Chapitre 3 : MODELES DE MAINTENANCE ET D’EVOLUTION

La maintenance du logiciel a des caractéristiques uniques même si elle partage plusieurs caractéristiques similaires avec le développement logiciel, parmi ces caractéristiques uniques on distingue :

* Les contraintes du système existant.
* Des délais plus courts.
* Disponibilité des données de test.

La maintenance du logiciel doit avoir son propre modèle de cycle de vie (SMLC) puisqu’elle implique plusieurs activités propres à elle. Dans ce chapitre nous décrirons trois modèles de maintenance : réutilisation (reuse), arrangement simple en étapes (simple staged) et modification mini-cycle (change mini-cycle) représentant respectivement d’anciens modèles, de nouveaux modèles et des modèles faisant l’objet de recherche actuelles.

1. **Modèle orienté réutilisation (REUSE-ORIENTED MODEL)**

On obtient une nouvelle version d'un ancien système par modification d'un ou plusieurs composants de l'ancien système et éventuellement par l'ajout de nouveaux composants. En conséquence, le nouveau système est susceptible de réutiliser de nombreux éléments de l'ancien système.

Basé sur ce concept, trois modèles de processus pour l'entretien ont été proposés par Basili [[1]](#footnote-1):

* Modèle de solution rapide (Quick fix model). : dans ce modèle, les modifications nécessaires sont rapidement apportées au code puis à la documentation correspondante (Figure 1).

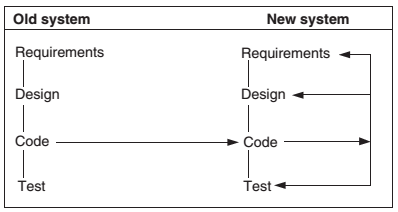


FIGURE 1. The quick fix model. 1990 IEEE

* Modèle d'amélioration itérative (Iterative enhancement model) : Dans ce modèle, comme illustré sur la figure 2, les premières modifications sont apportées aux documents de plus haut niveau. Finalement, les modifications sont propagées jusqu'au niveau du code.

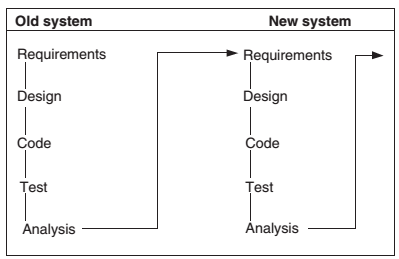
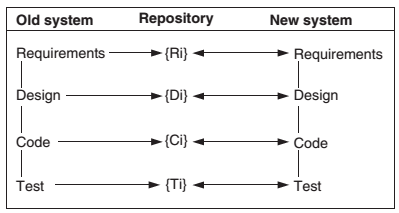


FIGURE 2. The iterative enhancement model. 1990 IEEE

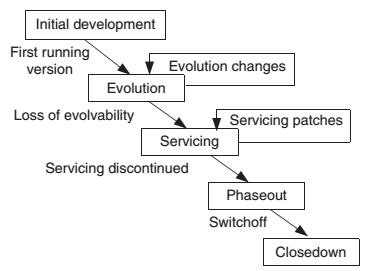
* Modèle de réutilisation totale (Full reuse model) : dans ce modèle, comme illustré à la figure 3, un nouveau système est construit à partir de composants de l'ancien système et d'autres composants disponibles dans le référentiel.



**FIGURE 3**. The full reuse model. © 1990 IEEE

1. **Modèle en étapes pour applications fermées (THE STAGED MODEL FOR CLOSED SOURCE SOFTWARE)**

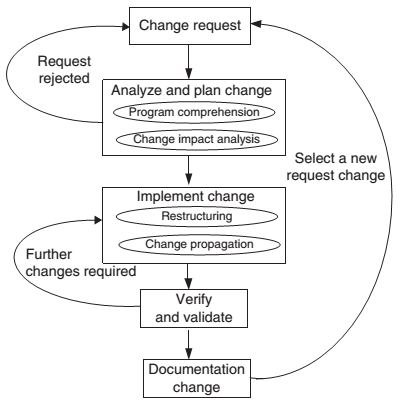
Rajlich et Bennett [[2]](#footnote-2) ont défini un modèle en étapes pour représenter le cycle de vie des applications fermées traditionnelles -- commercial Closed Source Software (CSS). Leur modèle est composé de cinq étapes comme illustré à la figure 4.



**FIGURE 4.** The simple staged model for the CSS life cycle. 2000 IEEE

* Développement initiale : Développe la première version fonctionnelle du logiciel.
* Evolution. : les développeurs améliorent les fonctionnalités et les capacités du logiciel pour répondre aux besoins et aux attentes du client.
* Entretien. Les développeurs ne corrigent que des défauts mineurs et d'urgence, et aucune fonctionnalité majeure n’est incluse.
* Elimination progressive : dans cette phase, aucun autre entretien n’est effectué, durant cette étape les vendeurs cherchent à générer des revenus aussi longtemps que possible.
* Abandon : Le logiciel est retiré du marché, et les clients sont dirigés à migrer vers une solution de remplacement.

1. **Modèle de modification mini-cycle (CHANGE MINI-CYCLE MODEL)**



**FIGURE 5**. The change mini-cycle. © 2008 Springer

Le modèle de modification mini-cycle se compose de cinq grandes phases : (1) RM, (2) analyser et planifier les modifications, (3) implémentation de la modification, (4) vérifier et valider, et (5) la modification de la documentation. Dans ce modèle de processus, de nouvelles activités ont été identifiées afin de refléter le fait que les modifications sur le logiciel sont rarement isolées. Des exemples de ces nouvelles activités sont l'analyse de l'impact du changement et de la propagation des modifications.

* **Requête de modification** : Une RM provient généralement de la direction, les utilisateurs du logiciel, ou des clients. Une RM peut prendre l'une des deux formes suivantes : un rapport de défaillance ou une demande d'amélioration.
* **Analyser et planifier les modifications** : dans la deuxième phase, la compréhension du programme et l’analyse d'impact sont effectuées. La compréhension du programme est essentielle pour comprendre quelles parties du logiciel peuvent être affectées par une RM. L'analyse d'impact est réalisée afin d'identifier les conséquences potentielles d'une modification et estimer les ressources nécessaires pour accomplir le changement. A travers l’analyse de l'impact, un système logiciel est analysé par le personnel de maintenance pour identifier les composants logiciels qui seront touchés par une RM. Les objectifs de l’analyse de l’'impact sont les suivants :
  + pour déterminer l'ensemble des composants du système affectés, compte tenu d’un SIS identifié par l’activité de compréhension du programme ;
  + d'établir des estimations précises des ressources nécessaires pour accomplir la tâche de mise en œuvre ; et
  + d'analyser le coût et les avantages de la RM et de prendre une décision sur l'opportunité à effectuer ou pas la RM.
* Implémentation de la modification : La RM est implémentée après que l’étude de la faisabilité de la modification soit établie. Toutefois, avant l’implémentation de la RM, une restructuration ou une rénovation (software refactoring) du logiciel est effectuée afin de répondre aux besoins de la modification demandée.
* Vérifier et valider. Dans cette phase, le logiciel est vérifié et validé afin de garantir que l'intégrité du système n'a pas été compromise. Cette activité comprend l'examen du code, les tests de régression, et l'exécution de nouveaux tests si nécessaire.
* Modification de la Documentation : la phase finale du modèle modification mini-cycle concerne la mise à jour de la documentation du programme qui peut inclure la mise à jour des besoins, des spécifications fonctionnelles et des spécifications conceptuelles, pour être compatible avec le code. En outre, les manuels d'utilisation et les guides d'installation et de dépannage sont mis à jour en conséquence.

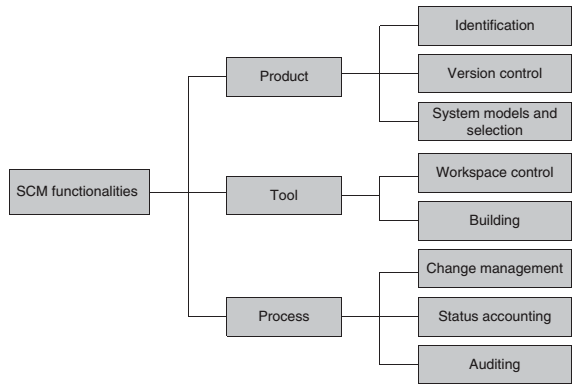
1. **Gestion de la configuration logicielle (SOFTWARE CONFIGURATION MANAGEMENT)**

La gestion des modifications dans les grands systèmes est non triviale. Par conséquent, le concept de  
gestion de la configuration (**GCL**) a été développé pour gérer les changements dans ces grands systèmes. Le but de **GCL** est de gérer et de contrôler les différentes corrections, extensions, et adaptations qui sont appliqués à un système durant son cycle de vie. Dans ce cas du logiciel, la **GCL** gère et contrôle les modifications des éléments de produits tels que les documents, logiciel exécutable, le code source, le matériel et les disques.

La **GCL** assure que les processus de développement sont traçables et systématiques afin que  
tous les changements soient gérés précisément. Par conséquent, le produit est toujours dans un  
état bien défini. Les objectifs de **GCL** sont les suivants :

* Identifier de manière unique chaque version de chaque logiciel à divers points dans le temps.
* Conservez les versions antérieures de documentations et de logiciels.
* Fournir une piste de vérification pour toutes les modifications effectuées.
* Tout au long du cycle de vie du logiciel, de maintenir la traçabilité et l'intégrité des modifications du système.
  1. **Les fonctionnalités de la GCL**

Estublier et al.[[3]](#footnote-3) ont classé les fonctionnalités en trois grands domaines : produit, outil  
et processus. Ensuite, chaque domaine est décomposé en un certain nombre de dimensions techniques comme le montre la figure 6.



**Figure 6. Dimensions techniques des systèmes GCL.**

**Identification**. Les éléments dont les configurations doivent être gérées sont identifiés dans cette fonction. Les éléments identifiés comprennent la spécification, la conception, les documents, les données, les schémas, le code source, le code exécutable, plan de test, script de test, les composants matériels, et les composants de l'environnement de développement du logiciel, à savoir, des compilateurs, des débogueurs, et émulateurs.

**Contrôle de Version**. Pour éviter toute confusion durant le processus de l'évolution d’artefact, un nouveau identifiant est attribué à chaque fois que l'artefact est modifié. Il est important à noter que l'attribution d'un nouvel identifiant pour chaque modification du même artefact peut cacher les relations importantes entre les objets identifiés de manière unique.

Ce type de relation peut être enregistré par le moyen de la fonctionnalité contrôle de version (CV) de la GCL par : **(i)** Interprétation des artefacts du logiciel comme des éléments de configuration et **(ii)** identifier les relations, si il y en a, parmi les éléments de configuration.

**Les modèles du système et la sélection**. La gestion d'un projet fichier par fichier n'est pas très efficace. Il est nécessaire de soutenir les artefacts agrégés, avec des relations entre ces artefacts qui aident à renforcer la cohérence des grands projets. C'est la tâche des modèles de système, qui fournissent le concept de configuration.

L'idée générale de configuration soulève le besoin de permettre aux utilisateurs d'avoir un accès sélectif à des parties et des versions de ces artefacts agrégés.

**Espace de travail**. Les utilisateurs effectuent les tâches habituelles de l'édition et de la gestion de leurs artefacts. Un tel environnement qui permet au mainteneur de faire et tester des changements de façon isolée est appelé espace de travail.

**Construction**. L'efficacité est une exigence clé des systèmes GCL afin que les développeurs puissent construire rapidement un fichier exécutable à partir des fichiers source versionnés. Une deuxième condition du système de GCL est qu'il doit permettre la construction d'anciennes versions du système pour la récupération, les tests, ou la maintenance.

**Gestion de modification**. Au début, les RMs ont été gérées sous forme papier. Cependant, actuellement les RMs sont enregistrées dans le référentiel du GCL et sont liées aux modifications effectuées, en plus d'être automatisées.

**Le bilan d’état**. Le principal objectif de la comptabilité de l'état est de : (i) conserver les enregistrements officiels de la configuration déjà existants et (ii) de produire des rapports périodiques sur l'état des configurations. Ces enregistrements conservent un historique des RMs, y compris celles approuvées et celles rejetées. Un historique de RM comprend les réponses aux questions suivantes :

* Pourquoi les modifications sont-elles apportées ?
* Quand les modifications ont elles été apportées ?
* Qui a effectué les modifications ?
* Quelles modifications ont été faites ?

Le bilan d'état est utile pour communiquer des détails importants sur le projet et sur les éléments de configuration aux parties prenantes du projet.

Un exemple du bilan d'état comprend le nombre de RM par élément de configuration et le temps moyen nécessaire pour implémenter cette RM.

**Audit**. Un système de GCL se comporte comme une archive consultable de tous les évènements qui  
se sont déroulés dans le passé. Par le biais de l'audit, l'organisation maintient l'intégrité  
des références et les mises à jour des configurations de tous les produits. Deux types d’audit sont effectués avant la sortie d’un logiciel : l’audit de la configuration fonctionnelle et  
l’audit de la configuration physique. Le premier détermine si oui ou non le logiciel  
satisfait à la spécification des exigences de l'utilisateur et la spécification des exigences du système.  
D'autre part, ce dernier vérifie si les documents de référence et de conception représentent avec précision le logiciel.

**4.2 Processus de gestion de configuration logicielle**

Par le biais d'un plan, une configuration de base est établie. L’établissement de configuration de base est le processus par lequel un ensemble d’éléments configurables deviennent disponibles publiquement dans un emplacement standard, tel dans un référentiel, aux personnes qui sont autorisées à l'utiliser. Suite à l'identification de la configuration initiale, un processus de contrôle de configuration, comme illustré sur la Figure 7, est invoqué. La Figure 7 montre les trois principales activités de mise en œuvre de GCL : planification, élaboration de configuration de base, et contrôle de configuration.

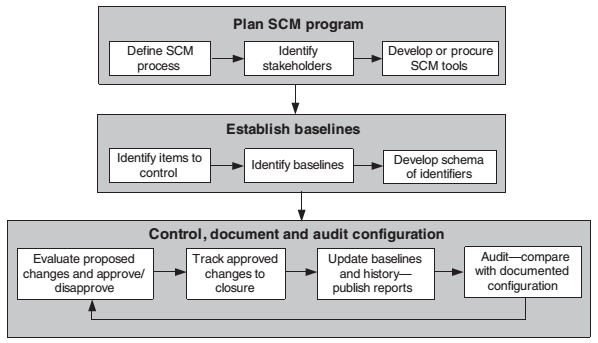


Figure 7. Processus d’implémentation d’un système de GCL.

**Planification**. La planification est débutée avec deux activités : (i) définir le processus GCL  
et (ii) établir des procédures pour contrôler et documenter le changement. Une étape clé pendant la planification est l'identification des parties prenantes. Les parties prenantes dans un système GCL sont les mainteneurs, ingénieurs de développement, les ingénieurs support de test, les auditeurs de l'assurance qualité, les utilisateurs et les gestionnaires. Divers outils CGL sont utilisés pour maintenir l'historique de configuration et faciliter le flux du processus GCL. Des exemples de tels outils sont le système de version concurrente (concurrent version system -- CVS) et ClearCase.

**Établissement des bases**. Une fois un plan de programme GCL est mis en place, la prochaine étape  
d’implémentation effective d’une GCL consiste à identifier les éléments qui font l'objet d’un contrôle de configuration. Quelques exemples de ces éléments sont le code, les données et les documents. Une fois les éléments de configuration identifiés, une bibliothèque de référence du logiciel est créée pour rendre l’ensemble des éléments configurables disponibles publiquement. La bibliothèque, appelé référentiel, est le cœur du système GCL.

**Contrôle, documentation et audit**. Après avoir établi une base de référence, les intervenants spécifiés dans la planification du GCL examinent et évaluent toutes les modifications apportées à la configuration. Après leurs évaluations, les approbations et désapprobations sont documentés. Les modifications approuvées sont suivies jusqu'à ce qu'elles soient vérifiées.

Ensuite, la base de référence appropriée est révisée en conjonction avec tous les documents pertinents, et les rapports sont ensuite générés. À intervalles réguliers, les enregistrements et les produits sont auditionnés pour vérifier que :

* il y a une correspondance acceptable entre la configuration documentée et la   
  configuration réelle ;
* la configuration est conforme aux exigences du projet ; et
* la documentation de toutes les activités relatives aux changements est complète et à jour.

Les trois étapes du cycle, à savoir, le contrôle, la documentation et l'audit, sont  
exécutée à plusieurs reprises tout au long de la durée de vie du projet

1. **Workflow de RM**

Une RM est un véhicule pour l'enregistrement d'informations sur un défaut du système, une amélioration demandée, ou l'amélioration de la qualité.

Un modèle formel est décrit pour représenter les RMs pour l'analyse et l'examination. Le cycle de vie d'une RM est illustré par la figure 8, au moyen d'un diagramme état-transition. Chaque état représente une étape distincte dans le cycle de vie d'une RM.

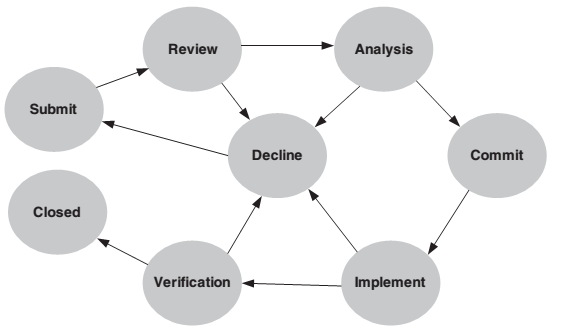


Figure 8. Diagramme Etat-Transition d’une RM.

Le modèle montre l'évolution d'une RM via les principaux états suivants : Soumission, Examen, analyse, engagement, Implémentation, vérification, et fermeture. Les RMs peuvent être stockées dans une base de données. Plus tard, afin de pister et rapporter l'état de la RM, les requêtes sont générées.

**Etat de soumission (submit).** Cet état est l'état initial d'une RM nouvellement soumise. Quand une nouvelle RM est déposée, les champs suivants, décrits dans le tableau 1, sont instanciés :  
***change\_request\_id***

***priority***

***description***

***maintenance\_type***

***component***

***note***

***product***

***customer***

Sur la base du niveau de priorité d'une RM, son état transite de submit à review.

**Etat d’examination (review)**. Généralement, un gestionnaire de marketing traite la RM dans l’état d’examination en coordonnant les activités suivantes :

* Il est possible que la RM nouvellement générée soit une copie d'une RM existante.
* Réévaluer le type de la maintenance de la RM initialement estimé par le demandeur,
* Déterminer le niveau de sévérité de la RM : normal et critique.

En résumé, les champs suivants sont mis à jour dans l'état d’examination :

***priority***

***severity***

***maintenance\_type***

***decline\_note***

***software\_release***

***marketing\_justification***

***description and note***

**Etat d’analyse (analysis).** Dans cet état, une analyse d’impact est conduite pour comprendre la RM et estimer le temps nécessaire à son implémentation. S’il est décidé qu’il n’est pas possible d’implémenter la RM alors le prochain état de la RM sera le refus (decline) sinon la RM passe à l’état engagement (commit). Dans cet état les champs suivants sont mis à jour :

***component***

***problem\_origin***

***impacts***

***time\_to\_implement***

***attachment***

***functional\_spec\_title***

***functional\_spec\_name***

***functional\_spec\_version***

***eng\_assigned***

**Etat d’engagement (commit).** (i) l'équipe d'ingénierie commence par modifier la documentation du composant logiciel, à savoir, des données et des diagrammes de contrôle de flux ; (ii) le personnel de test révise la RM et les spécifications fonctionnelles associées pour s’assurer que le RM est testable ; (iii) le personnel de test développe de nouveaux cas de test pour la RM; et (iv) le personnel de test sélectionne les tests de régression. « Committed\_release » est le seul champ mis à jour dans le Commit state.

**Etat d’implémentation**. Un certain nombre de différents scénarios peut se produire dans cet état comme suit :

* LA RM peut être refusée si son implémentation est infaisable.
* Si la RM est irréalisable dans sa forme actuelle, le directeur de l'ingénierie logicielle  
  doit fournir une explication,
* Si la RM ou sa version modifiée est faisable, le groupe de génie logiciel écrit le  
  code et effectue des tests unitaires.

Dans cet état les champs suivants sont modifiés :

***decline\_note***

***ec\_number***

***attachment***

***resolution***

**Etat de vérification**. Pour attribuer un verdict de test, la vérification peut être effectuée par une ou plusieurs méthodes : la démonstration, l'analyse, l'inspection et les tests. Si la vérification est effectuée par des tests, alors le logiciel est exécuté avec un ensemble de tests. L’inspection Signifie la révision du code pour détecter les défauts. L'analyse est effectuée à l'aide d'outils statistiques et / ou mathématiques. La Démonstration implique la vérification du système en mode opérationnelle. Un état de la vérification est fourni en termes de degré de conformité du système modifié par rapport à la RM : non-conformité, conformité partielle ou conformité totale.

Dans cet état les champs suivants sont modifiés :

***decline\_note***

***ec\_number***

***attachment***

***verification\_method***

***verification\_status***

***compliance***

***tc\_id***

***tc\_results***

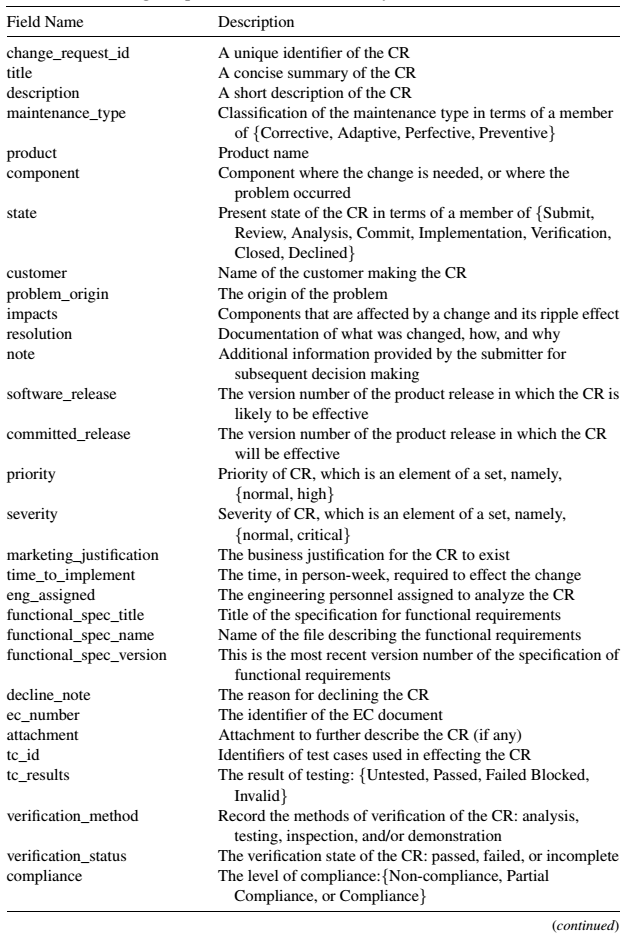
***defect\_id***

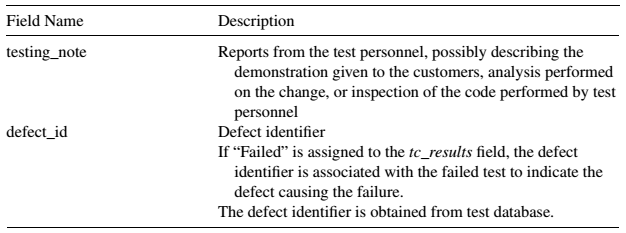
***testing\_note***

**État de fermeture**. Après avoir vérifié que la RM a été incorporée dans le logiciel, la RM est déplacé de l’état de vérification à l’état de fermeture.

**Etat de refus (Decline).** A cause d’une ou plusieurs des raisons suivantes, une RM peut se trouver dans cet état :

* En raison de l'impact commercial insuffisant de la RM, le département marketing  
  décide de rejeter la RM.
* Il est techniquement impossible d’implémenter la RM.
* Le manager de test ( sustaining test manager) conclut que les modifications apportées au logiciel pour satisfaire la RM ne pourront pas être vérifiée de manière satisfaisante. Une explication est fournie dans la forme d'un numéro CE.





1. V. R. Basili. 1990. Viewing maintenance as reuse-oriented software development.IEEE

   Software, January, 19–25. [↑](#footnote-ref-1)
2. V. T. Rajlich and K. H. Bennett. 2000. A staged model for the software life cycle.IEEE

   Computer, July, 2–8. [↑](#footnote-ref-2)
3. J. Estublier, D. Leblang, A. V. Hock, R. Conradi, G. Clemm, W. Tichy, and D. WiborgWeber. 2005. Impact of software engineering research on the practice of software configuration management. ACM Transactions on Software Engineering and Methodology, October, 383–430. [↑](#footnote-ref-3)