Technical Model for Remediation (Workpackage 2.2)

Simon Denier, Jannik Laval, Stéphane Ducasse, Fabrice Bellingard

▶ **To cite this version:**

Simon Denier, Jannik Laval, Stéphane Ducasse, Fabrice Bellingard. Technical Model for Remediation (Workpackage 2.2). [Research Report] 2010, pp.18. inria-00533659

HAL Id: inria-00533659
https://inria.hal.science/inria-00533659v1

Submitted on 8 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Technical Model for Remediation

Modèle technique de remédiation

Workpackage 2.2

November 8, 2010

This deliverable is available as a free download.

Copyright © 2010, 2009 by F. Balmas, F. Bellingard, S. Denier, S. Ducasse, J. Laval, K. Mordal-Manet.

The contents of this deliverable are protected under Creative Commons Attribution-Noncommercial-ShareAlike 3.0 Unported license.
You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Noncommercial. You may not use this work for commercial purposes.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

- For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page: `creativecommons.org/licenses/by-sa/3.0/`
- Any of the above conditions can be waived if you get permission from the copyright holder.
- Nothing in this license impairs or restricts the author's moral rights.

Your fair dealing and other rights are in no way affected by the above. This is a human-readable summary of the Legal Code (the full license):
`http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode`

Second Edition, October, 2010.

Workpackage: 2.2

Title: Technical Model for Remediation, First version

Revision: 1.0

Authors: S. Denier, J. Laval, S. Ducasse, F. Bellingard

Planning

- Delivery Date: October 2010
- First Version: March 2010

Contents

Chapter1. Objective and Technological Challenges

The objective of this workpackage is the definition of the global remediation effort on a project. It defines strategies for upgrading quality through assessment and organization of the single remediation tasks as described in WorkPackage 2.1.

The primary stance driving the remediation work in the Squale model is that quality should be increased incrementally. A remediation plan takes into account the defective components as well as the violated practices. However, such a plan must also be effective and provide as much as improvements at the lowest cost.

Different team cultures can come up with different ways to build a plan and split it in tasks between team members. It can involve technical architects, team leaders, and developers. The strategy to build the remediation plan is to be customized on a case-by-case basis.

Related Work. Identifying the changes and computing their cost in a component is definitively a challenge. Several works already covered such aspect and we identified the following work for the interested reader:

- Cost estimation model such as Cocomo, even if they do not take into account the potential of changes [Kem87].
- Refactorings (refactorings are behavior preserving operations [Rob99, FBB+99]) and maintenance actions [MT04, BMZ+05].
- Prediction models for maintenance effort [ME98, BB99, JJ97, Put78].
- Change impact model [BA96].

Chapter2. Problem Analysis: Aspects of Remediation Plans

Remediation plans have multiple characteristics which govern their scope and application in any company. The quality process aiming at remediation should be aware of such characteristics to make informed decisions. We discuss two aspects of such plans: the characteristics of tasks tackled by the plan, and the organizational aspect of a plan in development teams and during development process.

The following terms define a common vocabulary for description of remediation plans.

Transgression: a failed instance of a practice *i.e.,* either a component or a rule transgression.

Remediation task: descriptions of actions to be taken to resolve one transgression of a practice (see Workpackage 2.1).

Remediation cost: cost characterizing the work required to apply one remediation task (see Workpackage 2.1).

Remediation plan: ordered list of remediation tasks, from top priority to lowest priority.

Workload: work required to resolve all transgressions of a single practice.

2.1 Remediation tasks

A remediation task is the conceptual unit describing the actions to be taken to resolve one transgression *i.e.,* a failed practice on one component (or one rule transgression). The range and effort implied by a task vary greatly. For example, one task can target a single class which violates the number of methods practice, while another can target all methods which violate naming practice. Workpackage 2.1 describes for each practice the tentative remediation task *i.e.,* how the kind of transgression detected by the practice should be solved.

We identify three intrinsic characteristics to assess the work required by a remediation task:

- scope of practice and remediation;
- degree of automation of the remediation task;
- expertise required for remediation (technology-wise).

Four more characteristics are contextual, depending on the project and company standards:

- code organization and ownership;

Practice	Scope	Scope of remediation
Audit	Project	Selection of top priority components identified by the audit
Metric Model Test	Component	Component and related (package, class, method...)
Rule	Rule transgression	Component enclosing transgression

Table 2.1: Scope of practices and remediation.

- criticality of practice, depending on company standards and current stage in project lifecycle (for example, test practices are stressed in pre-release stage);
- criticality of component (as ruled by development team);
- characteristics of component (complexity, coupling, size...).

Eventually, the number of transgressions sets the workload to resolve all transgressions of the practice. We now discuss each characteristic in more details.

Scope of practice and remediation. This characteristic assesses what is the target of remediation in the project. Indeed, three kinds of practice identify three different scopes of transgressions. Then, remediation of those transgressions happens at a related scope as described in Table 2.1.

An audit targets the whole project and do not singularize components by construction. However, the audit expert may recommend some specific components which should have top priority in remediation plan. Most practices target a single component. The remediation for the practice can then target a single component, but may affect related components in some cases. Practices based on rule checking identify rule transgressions, which can be multiple for a single component. However, the rule of thumb is, for a single component, to collect all transgressions of a practice and correct them at once.

Automation of remediation. Some remediation tasks can be automated with few inputs from the developer. This is the case for example of practices targeting conformance to some coding standards. In those cases, it is better to target all transgressions at once across components. Indeed, the value of conformance comes from the high number of components following standards, promoting homogeneity and making subsequent actions easier.

Technological expertise. While standard-based practices are easy to follow, other practices require an expert in their field to be dealt with diligently. For example, tasks related to architecture and design, or security require an expert in their respective field.

Code organization and ownership. The remediation plan needs to take into account code organization in the project as well as code ownership. Some components may be out of bounds. As much as technological expertise is needed, expertise about the target

components and their history of development is needed to deal efficiently with some transgressions.

Criticality of practice. Not all practices are equally important during the different stages of development lifecycle. For example, testing practices should be stressed in testing stage. Moreover, due to specific requirements, project manager can stress some practices such as portability. This can play on the priority of the practice in remediation plans.

Criticality of component. Knowledge of project internals can lead to stress actions on components which are central in the current stage of development. On the other hand, some faulty components can be declared out of bounds if they work otherwise with no evolution planned.

Characteristics of component. Depending on the practice, characteristics such as size, complexity, coupling, can have a deep impact on the correction required by a transgression. Some practices state explicitly such a correlation: for example, providing more tests for complex methods.

2.2 Organizational aspects of remediation

Each company has its own policy of managing quality and applying remediation at different levels in development teams, from the individual developer to the technical manager. Quality and remediation might be used by each developer to assess its own work and correct it. The technical manager can use it to assess the health of its projects and target more critical components or practices, depending on the life-cycle of the project. Quality team can use it to assess the respect of standards in different projects and come with new recommandations.

Those different scopes of quality and remediation are of equal importance to build an efficient software quality process:

- The developer can manage its own work on daily basis and do not feel the software quality process as a burden. It is part of a self-applied discipline[1].
- The technical manager can use this tool to assess the situation and direct the effort.
- The quality team tailors the model to the needs and standards of the company. This is best done in iterative manner by analyzing previous model instances.

2.3 Basic strategies

Independently of higher level criteria such as cost, profitability, or criticality, two basic strategies are always available when tackling a faulty practice:

- slightly improve marks for many components;
- improve marks of worst components.

[1] In current instances of Squale model, some programmers consider the process as a game, trying to achieve the best marks.

Weighting applied by the global practice function (see WP1.3) dictates the best strategy: hard weighting pushes stress on bad marks, eliciting the remediation of worst components; soft weighting leverages all marks, making a slight increase in many components more interesting. It also depends on the complexity and automation of practice remediation. As noted in Section 2.1, rule-based practices for conformance should be corrected by batch of components, while practices requiring more expertise are more likely to be tackled by focusing on few bad components.

2.4 Remediation plan objectives

Based on the previous aspects, development teams may have two objectives to remedy software quality issues:

- achieve a significant reduction of risks by targeting critical components and practices. We name it the *Risk model*.
- achieve the best profitability in quality process *i.e.,* the best compromise between quality raise and remediation costs from Workpackage 2.1. We name it *Profitability model*.

Each objective relies on different models to be achieved. The following sections describe such models.

Chapter3. Current Remediation Approach in Squale

The purpose of the current remediation model is to first reduce risks by targeting worst marked practices then achieve some profitability by prioritizing remediation tasks with the lowest work to do. This model is practice-oriented, in that it only prioritizes the practices to be corrected, independently of the touched components. This model is tailored towards the developer, which can use hints from the remediation plan to perform quality tasks on a daily basis.

3.1 Mixed priority-cost strategy

The current strategy to prioritize remediation tasks follows four principles:

1. practices with lowest marks should have higher priority;
2. components which fail a practice should all be corrected before making improvement to others;
3. some practices are easier to correct than others;
4. the workload necessary to correct a practice is a function of the practice itself and the number of transgressions of the practice.

Each principle is embodied by different steps and metrics of the algorithm for planning remediation illustrated by Figure 3.1.

Practice prioritization based on marks. The Squale model defines three practice levels: “refused”, “accepted with reserve” and “accepted”. Depending on its global mark, each practice belongs to one of these levels. “Refused” practices have higher priority in the remediation plan, next are “accepted with reserve” practices, then “accepted” practices can be scheduled for improvement.

Mark	Level
$[0, 1]$	Refused
$]1, 2]$	Accepted with reserve
$]2, 3]$	Accepted

Component selection per practice. Given the level of a practice, only components which are ranked at the same level or below should appear in the remediation plan. For an “accepted with reserve” practice, only “accepted with reserve” components and “refused” components will be dealt with. “Accepted” components can only be interesting for improvement, not remediation. However, for rule-based practices, all transgressing components should be selected for remediation.

Correction coefficient per practice. A correction coefficient is assigned to each practice. It asserts the relative effort to correct a single transgression of the practice, compared to other practices. It is a constant and as such is independent of the transgressing component. This takes into account, for example, that a transgression of formatting

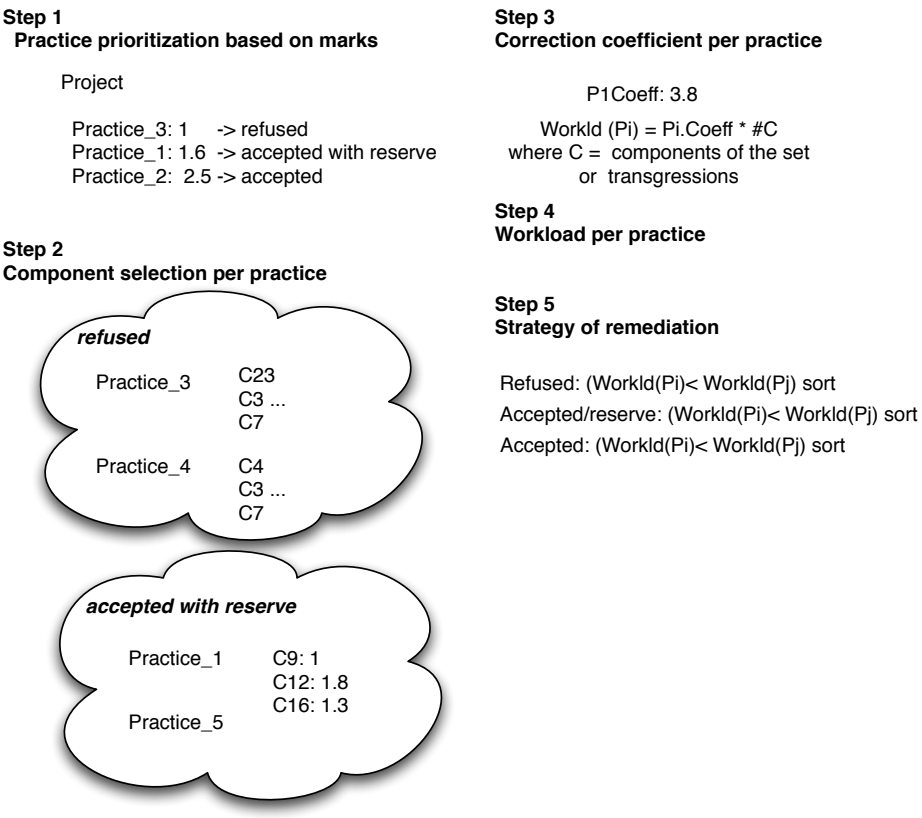

Figure 3.1: Steps

standards is easier to correct than a design issue due to coupling. Table 3.1 gives an example of correction coefficients for practices customized for Java projects.

Workload per practice. Given that the correction coefficient for a practice is constant, the workload to correct all transgressions of the practice is defined by the product of the coefficient by the number of transgressions. For metric-based practices, the number of transgressions is the number of selected components. For rule-based practices, this is directly the number of transgressions detected by the rule in selected components.

Strategy of remediation. The idea is to target each level of practices by decreasing risk (from "refused" to "accepted") and, for each level, to begin with practices with the lowest workload.

```
cluster practices by level = {refused, accepted with
    reserve, accepted};
for each level
  for each practice
    select components for remediation;
    compute practice workload as coefficient(practice)
        x number_of_transgressions(practice);
  sort level by workload;
  done;
done;
append levels;
```

3.2 Limitations

There are multiple limitations on the described strategy:

1. The coefficient is a constant, meaning that the remediation cost does not take into account characteristics of the component which may hinder the remediation. For the test coverage practice, the effort to write tests for complex methods (which is required by the practice) gets harder with the complexity itself.

2. Some practices are contradictory, so correcting a practice on a component can degrade other practices.

3. Although this strategy prioritizes practices by marks and effort, it always targets the full system: it can not predict where and when the quality (marks) will be most improved. Consequently, the workload might not be optimal in the end.

    - There is no parameter limiting the cost of the remediation plan. The plan targets a full remediation of the system without considering the total workload.
    - There is no estimation of the new practice mark, as we only give a list of corrections without estimating the improvement on the practice level (and if one follows the action plan, final notation would be very good, but perhaps the effort is not optimal).

Practice Name	Unitary Effort
Stability abstractness level	200
Afferent coupling	150
Efferent coupling	100
Swiss army knife	35
Copy paste	30
Spaghetti code	30
Inheritance depth	20
Lack of cohesion in method	20
Method size	20
Number of methods	20
Depend on child	20
Layer respect	40
Programming standard	15
Dependency cycle	10
Documentation standard	6
Naming standard	4
Formatting standard	1
Documentation	10

Table 3.1: Current effort ponderation for different practices

4. Components involved in multiple transgressions are not detected, which means they would be passed over and over.

Chapter4. Prospective Remediation Models

As stated in Section 2.4, a remediation plan is built with respect to a global quality objective, such as profitability or risk reduction. We discuss in the following sections possible strategies to build remediation plan for the different objectives.

Contrary to the mixed strategy which only considers the priority of practices, the models presented below focus more on components which violate multiple practices, offering to correct most practices at once for a single component. Then the remediation plan describes sequences of tasks focused on the same components. When tackling such a sequence, the programmer will gradually increase or refresh his knowledge of the component, facilitating further correction. We propose two solutions: the *Risk model* and the *Profitability model*.

4.1 Risk model

The risk model targets the reduction of risks with a focus on critical practices and critical components. Both sets of critical practices and critical components can be tailored by the development team following its current development goal. The model identifies components at risk *i.e.,* critical components most involved in transgressing practices, especially critical practices.

The strategy implies a *criticality map* practice to be assessed (and updated regularly) by the team in a preliminary step:

- assessment of criticality coefficients for each practice.

The algorithm computes a risk for each component as a function of practice criticality and component mark. Finally, component then practices for each component are sorted by criticality.

```
for each component
  for each practice
    criticality(practice, component) =
            min(max_criticality, -log( score(practice,
                component) / 3)
                    * criticality(practice));
  done;
  criticality(component)
          = sum( criticality(practice, component) )
          for all concerned practice;
  sort tasks by criticality(practice, component);
  // alternatively
  sort tasks by remediation_cost(practice, component);
done;
sort components by criticality(component);
```

The $-log(mark/3)$ factor weights the criticality score based on the mark. In particular, if $mark = 3$, the criticality score will be automatically nullified.

On the other side of the scale, the criticality is also bounded by $max\_criticality$ to avoid extremely large values for very low scores (a 0 score would give an infinite criticality for the component, regardless of whether the practice is itself critical or not). $max\_criticality$ is an arbitrary constant: for example, it can be defined as $max\_criticality = -log(\frac{1}{10\times3})$, which implies that max criticality is achieved for any score equalled or below $1/10$.

The global order is computed on the criticality of components, itself based on the criticality of practices and the score of the component for the practice (the lower the mark, the higher the criticality). Remediation costs, instead of criticality, can be taken into account to sort tasks within each component.

4.2 Profitability model

The profitability model is an evolution of the mixed priority-cost model, aiming at optimizing the ratio of quality gains versus workload of remediation. To be rationale and efficient, such a model should assess precisely two types of characteristics:

- remediation workload based on practice and component characteristics (such as complexity of components) instead of a constant per practice;
- estimated, or scheduled, gain of quality.

In practice, remediation costs in workpackage 2.1 already combines the two characteristics. A remediation cost is computed as the work necessary to achieve a given mark.

Quality objective. The model does not stick with the three-level clustering in "refused", "accepted with reserve", and "accepted" practices. Instead, the scheduled gain of quality prioritizes a quantitative objective to reach, which allows one to control allocated resources (instead of targeting a full remediation of the system).

For example, one could fix as objectives that a practice global mark should be above 1.2 and that any component mark for this practice should be above 0.8. Given these objectives, remediation costs can be computed.

Strategy. The model requires as input marks to be achieved for each practice and each component. For each practice, a set of candidate components is selected and the remediation cost is computed for each component (as the work necessary to achieve its goal mark, the lower the better). Each component receives a global profitability score from the sum of remediation costs, depending on the practices it is involved. Components are sorted in the remediation plan from most profitable to less profitable.

```
for each component
  for each practice
    compute remediation_cost(practice, component);
  done;
  profitability(component) = sum( remediation_cost(
      practice, component) ) for all concerned practices;
```

```
done;
sort components by profitability;
```

Chapter5. Conclusion

The above work can be extended in further directions:

- combining risk model and profitability model for a better management of profit vs risk; an idea is to draw a graph with on x-axis the measure of the profitability model, and on y-axis the measure of the risk model. The idea is to fix components with high profitability and low risk.
- managing practices with opposing effects: solving *Method Size* by creating new methods can result in a violation of *Number of Methods* at class level. Simulation of remediation results can help to detect such problems.

Bibliography

[BA96] S. A Bohner and R.S. Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, 1996.

[BB99] PerOlof Bengtsson and Jan Bosch. Architecture level prediction of software maintenance. In *European Conference on Software Maintenance and Reengineering (CSMR'99)*, pages 139–147, 1999.

[BMZ+05] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Günter Kniesel. Towards a taxonomy of software change. *Journal on Software Maintenance and Evolution: Research and Practice*, pages 309–332, 2005.

[FBB+99] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[JJ97] Henry J.E. and Cain J.P. A quantitative comparison of perfective and corrective software maintenance. *Journal of Software Maintenance: Research and Practice*, pages 281–297, 1997.

[Kem87] Chris F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 1987.

[ME98] J.C. Munson and S.G. Elbaum. Code churn: A measure for estimating the impact of code change. In *ICSM'98*, pages 24–34, 1998.

[MT04] Tom Mens and Tom Tourwé. A survey of software refactoring. *Transactions on Software Engineering*, 30(2):126–138, 2004.

[Put78] L. H. Putnam. Example of and early sizing, cost and schedule estimate for an application software system. In *Proceedings of COMPSAC'78*, pages 827–832, 1978.

[Rob99] Donald Bradley Roberts. *Practical Analysis for Refactoring*. PhD thesis, University of Illinois, 1999.