% of the critical systems good practices. No feedback compensation for measuring elements has been suggested.
- As to the four Finnish organizations, no measuring elements, no feedback compensation and no organizational support in the requirements validation area (except organization A who selects one practice in the last category and one relating to defining guidelines for formal requirements inspections) are selected as the RE improvement actions.

It is too early to claim with certainty that adapted principles of conventional control theory presented in this paper are immediately deployable in RE process improvement. However, we have learned that more good RE practices relating to measuring elements, including those in feedback compensation and organizational support, should be introduced into the existing REGPG framework if an RE process improvement model can really operate as a process control system.

This is an interesting finding since the collection of quantitative historical data is generally considered to be feasible only by organizations whose processes are already mature. For example, in the CMMI (CMU, 2002), quantitative data collection is mandated at level 4 (*Quantitatively Managed*). The state-of-the-art in RE processes at the time that the REGPG was designed was such that the characteristics of mature RE processes were uncertain. There simply were not any mature RE processes known to the authors from which guidelines and practices for the higher levels in the REGPG maturity model could be synthesized. For this reason, the REGPG is limited to the levels that correspond broadly to the lower three of the CMMI. Given this fact, it is not surprising that the REGPG is deficient in measurement practices. Nevertheless, it interesting that our work has indirectly validated the focus on measurement at the CMMI's upper levels.

7. Related work and conclusions

Recognizing the emerging field of software cybernetics, the first international workshop on software cybernetics, in conjunction with COMPSAC 2004, was held in Hong Kong, China in 2004: "This workshop is motivated by a strong perceived need for formalization in the area of software development process (IWSC, 2004)". The ideas of software cybernetics, the state-of-the-art, "the on-going work in the area of software cybernetics and the background on which it rests and borrows from", and the potential research and application topics are summarized in (Cai et al., 2003; Cai and Chen, 2002). What distinguishes our work from the existing efforts to explicitly apply the concepts, principles or approaches of control theory to the research on software process (Cai et al., 2003; Cai and Chen, 2002; FEAST, 2001; Lehman, 1998; Lehman, 2001; Lehman, 2000; Cangussu et al., 2003; Cangussu et al., 2002) lies in:

- Problem scope. Our work focuses on the RE process. However, this is for essentially pragmatic reasons given the experience of two of the authors in this area. The scope of the work could be widened to apply to software process improvement models that spanned the whole lifecycle.
- Modeling method. We view the software process as a software process control system that consists of different system components. We apply modeling using the conventional block diagrams in control theory and later try to integrate this with statistical techniques. Other work has either has paid attention to the specific role of feedback mechanisms or considered the process under study as a black-box plant, and has taken different approaches to the problem such as system dynamics and state space modeling.

The reason there is little, if any, parallel work on the adaptation of the principles of conventional control theory to the field of software process may be that it is difficult to develop adequate models in the sense of conventional control theory of different system components and the software process control system as a whole. "Even an experienced control engineer can do nothing without a model of the system to be controlled (Lehman, 2001)". Moreover "Conventional control requirements such as stability, rise time and overshoot may no longer be appropriate (Cai et al., 2003)". Our work opens a new way to system modeling in the domain of classical control theory, presents a new perspective of control requirements, and gives some new ideas about the adaptation of control theory to the field of software processes.

To summarize, we have adapted the principles of conventional control theory to define the RE process as an RPC system, reorganize the RE activities as different types of RPC system components, map the goals of RE process establishment and improvement to the specifications of such a system, analyze its dynamic and steady-state performance. By doing so, we have:

- Structured RE good practices. Not only can companies know what these RE good practices are, but also their roles in the RPC system. It becomes possible to apply the knowledge about control theory to the selection of RE process models, technologies, and good requirement practices given the resources available.
- Related the individual behavioral performance to the dynamic RE process and up to the wider organizational context and business goals. Not only can companies identify the strengths and weaknesses in their RE processes across the board, but also the factors both within and external to the processes that may impact the behavior and performance of the RPC systems. In principle, therefore, it becomes possible to systematically start and/or incrementally improve the RE processes, and objectively see the progression at individual, team and organizational levels according to the improvement efforts.
- Outlined an active Requirements Process Control (RPC) System framework. Not only can companies flexibly configure their RE processes, but also quantitatively evaluate different RE process solutions and their benefits using unified system performance indicators. It even becomes possible to make sensible comparisons of RE capability maturity and process improvement across companies.

The RPC system is an active, continuous, configurable requirement process establishment and improvement framework. To make it feasible in practice and complete in theory, we need to elaborate, assess and improve the RPC process framework in real industrial settings. By collecting RE process data using the unified data set model defined in the RPC system framework and employing statistical techniques and/or others, we may further extract different RPC system components and their process patterns, study the stability and performance relation between them and the whole RPC system, and conduct the error analysis of the RPC system modeled as opposed to the actual one. We need to develop tools superimposed on the existing SPIToolkit, (a web application supporting software process management, improvement and integration by producing, organizing and utilizing knowledge based on data collected from software processes (Xu, 1999, 2004) to facilitate the computing in the RPC system framework and hence its practical application. It should then be possible to evaluate the extent to which the RPC system framework can be exploited for establishing, managing and improving industrial RE processes and may further give possible support to more objective, accurate, reliable, repeatable and comparable RE process assessment.

A feature of the work is that it requires the collection of consistent historical data on RE process activities. This limits the practical application of classical control theory and its deployment through our RPC system to mature organizations. However, we believe that our work has shown that classical control theory offers significant insights into the nature of software processes and, crucially, a means for classifying and understanding good practices and the real role they play within a software process.

Acknowledgement

The presented research is partly funded by China Scholarships No. 3006(2003) and No. 3122(2004).

Appendix A. The 66 key practice areas viewed from the perspective of different RPC system components

RPC system components	RE process area	Good practice
Actuators	Feasibility study (AC4)	4.1 Assess system feasibility
	Requirements elicitation (AC4)	4.3 Identify and consult system stakeholders 4.5 Define the system's operating environment 4.6 Use business concerns to drive requirements elicitation 4.7 Look for domain constraints 4.9 Collect requirements from multiple viewpoints 4.11 Use scenarios to elicit requirements 4.12 Define operational processes
	Requirements description (AC6) (System modeling)	6 Describe requirements 7.1 Develop complementary system models 7.2 Model the system's environment—4.5, 4.6, 4.11, 4.12 7.3 Model the system architecture
	Requirements management (AC9)	9 Manage requirements (more important to Critical Systems Domain)
Series compensation	Feasibility study (SC4)	10.3 Identify and analyze hazards (Critical Systems Domain)
	Requirements elicitation (SC4)	4.10 Prototype poorly understood requirements 4.13 Reuse requirements 10.4 Derive safety requirements from hazard analysis (Critical Systems Domain)
	Requirements management (SC9)	9.5 Use a database to manage requirements (more important to Critical Systems Domain)
Measuring elements	Requirements validation (ME8)	8.1 Check that the requirements document meet your standards 8.2 Organize formal requirements inspections 8.5 Use prototyping to animate requirements 8.6 Write a draft user manual 8.7 Propose requirements test cases
Feedback compensation	Requirements elicitation (FC4)	4.10 Prototype poorly understood requirements
	Requirements analysis (and negotiation) (FC5)	5.1 Define system boundaries 5.2 Use checklists for requirements analysis 5.4 Have requirements negotiation meetings 5.5 Prioritize requirements 5.6 Classify requirements using a multi-dimensional approach 5.7 Use interaction matrices to find conflicts and overlaps 5.8 Assess requirements risks 10.5 Cross-check operational and functional requirements against safety requirements (Critical Systems Domain)
	Requirements description (FC6) (System modeling)	7.1 Develop complementary system models 7.2 Model the system's environment 7.6 Document the links between stakeholder requirements and system models 10.6 Specify systems using formal specifications (Critical Systems Domain)
	Requirements validation (FC8)	8.3 Use multiple-disciplinary teams to review requirements 10.2 Involve external reviewers in the validation process (Critical Systems Domain) 8.8 Paraphrase system models (Critical Systems Domain) 10.5 Cross-check operational and functional requirements against safety requirements (Critical Systems Domain)

(*continued on next page*)

Appendix A (*continued*)

RPC system components	RE process area	Good practice
Organizational support	Requirements documentation (OS3)	3.1 Define a standard document structure
		3.2 Explain to use the document
		3.3 Include a summary of the requirements
		3.4 Make a business case for the system
		3.5 Define specialized terms
		3.6 Lay out the document for readability
		3.7 Help readers find information
		3.8 Make the document easy to change
	Requirements elicitation (OS4)	4.2 Be sensitive to organizational and political considerations
		4.4 Record requirements sources
		4.8 Record requirements rationale
		4.9 Collect requirements from multiple viewpoints
		4.10 Prototype poorly understood requirements
		4.11 Use scenarios to elicit requirements
		4.13 Reuse requirements
	Requirements analysis and negotiation (OS5)	5.2 Use checklists for requirements analysis
		5.5 Prioritize requirements
		5.6 Classify requirements using a multi-dimensional approach
		5.7 Use interaction matrices to find conflicts and overlaps
		5.8 Assess requirements risks
	Requirements description (OS6) (System modeling)	6.1 Define standard templates for describing requirements
		6.2 Use language simply, consistently and concisely
		6.3 Use diagrams appropriately
		6.4 Supplement natural language with other descriptions of requirements
		6.5 Specify requirements quantitatively
		7.1 Develop Complementary system models
		7.2 Model the system's environment
		7.3 Model the system architecture
		7.4 Use structured methods for system modeling
		7.5 Use a data dictionary
		7.6 Document the links between stakeholder requirements and system models
		10.6 Specify systems using formal specifications (Critical Systems Domain)
	Requirements validation (OS8)	8.2 Organize formal requirements inspections
		8.4 Define validation checklists
		10.1 Create safety requirement checklists (Critical Systems Domain)
		8.5 Use prototyping to animate requirements
		8.7 Propose requirements test cases
		8.8 Paraphrase system models (Critical Systems Domain)
	Requirements management (OS9)	9.1 Uniquely identify each requirement
		9.2 Define policies for requirements management
		9.3 Define traceability policies
		9.4 Maintain a traceability manual
		9.5 Use a database to manage requirements
		9.6 Define change management policies
		9.7 Identify global system requirements
		9.8 Identify volatile requirements
		9.9 Record rejected requirements (more important to Critical Systems Domain)
	To All	10.3 Identify and analyze hazards
		10.7 Collect incident experience
		10.8 Learn from incident experience
		10.9 Establish an organizational safety culture (Critical Systems Domain)

References

Cai, K-Y., Chen, T., 2002. Towards research on software cybernetics, In: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering, Tokyo, Japan, pp. 240–241.

Cai, K-Y., Cangussu, J., DeCarlo, R., Mathur, M., 2003. An overview of software cybernetics, In: Proceedings of the 11th Annual International Workshop on Software Technology and Engineering Practice, Amsterdam, The Netherlands, pp. 77–86.

Cangussu, J., DeCarlo, R., Mathur, A., 2002. A formal model of the software test process. IEEE Transactions on Software Engineering 28 (8), 782–796.

Cangussu, J., DeCarlo, R., Mathur, A., 2003. Using sensitivity analysis to validate a state variable model of the software test process. IEEE Transactions on Software Engineering 29 (5), 430–443.

CMU, 2002. Capability maturity model integration (CMMI). Version 1.1, CMU/SEI-2002-TR-029.

FEAST, 2001. Available from: <http://www.doc.ic.ac.uk/~mml/feast/>.

Humphrey, W., 2000. Introduction to the Team Software Process. Addison-Wesley.

IWSC, 2004. The first international workshop on software cybernetics. Available from: <http://rachel.utdallas.edu/compasc/>.

Kauppinen, M., Kujala, S., 2001. Assessing requirements practices with the REAIMS model: lessons learned, In: Proceedings of the 11th Annual Internation Symposium of the International Council on Systems Engineering, Melboune, Australia.

Kauppinen, M., Aaltio, T., Kujala, S., 2002. Lessons learned from applying the requirements engineering good practice guide for process improvement, In: Proceedings of the 7th european conference on software quality, Helsinki, Finland, pp. 73–81.

Kitchenham, B., Hughes, R., Linkman, G., 2001. Modeling software measurement data. IEEE Transactions on Software Engineering 27 (9), 788–804.

Lehman, M., 1998. FEAST/1 Final Report—Grant Number GR/K86008.

Lehman, 2000. Rules and tools for software evolution planning and management, FEAST 2000 workshop, London, UK.

Lehman, M., 2001. FEAST/2 Final Report—Grant Number GR/M44101.

Nikula, U., 2003. Experiences from lightweight RE method evaluations, In: Proceedings of the 1st International Workshop on Comparative Evaluation in Requirements Engineering, California, USA, pp. 53–60.

Nikula, U., Sajaniemi, J., 2002. BaSyRE: a lightweight combination of proven RE techniques, In: Proceedings of International Workshop on Time-Constrained Requirements Engineering, Essen, Germany, pp. 69–78.

Ogata, K., 1970. Modern Control Engineering. Prentice-Hall.

Regnell, B., Beremark, P., Eklundh, O., 1998. A market-driven requirements engineering process—results from an industrial process improvement programme. Requirements Engineering Journal 3 (2), 121–129.

Sawyer, P., Sommerville, I., Viller, S., 1999. Capturing the benefits of requirements engineering. IEEE Software 16 (2), 78–85.

Sommerville, I., Ransom, J., 2005. An empirical study of industrial requirements engineering process assessment and improvement. ACM Transactions on Software Engineering and Methodology (TOSEM) 14 (1), 1–33.

Sommerville, I., Sawyer, P., 1997. Requirements Engineering A Good Practice Guide. John Wiley & Sons, Chichester.

Xu, H., 1999. Put CMM/TSP/PSP in Practice for Software Quality Assurance, 863 (Hi-Tech Research and Development Program of China) Workshop on the System of Software Quality Assurance, Beijing, China.

Xu, H., 2004. Software Process Management, Improvement and Integration Toolkit, Technical Report, BHU/SEI TR SPIToolkit-5.

Hong Xu received the MS and BS degrees in Electrical Engineering from BeiHang University. She is an Associate Professor in the School of Computer Science and Engineering at BeiHang University. Her main research interests are centered on software process establishment and improvement, quality management and control, and software cybernetics. She has been a visiting researcher in software engineering in the Computing Department of Lancaster University. She was involved with several national software engineering research projects and international system engineering environment development projects, and awarded prizes for the major contribution towards the progress of science and technology.

Pete Sawyer is a senior lecturer in the Computing Department at Lancaster University and Associate Dean for Postgraduate Studies in the Faculty of Science and Technology. He received his B.Sc. in 1985 and his Ph.D. in Computer Science from Lancaster University in 1990. His main research interests are in the general area of software systems engineering, particularly requirements engineering, service-centric systems, system dependability and aspect-oriented software development. He is chairman of the Requirements Engineering Specialist Group (RESG) of the British Computer Society (BCS, is a member of the BCS, the IEEE Computer Society and the ACM and is a chartered engineer.

Ian Sommerville has a B.Sc. in Physics from Strathclyde University, Scotland and an M.Sc. and Ph.D. in Computer Science from St. Andrews University, Scotland. He has held a Chair in Computer Science at Lancaster University since 1986. Sommerville's principal research interests are in software systems engineering with a particular focus on system dependability, requirements engineering and social informatics. He leads a group of about 25 academics, research staff and students working in these areas at Lancaster. He is currently principal investigator on research projects concerned with system dependability, service-centric systems engineering and social informatics. Sommerville has published around 50 refereed journal papers, more than 70 refereed conference papers and have about 20 invited papers included in journals, books and conference proceedings. He has also published a widely used textbook in Software Engineering which is now in its 7th edition and which is widely adopted worldwide. In addition, he has published (with P. Sawyer) a book on requirements engineering practice, aimed at engineers in industry and (with G. Kotonya) a student textbook on requirements engineering.