Continuous Learning with the Sandwich of Happiness and Result Planning

Theo Theunissen
theo.theunissen@han.nl
HAN University of Applied Sciences, Department of ICT
Arnhem, the Netherlands

Stijn Hoppenbrouwers
stijn.hoppenbrouwers@han.nl
HAN University of Applied Sciences, Department of ICT
Arnhem, the Netherlands
Radboud University, Institute for Computing and Information Sciences
Nijmegen, the Netherlands

Sietse Overbeek
s.j.overbeek@uu.nl
Utrecht University, Department of Information and Computing Sciences
Utrecht, the Netherlands

ABSTRACT

With an increase in fast time-to-market and keeping up with fast mandatory legal changes, we observe a demand for continuous software development which is reflected by the emergence of Lean, Agile, and DevOps approaches. At the same time, we observe the phenomenon of lifelong learning that is both manifest and propagated by government, industry, and education. We introduce two patterns that match these two phenomena: the Sandwich of Happiness and Result Planning. Together, these patterns support learning for students in an educational setting and continuous learning for professionals in industry, especially in the context of Continuous Software Development.

CCS CONCEPTS

• **Applied computing → Education**.

KEYWORDS

Agile, Documentation, Continuous Learning, Continuous Software Development, Result Planning, Sandwich of Happiness

ACM Reference Format:
Theo Theunissen, Stijn Hoppenbrouwers, and Sietse Overbeek. 2021. Continuous Learning with the Sandwich of Happiness and Result Planning. In *European Conference on Pattern Languages of Programs (EuroPLoP'21), July 7–11, 2021, Graz, Austria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3489449.3489974

1 INTRODUCTION

Modern software development comes with continuous changes, reflected in agile methods and through demands for a fast time-to-market. At the same time, technology innovates in a fast pace and to keep up with these changes, lifelong learning is fundamental. In this paper we introduce two patterns to adapt to continuous change in software development by keeping up with lifelong learning. In a continuously changing environment, with new technology, engineering solutions and processes, lifelong learning is required to match these changes. The objectives of this paper are to present two patterns that can support this continuous change, and to provide insights for better comprehension and application of these patterns.

Nowadays, students as well as professionals have to frequently adapt to new situations driven by an ever-changing world: continuous improvements [10], fast time to market (TTM) [6] and lifelong learning [9]. This applies specifically to software engineering that is in a continuous evolution because of innovation [18]. Employees are required to adapt to an ever-changing world [12].

We will now introduce continuous software development, lifelong learning and the relation between these two.

1.1 Continuous Software Development

The primary context for the presented patterns concerns continuously changing software, including technologies, processes, development and maintenance as can be found in industry. The relation between lifelong learning and continuous software development is that continuous software development often includes several iterations of the software product. For example, some parts, such as data, business logic, and the user interface may persist and other parts, such as a database, the programming language, and a software framework such as jQuery may be replaced [16]. We refer to this continuously changing state as Continuous Software Development (CSD). The main characteristics of CSD [17] are that:

(1) it covers values, principles, practices, tools, procedures and processes from Lean [14], Agile [4], and DevOps [3];
(2) it embraces activities from the whole life cycle of a software product, i.e. from concept to end-of-life. In addition to Agile and Lean software development, it includes maintenance activities. In addition to DevOps, it includes continuous architecting activities. Continuous Integration and Continuous Deployment (CI/CD) are part of it;
(3) it considers the continuously changing state of the software product and progress, such as progressive insights (e.g., regarding process, design, implementation), changes in contextual factors, new features or requirements, bug fixes, or other unforeseen factors;
(4) it distributes information about software development across multiple tools. There is no central repository for all relevant information. Because of high demands for fast time-to-market, process automation comes with a wide range of

EuroPLoP'21, July 7–11, 2021, Graz, Austria

ACM ISBN 978-1-4503-8997-6/21/07…$15.00
https://doi.org/10.1145/3489449.3489974

tooling for designing, developing, testing, deployment and monitoring.

Second, lifelong learning applies to industry practitioners who are confronted with new concepts, frameworks, technologies and communities. Furthermore, lifelong learning includes, besides professional development, also personal, relational and organizational development.

1.2 Lifelong Learning

Learning refers to knowledge, skills, and attitude [2]. Of these three, knowledge is relatively easy to assess by means of tests. Someone taking a test can get the maximum score by answering all answers correctly. Skills are less easy to test unambiguously, and scores typically fall in broad categories like completed or failed, sufficient or success. The hardest and and most ambiguously to assess is attitude, because it is based on culture, beliefs, and values that cause mentors and professionals to judge differently. The Sandwich of Happiness refers to all three aspects of learning. Students and professionals are primarily stimulated to reflect on knowledge, skills, and attitude. Furthermore, any comfort or annoyance that influences the results or process can be reflected upon. Individuals can become better professionals by taking into account personal habits. As we live in an ever-changing world, and especially for engineers that live by creating innovative artifacts, both students and professionals have to keep up with these evolving creations [15]. This is reflected in CSD which implies continuous changes during the lifetime of software as a product. Unskilled people in particular tend to overestimate their own competences and underestimate problems. This is the so-called Dunning-Kruger effect [13].

A second aspect of lifelong learning is the continuous balance between skills and challenges, an increasing number of skills and challenges are in a balanced flow [8]. This is depicted in Figure 1. This flow starts from the first learning experience, includes learning years at school and at the university and continues for practitioners in the industry. The balance refers to related emotions staying between boredom and anxiety.

1.3 Patterns

The objective of this paper is to provide practical guidance for students and practitioners for continuous learning in CSD. It aims to be operationalized by a handout with practical usage and a theoretical background. To support this objective, in this section a brief conceptualization is presented for better understanding of the approach and patterns. In general, **a pattern is defined by a proven solution for a problem in a recurrent context**.

1.3.1 Alexandrian patterns. The 'pattern' concept was coined by Christopher Alexander in the context of problems when designing towns, buildings or windows. For Alexander, a pattern is defined by the following characteristics [1].

(1) A **picture** which shows an archetypal example of the pattern.
(2) An introduction that sets the **context** of the problem.
(3) The **problem** in one or two sentences.
(4) The **body of the problem**, including empirical background, evidence, and range of manifestations.
(5) The **solution** is always stated in the form of an instruction.
(6) A **diagram** of the solution that indicates the main components.
(7) A **relation** of the pattern to other patterns in a pattern language.

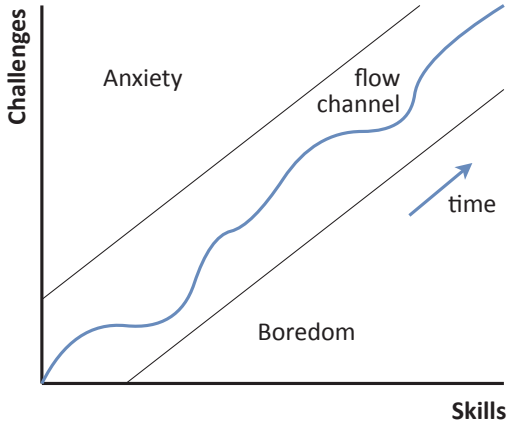

Figure 1: Flow is a channel between Boredom and Anxiety based on a balance between Skills and Challenges [8].

1.3.2 Coplien patterns. Coplien introduced this pattern [7] and was a co-founding member of the Hillside pattern community.

(1) The **name** of the pattern.
(2) The **alias** refers to alternate names or 'also known as'.
(3) The **problem** describes the goals and objectives to achieve.
(4) The **context** describes the preconditions or applicability.
(5) The **forces** section lists the constraints, and interactions and trade-offs between the constraints.
(6) The **solution** section instructs how to construct the product.
(7) The **example** section illustrates a specific, easy to use, application of the pattern.
(8) The **resulting context** section describes the consequences, post-conditions and side-effects of the pattern.
(9) The **rationale** section justifies and explains the steps and rules how the trade-offs are applied.
(10) The **known uses** section demonstrates the accuracy and firmness of the pattern for recurring problems.
(11) The **related patterns** section relates to other patterns with common forces, initial or resulting context.

In this paper, we select the characteristics that are related to taking decisions: context, problem, forces, solution and consequences.

2 OUR TWO PATTERNS

The patterns Sandwich of Happiness (Section 2.1) and Result Planning (Section 2.2) have been used for several years at HAN UAS to support students working on school projects and in internships.

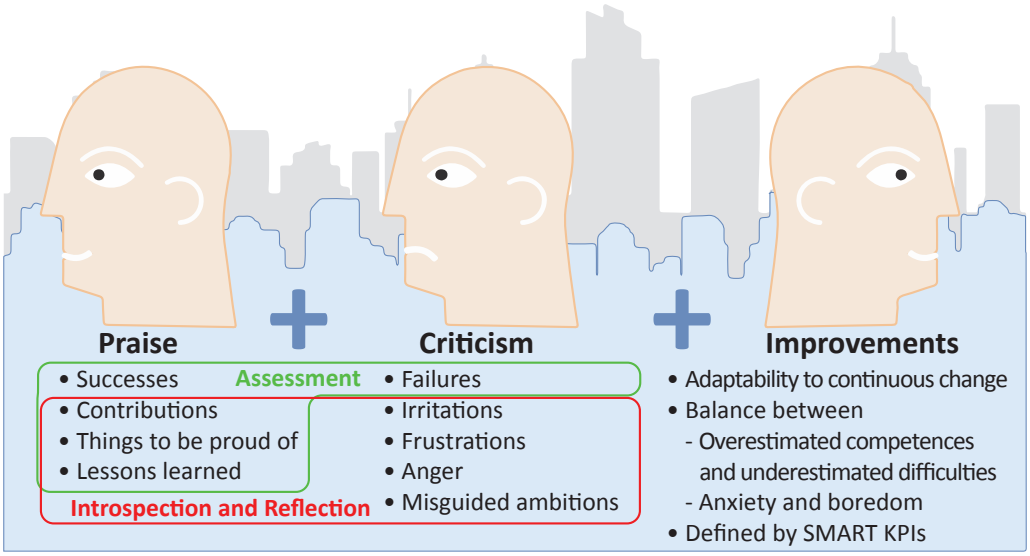

Figure 2: With the Sandwich of Happiness, assessors look backward for praise and constructive criticism. Assessors look forward to improvements. The bitter pill is in the middle, sandwiched between a positive evaluation and a positive outlook. The assessment can be performed by a coach or as a self-assessment.

2.1 Pattern: The Sandwich of Happiness (SoH)

Continuous learning by answering three questions: what was good, what was bad, what could be better?

2.1.1 Context. An SoH is prepared several times during a process (e.g., in an iteration, at mid-term, during a semester or regular coaching sessions) because an evaluation only at the end of the project, without the possibility of improving during a project, would be rather pointless. Typically, it is hard for students and professionals to be critical of someone's performance as it may easily turn into non-constructive social dynamics that are hard to escape. The Sandwich of Happiness aims at eliciting constructive criticism by looking back for positive and negative results as well as looking forward for improvement.

2.1.2 Problem. Learning, improving, and adapting to change is problematic because it involves cognitive aspects, performance, and attitude [2]. It is a continuous process [9] that starts at university, goes on in professional life and extends to retirement, in short: it takes a lifetime.

Developments in engineering are a continuous process of improvements, evolutions, and revolutions. This relates primarily to technology but extends to processes and societies where developments are used. A **mind shift** is required to keep up with or stay ahead of these developments.

Scrum, a widely used software development process, has **no mechanism for emotional hygiene** where developers can solve personal issues or build on personal characteristics.

There are demands from the market to stay ahead of **competition**. In an ever faster moving world with competition from every website, high demands for fast time to market require continuous adapting to change.

There are **Legal requirements** (e.g. GDPR). Demands for change might come from legislation to implement the requirements that might be unforeseen when designing a solution.

The team needs to **repair design flaws and bug fixes** to keep the product functioning as intended.

Feature requests from end users or customers require developers to continuously learn to understand end users and customers, to translate requirements into executable and manageable tasks.

There are **Progressive insights**. Once a whiteboard sketch is drawn in a team or a specification is written down, almost immediately improvements to the sketch or specification emerge.

Refactoring takes place: fixing code that is not broken but improving performance, readability or maintenance requires changes to the software product.

Peaceful and quiet delivery of requirements in a waterfall process is in strong contrast with high demands for continuous change.

2.1.3 Forces. Following Kruger and Dunning [13], **incompetence** makes it hard to reflect on inadequate performances. More in general, people overestimate their competences and underestimate the difficulties at hand.

Csikszentmihalyi et al. [8] define a **balance between anxiety and boredom** that flows in time. See Figure 1.

It is more easy, and perhaps more kind, to give positive feedback, such as in tips and tops, than it is to **express irritations and frustrations**. Irritations, frustrations and misguided ambitions are not candidate for improvements and should be avoided.

Keep up with **realistic ambitions**. With all the failures, missed deadlines, overestimated competences and underestimated assignments, it is sometimes hard to be **proud of the positive achievements** such as contributions and lessons learned.

2.1.4 Solution. The resolving principles that are taken into account are depicted in Figure 2. The Sandwich of Happiness can be instructed as "three questions with three answers". Two questions look backwards, the third looks forward. For all three answers, use introspection and reflection to assess the results and the course for the results. With introspection, the person contemplates on his or her own actions and motives whereas reflection concerns other persons and context. The most specific objects for introspection and reflection are results, but tasks, processes and emotions can also be included.

(1) *Looking backward*, start with writing down **positive results** such as your achievements and successes. Furthermore, reflect on other positive results such as contributions, and outcomes to be proud of. Even lessons learned in case of failures are positive.
(2) Also *looking backward*, write down your **negative results** such as failures, and reflect on the causes, course and realization of how the objectives were not met. Irritations emerge in situations that do not contribute to productivity or happiness. Frustration relate to situations that someone can not change to improve. Ambitions that do not match with capabilities should therefore be avoided. Note that it takes some practice to be critical of ones own performance and also that it takes some exercise to be professionally 'cool' in dealing with negative results.
(3) *Looking forward*: based on your results, the causes of the results, and your ambitions, define result goals for the next iteration. Typically, looking forward includes commitment to **positive results**. Looking forward includes contemplation and reflection on adapting to change, and keeping a balance between overestimated competences and underestimated difficulties, thereby finding a flow between anxiety and boredom.

2.1.5 Consequences. The objective for the Sandwich of Happinesss is a continuous improvement in an ever changing environment, such as in CSD. The time it takes to write down a Sandwich of Happiness is about 15-30 minutes. The text is concise, to the point, relevant, easy to read and makes it possible for a coach or mentor to do quick assessments on the results and reflections.

Because the Sandwich of Happiness is liberal in trade-offs between the forces, students can reflect on anything that influences the results. This includes who you are, what you want, your capabilities, barriers and limitations.

Furthermore, improvement does not imply an ever continuous upward path. It also includes -like a bull and bear market- downfalls, dips and drops. However, these downfalls as well as peaks lead to flowering advancements and evolution.

2.1.6 Examples. In Appendix 2.2.6, an example is presented of the template for the Sandwich of Happiness. The description that goes with the template is presented below.

The size of a typical Sandwich of Happiness is more than 1/2 A4 page but does not exceed the size of a full A4.

The Sandwich of Happiness is partly –under the first two questions– a thoughtful reflection concerning last week's accountability of the planning and results. The third question is looking forward to where you define improvements.

(1) Looking backward, what was successful, what are you proud of, what have you contributed for others and to results, what was an example to others, what should be continued, what was positive?
(2) Looking backward, what went wrong, what were irritations and frustrations, what was a misguided ambition, what must not be tried (happen) again, what should be stopped; in short: what was negative?
(3) Based on 1) en 2), what will you do next week, what will you start with? Make it specific, for example SMART.

2.1.7 Related Patterns. There is a Feedback Pattern pattern described by Bergin et al. [5], published in a collection of education patterns in 2012. The pattern describes two types of positive feedback and improvements in the middle for students to remain confident in their understanding. We consider this pattern to lack structure; it could be better thought through. Furthermore, we consider that while he pattern described by Bergin et al. [5] focuses on students only, the pattern is also applicable to practitioners in industry in context of lifelong learning. Finally, the Feedback Pattern is part of Feedback Patterns in general. The feedback does not make a distinction between introspection (only take your own actions and motivations into account) or reflection (also take the context and others into account).

2.2 Pattern: Result Planning

Continuous improvement by being strict on planning and reporting of verifiable results and relaxed on accountability.

2.2.1 Context. This pattern is applied when working in processes that have the characteristics of CSD. It does not depend on the evaluation loop, feedback, phases, sprints, or planning/feedback loops. One of the values from agile is that working software is more valued than comprehensive documentation. Beck [4] assumes that all time is spent on developing software. This value ignores other tasks such as, for instance, –for both students and professionals– planning, meetings, administration, presentations, and –for professionals– acquisition and traveling.

It is hard to estimate the time spent on tasks other than development, for example by students who cannot lean from experience, or by professionals who have to deal with managers pushing fast time-to-market features. Students and professionals have tasks that do not directly contribute to productivity, such as all kinds of meetings or administration. Professionals might have additional tasks such as acquisition, training, and traveling. On top of that, switching context between tasks takes time to accommodate. When accountability for productivity and time is missing because of "unproductive" tasks, the time spent might become marked as unprofitable.

2.2.2 Problem. The problem with measuring results is that:

(1) It is hard to define **verifiable objectives**. For instance in scrum, user stories and tasks can be defined. However, the formulation of definitions of done for user stories or acceptance criteria for tasks is often sloppy, if they are defined at all.

(2) Results are typically expressed as **efforts**. Examples of efforts are activities such as 'reading', 'meeting', 'designing', and 'programming'.
(3) No rules for **timing**. In scrum, there are no explicit rules how long tasks or iterations should take.
(4) People may fear **failing** and taking **blame** when results are not accomplished. Usually, working software is not questioned but failing software can have valuable learning results.

2.2.3 Forces.
The following forces are taken into account:
Method bonanza. There are many approaches in CSD that define how to plan tasks or results, such as Lean, Agile and DevOps. Students are taught textbook definitions but professionals often use best practices that leave out parts or introduce processes that fit for their purpose.
Death by planning. Popular tools such as Jira support the planning of user stories and tasks. However, extensive upfront contemplation of definitions of done and acceptance criteria keep students and professionals away from committing to results.
Efforts instead of results. Results can range from a strict focus on SMART key performance indicators (KPI) on the one hand to just mentioning efforts on the other. It is both hard to *define* SMART results as well as to *achieve* results when defining efforts.
Programming is fun. Developers love the act of coding and tend to avoid diverting activities such as planning, documenting or process-oriented meetings.
Deviating from textbook definitions of methods. Students are taught textbook definitions of processes such as scrum, including all ceremonies. Practitioners from the industry hardly follow the paradigm from these textbook definitions but only use what is applicable in their professional context. It can be hard to grasp which aspects of methods are strictly necessary and which are optional.
Time estimation can be really difficult. Jones [11] shows that there is no scientific literature on estimating time for tasks. Furthermore, the combination of experience, (lack of) control of process and context, mitigation of risks and other potential threats, and creative characteristics of inventing solutions make it hard to give a reliable estimation for time.
When being held **accountable**, students and practitioners tend to find explanations that point to others or to external causes for failing to achieve results. The aim, however, is not to deal blame but to improve, and a full assessment can best accomplish this for the student or professional. For instance, one cannot blame another for being late delivering a requirement, but one *should* address the other for being late.

2.2.4 Solution. The resolving principles to balance the forces are related to delivering verifiable and transparent results. The forces for the Result Planning tend to involve individual accountability rather than team accountability.

The template for the Result Planning, as shown in Appendix 2.2.6 has columns for team member, results, achievements and, optionally, notes. The result statement refers to being concise and informative in single line sentences. Only the main result should be mentioned, so when there are more results for half a day, then the most significant result should be mentioned. Again, efforts are not allowed since these are always 'successful', i.e./ the effort taking place means success. For example, sometimes, students (as well as professionals) have things to do other than work, such as medical or driving lessons. However, it is mandatory in such cases that they find alternative time slot to do the work, even if this is in the evening or weekends. The achievement column is small and, when a result is achieved, contains a commit hash or link to some live document. In case of failure, there is only space for the core message of failure. Long descriptions are not allowed, to hide incompetence.

The time horizon is one week, and often students cannot oversee more than three days with tasks stretching in time. The template is used within a team and results are planned together.

The next week, each team member has taken two actions: 1) fill in the achievement column and 2) define new results. The result statements and achievements are, together with the Sandwich of Happiness, discussed with the teacher, coach or team lead.

2.2.5 Consequences.
SMART(ish) results. Results are defined SMART(ish) so they can be measured. Activities like 'reading', 'programming', 'learning', 'attending (meeting/class)' and so on are by definition successful and stretch into the available time. Results like 'lists of concepts or frameworks', 'working software', 'asked questions, got the answers', 'updated Result Planning' and so on are examples of activities that can be measured.
Measure velocity. It is easier to check whether you make progress when main results are defined for half a day.
Smaller iterations. It is easier to both define moonshot goals, as well as realistic objectives, than it is to split the task up in smaller tasks that take risk, dependencies, and resources into account. Also, there is no scientific literature that prescribes an optimal duration for an iteration.
More significant results. Over time, students learn to define a more significant, relevant and contributing result in half a day. The period of 4 hours remains the same, but results are increasing. (In the beginning result are often too ambitious and expectations can not be met.)
Achieving goals makes people happy. Happy customers, happy developers and a happy team are within reach. Suppose a team consist of 5 people who define 2 results per day in an iteration for 1 week. That sums up to 50 results per week! Even when not successful for all projected results, this is impressive and encouraging.
Reliable. When using this pattern, you will become more reliable for yourself, team, teacher and manager because you will be able to make better predictions concerning your knowledge, expertise and how you can use your experience.
Continuous learning and improvements. In CSD, this helps in adapting to change due to continuously changing demands from stakeholders or fast time-to-market, or changing constraints, risks or contexts. Additionally, learning experiences are higher in small iterations than in large iterations.

2.2.6 Examples. An example of a Result planning is shown in Appendix 2.2.6. The description that goes with the template is shown below.

Usage Per team member, you define a result per half working day. For a team with five people, you have 50 results per week or 100 results if your iteration takes two weeks.

Objectives What result do you want to achieve at the end of the day? Mention results that can be demonstrated, verified, or tested. Do not describe mere activities, such as 'reading', 'meetings', 'designing', 'programming' etc. but verifiable results. Typical results are a list of concepts, decisions, design or working software.

Achieved results Make sure that you can demonstrate verifiable results and these results relate to the planning:

(1) Make links to GitHub for comments and code.
(2) Make links to, e.g., Dropbox, Google Drive for design, documents and books.
(3) Make links to websites.
(4) Reflect on achieved results in your individual Sandwich of Happiness.

What went wrong? It is OK if an objective is not achieved. Do not spend more time than planned. Ask your coach, instructor or an expert for support.
Mention the reason for failure. There are a lot of valid reasons why you could not achieve the objectives. Be fair and try to be honest to yourself and your team. You will learn the most if you are critical towards your results and process.

The objectives of this pattern are to:

(1) Use your experiences to learn (not to deal blame).
(2) Every week you will have better estimations and will better close the gap between planning and results.
(3) Make increasingly reliable predictions involving your knowledge, expertise and experience.

ACKNOWLEDGMENTS

We want to thank our shepherd Christopher Preschern for shearing our sheep.

REFERENCES

[1] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford university press, Berkeley, CA.
[2] Liesbeth K.J. Baartman and Elly de Bruijn. 2011. Integrating knowledge, skills and attitudes: Conceptualising learning processes towards vocational competence. *Educational Research Review* 6, 2 (2011), 125–134. https://doi.org/10.1016/j.edurev.2011.03.001
[3] Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A Software Architect's Perspective.* Addison-Wesley Professional, New York, NY. 352 pages. https://doi.org/10.1017/CBO9781107415324.004 arXiv:25246403
[4] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. Manifesto for Agile Software Development Twelve Principles of Agile Software. http://www.agilemanifesto.org
[5] Joseph Bergin, Jutta Eckstein, Markus Volter, Marianna Sipos, Eugene Wallingford, Klaus Marquardt, Jane Chandler, Helen Sharp, and Mary Lynn Manns. 2012. *Pedagogical Patterns: Advice for Educators.* Joseph Bergin Software Tools, Pleasantville, NY.
[6] Morris A. Cohen, Jehoshua Eliasberg, and Teck-Hua Ho. 1996. New Product Development: The Performance and Time-to-Market Tradeoff. *Management Science* 42, 2 (1996), 173–186. https://doi.org/10.1287/mnsc.42.2.173
[7] James O. Coplien. 2021. Canonical Form. http://wiki.c2.com/?CanonicalForm
[8] Mihaly Csikszentmihalyi, Sami Abuhamdeh, and Jeanne Nakamura. 2014. *Flow.* Springer, London, UK.
[9] John 1949 Field. 2000. *Lifelong Learning and the New Educational Order.* Trentham, Stoke-on-Trent, UK.
[10] Brian Fitzgerald and Klaas-Jan Stol. 2014. Continuous Software Engineering and beyond: Trends and Challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering (RCoSE 2014)*, Matthias Tichy, Jan Bosch, Michael Goedicke, and Magnus Larsson (Eds.). Association for Computing Machinery, New York, NY, USA, 1–9. https://doi.org/10.1145/2593812.2593813
[11] Derek Jones. 2021-01-17. *Software Effort Estimation Is Mostly Fake Research.* The Shape of Code. http://shape-of-code.coding-guidelines.com/2021/01/17/software-effort-estimation-is-mostly-fake-research/
[12] Philippe Kruchten. 2015. Lifelong learning for lifelong employment. *IEEE Software* 32, 4 (2015), 85–87.
[13] Justin Kruger and David Dunning. 1999. Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of personality and social psychology* 77, 6 (1999), 1121.
[14] Mary Poppendieck and Tom Poppendieck. 2003. *Lean Software Development: An Agile Toolkit.* Addison-Wesley, Boston, Mass.
[15] Donald A. Schön. 1984. *The Reflective Practitioner.* Basic Books, Inc., New York, NY, USA. https://doi.org/10.4324/9780203963371
[16] Theo Theunissen and Uwe van Heesch. 2016. The Disappearance of Technical Specifications in Web and Mobile Applications. In *Software Architecture*, B. Tekinerdogan and U. Zdun (Eds.). Software Architecture. ECSA 2016. Lecture Notes in Computer Science, Vol. 9839. Springer International Publishing, Springer, Cham, 265–273. https://doi.org/10.1007/978-3-319-48992-6_20
[17] Theo Theunissen and Uwe Van Heesch. 2017. Specification in Continuous Software Development. In *Proceedings of the 22nd European Conference on Pattern Languages of Programs (EuroPLoP '17)*, Association for Computing Machinery (Ed.). ACM, Association for Computing Machinery, New York, NY, USA, 1–19. https://doi.org/10.1145/3147704.3147709
[18] Lorna Uden and Alan Dix. 2004. Lifelong learning for software engineers. *International Journal of Continuing Engineering Education and Life Long Learning* 14, 1-2 (2004), 101–110.

APPENDIX 1: TEMPLATE FOR SANDWICH OF HAPPINESS

Sandwich of Happiness (theotheunissen.nl/happiness, goo.gl/YyTYj5)

Week of year		**Date** (on Monday)	

The Sandwich of Happiness is partly –in the first two questions– a thoughtful reflection of last week's accountability of the planning and results. The third question is looking forward to where you define improvements.

1) What was successful, were you proud of, have you contributed to others, was an example for others, was positive?
2) What went wrong, where irritations and frustrations, was a misguided ambition, must not be tried (happen) again; in short: what was negative?
3) Based on 1) en 2), what will you do next week? Make it SMART.

1.) <Your reflection on positive outcomes; **Replace YELLOW with your text**>
2.) <Your reflection on adverse outcomes, do *not* mention improvements>
3.) <Fill in below>

Specific	
Measurable	
Acceptable	
Relevant	
Time-bound	

These reflections and improvements take more than half a page and not more than a full page.

Author	_____
Student number	_____

Figure 3: Template for Results planning. See also http://theotheunissen.nl/happiness

APPENDIX 2: TEMPLATE FOR RESULT PLANNING

Group			**Template url on**
Start date			https://goo.gl/ujD5Al
End date			http://theotheunissen.nl/results
	Planning before		**Accountability after**
	Who	**Planned results**	**Achieved results**
Monday morning	Alice	Set up infrastructure	link to document
	Bob	List of concepts and literature	link to document
	Charlie	Login form	link to git commit
	Dave	CSS framework selection	Not successful. Too many options. Smaller step to decide first about an easy framework or advanced framework.
Monday afternoon			
Monday evening			
Tuesday morning			
Tuesday afternoon			
Tuesday evening			
Wednesday morning			
Wednesday afternoon			
Wednesday evening			
Thursday morning			
Thursday afternoon			
Thursday evening			
Friday morning			
Friday afternoon			
Friday evening			
Saturday evening			
Saturday evening			
Saturday evening			
Sunday evening			
Sunday evening			
Sunday evening			

Objectives
What result do you want to achieve at the end of the part of the day? Mention results that can be shown, verified or tested. Do not describe mere activities (like reading, meetings, designing, programming &c)

Examples:
Questions as a result of reading books, visiting websites or viewing videos; interview transcripts; UI wireframes; functional or technical requirements; bug solutions; contributions to other teams or community; change list in code or requirements; infrastructure; documents; functionality.

Group results together, or split results that it fits in a part of a day.

Achieved results
Make sure that you can show visible results and that results are related to the planning.

- Make links to github for comments, code.
- Make links to dropbox, google drive for design, documents and books.
- Make links to websites.
- Reflection on achieved results in your individual Sandwich of Happiness.

What went wrong
It is OK if an objective is not achieved. Do not spend more time then planned. Ask your expert for support.

Mention what the reason was for failure.
There are lots of valid reasons why you could not achieve the objectives.
There can be good reasons for failure or not so good reasons. Be fair and try to be honest to yourself and your team. You will learn the most of you be critical on your results and process.

Reasons for failure can be valid.

The objectives of this method are to:
- Use your experience to learn (not to blame).
- Every week you will have better estimations and will close the gap between planning and result.
- Make reliable predictions about your knowledge, expertise and experience.

Figure 4: Template for Results Planning. See also http://theotheunissen.nl/results