Using Students' Screencasts as an Alternative to Written Submissions

Henrik Bærbak Christensen
Computer Science Department, Aarhus University, Denmark
hbc@cs.au.dk

Abstract

In this paper, we report our experiences on using student produced screencasts as a medium for students to explain and provide overview of their solution to advanced design and programming exercises. In our context, the screencasts have replaced written reports as submissions, and we report both on students' perception on work effort and effectiveness of screencasts as well as teaching assistants' experiences in assessing and marking the screencasts. Our main conclusions are that screencasted submissions is an important tool in the teacher's toolbox for some categories of learning tasks, but there are a number of best practices to follow to gain the full benefits of the approach.

1. Introduction

Guiding and helping students to acquire advanced design, programming and software engineering skills is a daunting task. Central to the constructivist learning paradigm [1] is the students' own learning activities, which in our context means finding appropriate software designs and craft quality implementations of them in a programming language. However, good teaching requires more than just reviewing students' produced code—the process is just as important. To assess that, we have previously asked students to hand in a written report together with their source code; a report that details their ideas and design, and document their process.

A major challenge to instructors is the timely, efficient, and proper feedback on the artefacts that students produce, pointing out strengths and weakness in their work and suggesting alternatives and ways of improvement.

The motivation for the present work came from a wish to make more efficient use of both students' and instructors' precious time, and to ensure a better alignment between process oriented assignments and what students actually do. The central hypothesis was that students should produce *screencasts* (a digital recording/video of the computer screen output enhanced with audio narration) instead of reports. Thus our three central research questions were

- R1: Are screencasts better to convey fulfillment of learning goals compared to written reports?
- R2: Are screencasts more efficient to produce than written reports?
- R3: Are screencasts more efficient to evaluate than written reports for the teaching assistants?

The present paper presents data, experience, and best practices gained from 2014 until 2020 in two courses in nine instances. The data includes students' perceptions recorded through answering both Likert scale questions as well as free text; interviews with teaching assistants, and finally the instructor's impressions.

The main conclusions are that screencasting is a viable and valuable tool for certain types of assignments, notably process-focused, overview-focused, and demonstration-oriented assignments; that they indeed are considered time-saving and thus preferred over written reports by students, but it is less so for the teaching assistants. Another important learning is that efficient use of screencasting requires strong communication and guidance by the instructor to avoid confusion and resistance.

2. Related Work

Screencasts have been widely adopted in teaching in higher education in order to produce material from the teacher to the students. Reasons range from improving access, efficiency, and student engagement [2], to adopting flipped classroom teaching [3] that allows developing a more student-centered approach through "content in advance" by prerecorded screencasts or captured videos. However, the most prominent use is their use to address a larger audience and/or allowing self-pacing, as seen in massive online (MOOC) teaching at edX, Coursera, Udacity and others [4, 5].

URI: https://hdl.handle.net/10125/79440
978-0-9981331-5-7

The aspects of making students produce screencasts is however much less studied. Powell et al. [6] evaluate the effectiveness of students' screencasts as a way to increase student learning outcomes—by advocating screencasts without narration as an alternative to note taking in a introductory programming course. Thus, the produced screencasts were not submitted, they were only used for the student's own use. Schafer [7] presents a study on students producing screencasts of geometric proofs which are then peer-reviewed and discussed in-class to increase student's self-reflection. Mohorovičić [8] describes both teacher and student produced screencasts, but emphasis is on the work flow of screencast creation, and the selection of tools.

Our work supplement and extend research in student produced screencasts in several ways. First, our emphasis is screencasts as the submission of an assignment, and thus basis for assessment and grading, more than a tool for reflection and own learning. Second, we present data from an extended period (7 years), we present observations from both the producers (students) and receivers (teaching assistants), and finally we present concrete advice for teachers wanting to incorporate screencasting in their teaching. We acknowledge that our data is an experience report rather than rigorously validated data: Our focus in our questionnaires and interviews have been to improve teaching in the courses' subject areas rather than providing rigorous insights into screencasting as such. Still, we find our experiences relevant as basis for further work and study, as well as inspiration for teachers.

3. Course Contexts

Our use of screencasts as students' submissions was introduced in 2014 in a bachelor level (2nd year) quarter length (7 week long) course with about 130 students enrolled. Each week is organized with four 45-minute plenary lectures held by the lecturer and three 45-minute classes (20-25 students) focused on the mandatory project (outlined below) staffed by teaching assistants. The course is a second year university level course and learning goals are primarily advanced programming techniques: design patterns, frameworks, testing, test driven development, refactoring, and software engineering and associated tools (JUnit, Ant, Subversion, ...), and to a minor extent aspects of software architecture and systematic testing. The students are required to have passed exams in introductory object-oriented programming and to a minor extend object-oriented design.

Our learning perspective is based on *constructivist* theories and the main learning vehicle in the course is a mandatory project consisting of six *learning iterations* or sprints that each adds functional and design increments that ends up with a configurable framework (in Java) for creating Civilization type games, complete with a graphical user interface. Students are organized in groups of 2–3 persons that work and hand in as a unit. While each iteration requires adding, modifying, and refactoring the group's designs and source code, it also requires academic reflections over the suitability of design choices and software quality assessments. For more detail consult [9, 10].

Up until the 2014 instance of the course, the students were required to hand in two artifacts in each of the six deliveries: a written report answering specific questions about their design and implementation, as well as a zip file of their production and test code. The questions to be answered in the report typically required the students to document designs using UML diagrams as well as include and explain relevant code fragments that express design patterns, JUnit test code, framework hotspots, etc. In iterations that focus on *process*, like e.g. test driven development [11] or refactoring [12], students were required to document and reflect about their process as well as provide "before-and-after" code fragments.

As reported below, the experience gained led us to keep using screencasting in later and revised instances of the course. In 2017 a faculty level decision required revising the course into a full semester length course (14 weeks), keeping the core advanced programming focus and extending it with extra topics as well as going into more detail—putting extra emphasis on build management (migrating to Gradle), version control (migrating to Git, and using a University GitLab repository for code deliveries), as well as introducing clean code principles [13], concurrent programming, as well as distributed programming using the Broker pattern [14]. The original project's six learning iterations (that is six assignments and submissions) were extended to 10, covering the extra topics of the course.

In addition, a single master level course on cloud computing (again, with a strong programming and skill-oriented focus) used screencasting for some of the submissions, as detailed further in Section 7.

4. Screencasts as Assignment Submissions

The initial motivation for trying screencasting was actually to try to lower the workload of the students and the teaching assistants (TAs). Economic considerations at our department urged the teachers to consider how to reduce expenditure on teaching assistants. Therefore we conducted a brainstorm with course TAs from the 2013

instance of the course with that aim. The brainstorm actually brought up a lot of potential changes in the way the exercise classes were conducted and in the way student groups' and TAs' work were organized but we settled on replacing written reports with screencasts as the most promising based on the following observations:

- The TAs' experience was that students focus heavily on the design and implementation effort embodied in the project's learning iterations (which is also aligned with the intentions of the lecturer) which leads to written reports of low quality. Basically, they reported them as "put together in a rush just before hand-in deadline."
- The workload on the students are measured by the department's official evaluation system filled out by the students. Here, the course is in the upper bracket of workload and above the required. It was therefore important to try to find ways that lowered workload on the students while retaining the same learning outcome.
- TAs spend much of their time correcting the group's report and code deliveries, and providing feedback to the groups. This entails reading the (often poorly structured) report and evaluating the code (executing tests and demonstrators, browsing and understanding code structure). The latter is a rather laborious process as the project ends with a large code base in a complex package structure. Screencasts were hypothesized as a way to lower workload on the TAs as the source code and its structure is already shown while the students explain and browse their code in the screencast, and run their test cases or demos. Ideally, the TAs would not need to inspect/run the code at all, if the screencast provides a comprehensive and compelling overview.
- Screencasts were in the assignments required to between 6-15 minutes. This was deemed (much) less time spent by the TA per group than it takes to read a report, get hold of source code, review and execute it.
- Several central learning objectives, such as test driven development and refactoring, are basically processes. Processes unfold over time and are thus poorly documented in written form. Screencasting was deemed a better medium for showing process.
- Previous work [15] had documented that even in courses that focus on test driven development, many students adopt misaligned practices in that they write the tests *after* developing the production code using traditional techniques. This is impossible to spot in a written report or resulting source code, but obvious in a screencast. Thus changing medium will force students to adopt the proper process, thereby aligning their work with the course' learning goals.

Table 1. The focus of each screencast

Week	Focus	Duration (min)
1	Test-driven Dev	10–15
2	Refactoring	5–10
2	Strategy pattern	5–10
3	Test stubs	5–10
3	State pattern	5–10
6	Design and Demo	5–10

The change in 2014 entailed replacing the requirement of handing in a written report with one or two screencasts for four of the six weekly deliveries, for a total of six screencasts of between 5–15 minutes duration. The focus of each screencast is outlined in Table 4. The current use is provided in Appendix B.

As an example of a reformulation of an exercise whose learning goal is developing code using the test driven development paradigm. Initially it was formulated as (2013):

> Write a report that includes
>
> 1. The final test list.
> 2. An outline of two or three *interesting* TDD iterations from your AlphaCiv development in detail, outlining the steps of the rhythm, testing principles used, and refactorings made.

which was initially rewritten into a screencast submission (2014)

> Create an approximately 10-15 minutes long screencast with audio narration of one to three *interesting* TDD iterations from your AlphaCiv development. The audio must refer to the TDD rhythm's steps and TDD principles used during development.

It should be noted that this formulation is problematic, as explained below. The current formulation as well as other examples are given in Section 8 and the appendix.

5. Student Evaluation

We asked students about their experience of producing screencasts through questionnaires containing both statements to evaluate on a Likert scale as well as a option to provide free text feedback. The initial 2014

course instance had an elaborate questionnaire of nine question, many of which were related to the internal group process of producing the screencast. Later course instances reduced questions to one or two questions regarding "work effort" and "suitability", as reported below. No evaluations were made in 2015 and 2016 due to testing other pedagogical aspects in the course, therefore the questionnaires these years had other foci. Generally, we strive to have as few questions as possible to avoid "questionnaire fatigue", leading to low response rates.

5.1. Question 1: Suitability

The statement for students to evaluate on the Likert scale deals with the approach's suitability for demonstrating students' skills, and thus addressing research question R1 in the introduction. It is formulated as:

> *Statement 1: Screencasting demonstrates our work better than written reports (Exercises on TDD, refactoring, clean code, etc.)*

The results for each year and student answers (Count) in percentages of answers in categories *Strong Agree (SA)* over *Agree (A)*, *Neutral (N)*, *Disagree (D)* to *Strong Disagree (SD)* are shown below.

Table 2. Statement 1

S1: Screencast demonstrates our work better...						
Year	**Count**	**SA**	**A**	**N**	**D**	**SD**
2014	63	41%	34%	12%	9%	3%
2017	72	16%	26%	31%	8%	15%
2018	79	34%	36%	17%	10%	3%
2019	113	29%	30%	20%	16%	5%
2020	86	26%	30%	34%	6%	4%

It appears the numbers for 2014 are a bit higher than later years, perhaps attributable to the *novelty* factor. Never-the-less student's perception is generally positive, with only 10%-23% finding that written report would be a better medium for submitting their work. That is, roughly more than 75% prefer or are neutral towards screencasts over the years. It should be noted, however, that the neutral group is generally quite large.

5.2. Question 2: Work effort

The perceived workload of producing screencasts compared to writing a report with similar contents, related to our research question R2, was evaluated through the statement:

> *Statement 2: Making screencasts requires less effort than writing a report on the same topic*

The distribution of answers were:

Table 3. Statement 2

S2: Requires less effort...						
Year	**Count**	**SA**	**A**	**N**	**D**	**SD**
2014	63	29%	27%	14%	13%	8%
2019	113	38%	30%	14%	8%	7%
2020	86	26%	33%	20%	13%	8%

Thus, between 56%–68% agree that screencasting is less effort to produce, while only 15%–21% disagree. The neutral group is notably smaller than for the S1 statement.

Question S2 was unfortunately not part of the evaluation 2017+2018. The author simply forgot to add the question to the pool, and only discovered it in 2019.

5.3. Free Text Answers

Both S1 and S2 allowed students to fill in free text comments, and many did so. As many answers are not strictly associated with the particular statement (S1 or S2), we present opinions and insights from the students comments to both below. We have read through all free text evaluations, and selected representative statements for similar concerns.

The benefits outlined below are directly copied from the evaluations, or the author's translation to English:

- "Good to illustrate work flow, and group interactions"
- "Easier to demonstrate execution"
- "They are just quicker to do"
- "It was far quicker to explain and communicate our work through screencasts"
- "They can be annoying at times, but are definitely to be preferred over the alternative"
- "It is OK, once you find out they do not have to be production quality movies"
- "Super nice format, once you have your setup running. Lovely to be able to demonstrate the process"
- "I have become better at explaining the material. Great preparation for the exam"[1]

And liabilities mentioned:

[1] The course has an oral exam.

- "Difficult to be certain if we covered all required aspects"
- "Highly work intensive. We had to do all exercises, make a story board of the screen cast, rehearse it, and potentially retake it"
- "I do not like it, I sound like a moron"
- "I'd rather just write a report; it is easier, quicker and any problem in it are easier to discuss with the TA"
- "Reports are easier to edit later. A screencast has to be redone"
- "In the beginning it takes longer time"

The feedback supports the Likert scale results and pinpoints some of the underlying issues that students experience.

First, it is a new way of submitting your work requiring a new technical setup and thus requires an investment in time up-front. It should be noted that informally we have not heard similar comments in the 2019–2020 course. We hypothesize that complete change to online teaching due to the COVID-19 pandemic has forced students into having the setup anyway. Some students initially feel quite uncomfortable with the format. In contrast, writing a report is well known, and perhaps also a bit more private.

Another issue is that of "fear of not being perfect" which some groups express. Since the revision in 2017, the markings of assignments have contributed towards the final course grade, and especially strong students feared "missing some points" leading to spending way too much effort on making the screencasts perfect. As discussed below, it is a concern that needs to be addressed.

And finally, there is the inherent property of screencasts that they are more difficult to overview, edit and restructure than reports—and also more difficult to pinpoint trouble spots in, for the TAs. As discussed below, this has implications on which types of assignments screencasts are suited for.

5.4. Question 3: Exam preparation

The course has an oral exam. While the focus is software engineering and programming, it is of course evident that the ability to present your work orally in a clear and concise way is important. While our data below is not directly tied to any of the research questions, R1–R3, we still find our findings interesting.

The original questionnaire (2014) had a single question regarding this aspect.

Statement 3: Screencasting prepares me better for the oral exam than writing reports.

The questionnaire data is:

Table 4. Statement 3

S3: Better preparation for oral exam...				
SA	**A**	**N**	**D**	**SD**
25%	41%	30%	2%	2%

We therefore find, that students perceive it as a good training, as also supported by comments in the free text sections over the years.

6. Teaching Assistant Evaluation

Research question R3 is about the workload of the teaching assistants. The 2014 course had five teaching assistants associated. One of the teaching assistants was also employed in the 2013 instance of the course which allowed her to discuss the change more precisely. Informal group interviews were conducted with them once a week as part of the regular reporting on student and course progress. Key points from their experience in reviewing and assessing the students' screencasts are:

1. Assessing screencasts are about as labour intensive as assessing written reports. Thus they fare no better nor worse in this respect.
2. Screencasts are more difficult to overview than a written report. As examples, the TAs reported it more difficult to check whether a screencast satisfactorily answers all questions and learning goals set forth in the exercise. If the TA fears a question is not really dealt with in the screencast he/she basically has to review it all again to be absolutely sure.
3. Providing written feedback is more cumbersome. For a written report you are used to referring using notes in PDF, or page number and position. For a screencast you need to refer to time, like (minute:second), and that forced the TAs to spend quite some time finding a particular frame again in the video.
4. Especially in the beginning, the TAs reported students were handing in unstructured, unfocused, and verbose screencasts. One particular video lasted 18 minutes ending in one student asking the others "Don't you think we should re-record this video?" to which his fellow students say: "No, we don't want to do all that work again."

5. Some screencasts were of low technical quality, like low volume of narration, low video quality, ambient and/or keyboard noise, etc.

Some of the issues can be remedied with a better reviewing process and their problems may be attributed to the TAs being faced with a new and relatively unknown medium. For instance, having a checklist of required answers to fill in while reviewing the screencast will remedy the problem in item [2]; noting time on a feedback sheet if a wrong or problematic answer is given in the screencast will lower issue [3].

One TA reported that you could actually speed-view the screencasts at twice the ordinary speed and still make out the narration, to lower the work on issues [2] and [3]. And the technical issues of low quality videos seems to be an issue of the past.

On the positive side, the TAs also reported that the screencasts does provide a relatively better overview and understanding of the design and implementation so the actual time they have to spend reviewing the source code is lowered compared to relying only on the students' written reports.

As part of course planning we let the TAs choose whether they preferred a written report or screencast delivery for the design exercise in week 6. They chose a screencast, as is evident from Table 1.

In 2017 a semi-structured open-ended interview [16] was held with the TAs using an interview guide based on four areas of interest: A) benefits, B), liabilities, C) TA's review process, D) how to provide feedback. Generally, this interview supports the main conclusions of the informal discussions above.

Regarding benefits, the TAs generally points to an aspect also identified by the students, namely that it provides insights into the process as it unfolds. They judge "the best screencasts" are those in which the members of the group talks together about what they do, as opposed to a single presenter. They highlight the inability of students "to cheat" and "we can catch those groups that simply do not get it". Some of the later assignments require students to demonstrate their final Broker based client-server systems, and the TAs agreed that it is easier to view their screencast because "starting up a server and four clients takes one h. . . of a time." They pointed out that depending on the concrete assignment and the skills of the group, they could sometimes get away with only reviewing the screencast and avoid having to get the full source code and execute it to validate it. In particular, strongly skilled groups may cover all learning goals in their screencast to a satisfactory degree.

Regarding liabilities there were also overlap with students' experiences, like some "spend too much time with numerous retakes", and that some group members never do the talking as they experience "stage fright". However, much of the discussion of liabilities actually stems from the reviewing process.

The inherent property of screencasts as a streaming medium makes it cumbersome to overview, navigate, and comment upon. For instance, ensuring that the set of required aspects have been covered is difficult, "Did they mention the "One Level of Abstraction" clean code issue", which sometimes forces the TA to rewind the screencast or even view it one more time. Especially submissions made by below-average students are hard to mark as it is difficult to "assess that they mention all learning aspects". Some aspects are also difficult to discern from the screencast—one example is the TDD rhythm that requires all tests to be run at certain time in the process. In IntelliJ this is a short-cut key and executing the tests is so fast it is nearly not visible on the video.

Fortunately, TAs have evolved better processes and mention techniques like "viewing the screencast at 1½ or 2 times normal speed" in which you can still get the main aspects of the spoken word; using two (or more) physical monitors—one with the video and the other with an open feedback document for marking exact time and noting issues or noting that a required issue has been covered; and watching the screencast "stop/go" with a checklist of learning goals to look out for. Still, some TAs report "video feed fatigue" and concentration issues, noting that they "zone out" and have to review a given screencast again. Generally, the TAs provide feedback in writing, using overall comments and also (minute:second) marks for specific comments or suggestions.

7. Instructors Experience

Screencasts as submissions was also employed in two instances of a quarter length courses in cloud computing. The course has a strong programming and tool-stack focus with learning goals set on microservices, virtualization and container technology (Docker), stability patterns, NoSQL (MongoDB), messaging (RabbitMQ), replication and load balancing; again using a project as main learning vehicle [17]. Here screencasts were required for demonstration focussed assignments, like demonstrating a properly set up MongoDB replica set or sharding cluster (the latter requires at least seven correctly configured servers); or demonstrating fail-over handling.

As a smaller course, the author fulfilled the role of TA in the course, assessing the student groups' submissions.

Generally our experiences align with those already reported. As an example, downloading student's code base and trying to replicate their written documentation of how they started seven servers and configured them, and finally made their application use it, does not compare favorably to just reviewing a 6 minutes long screencast in which they talk you through the process while their system executes and is responding to the execution scenario detailed in the assignment. Furthermore, it is impossible (or would require enormous effort) to produce a screencast, if they have not actually solved the assignment correctly.

Again, formulating the exact requirements is essential. An example below is an assignment in which the system must use a session-database pattern as cache, shows the scenario/user story that the assignment details along with the required system output allowing assessment.

> Submit a screencast (or a link to a youtube or other video) that is about 5 minutes long screencast containing:
>
> - A shell that show 'docker ps' output in which there are containers running for MongoDB, RabbitMQ, Memcached, and at least TWO servers connected to the MQ.
> - A shell with the client running where you MOVE through three rooms and then 'BACK' your way.
> - Shell(s) showing that the TWO servers alternate to retrieve messages from the MQ.
> - The RabbitMQ dashboard in which the two connections to the two servers are shown, AND also show the message rate indicating that messages flowed from the client
> - A TELNET connection to the memcached in which a GET command show that an object is stored associated with the playerID.

Similar assessment techniques as outline above were employed expect we never got used to listening to "mouse voices" when running videos as twice the speed and thus avoided it.

8. Best Practices

Based upon years of using screencasting in teaching a set of best practices or guide lines emerge.

- **Use screencasts for process, overview, and demonstration assignments.** Initially, screencasts were used for many types of assignments, but they should be used primarily for process oriented assignments, assignments to provide overview over how a specific feature is implemented across a large code base, or assignments in which demonstration is central, due to the issues the TAs experience. In our course, these are assignments on test driven development, refactoring and cleaning up code, introducing a specific design patterns, and demonstrations of GUI and distribution.

- **Use screencasts to enforce process adherence.** Screencasts excel as they unfold over time what students actually do, or what their software systems actually do. If your code does not compile or hangs while executing, you can still easily fool a stressed TA with some screenshots (before the system crashed) and code fragments in a written report. If you did your test-first assignment by writing production code and adding tests after-the-fact, no one will notice in a written report. Needless to say, this is not possible in a screencast.

- **Be clear, that it is not a "Hollywood movie."** Especially ambitious students have a fear of falling short in producing the screen cast. The abundance of screencasts by professional teachers or presenters misleads students to think that they have to produce screencasts of similar quality. As shown above and often heard, these students complain of high workloads because they produce story boards, rehearse it several times, and retakes it endlessly to ensure every bit is as polished as possible. We have countered this with success by being very clear to TAs that they should be lenient in grading and very clear to students that they should make a draft of what to say, and then do the screencast, and only do a single retake if it goes wrong. We are actively communicating these points in our lectures before the first submission is due. As one of the comments were: "It is OK, once you find out they do not have to be production quality movies"

- **Provide a template for the screencast.** A report has a accepted structure, and a screencast must have one as well. The initial requirements of the assignments was "Make a screencast" in the anticipation that students had seen enough to be able to figure out how to organize it. It turned out that this was not true. Therefore, each assignment today provides a template of how to section the screencast, and what to emphasize in central sections. The current wording on the example assignment from Section 4 is the following:

> Create an approximately 6-12 minutes long screencast with audio narration, outlining one or a few *interesting* TDD iteration(s) from your AlphaCiv development. The screencast must be structured with
>
> 1. Intro: State your names and group name
> 2. Shortly outline what you achieved in the last TDD iteration of AlphaCiv, briefly

showing the test list items and associated code.
3. State the purpose of this iteration
4. *Do* the iteration, and follow and refer to TDD principles applied and the five steps of the rhythm as you go. *(Say out loud: "Now we quickly add a test", add the test and explain what it does as you code it - if you "fake it" then say that, if you "triangulate" then say that, etc.; mention that you now "run test and see it fail", etc.)*
5. Conclude

For other examples, refer to the appendix.

- **Use TAs best practices.** TAs develop ways to improve the assessment process. These practices are important to bring on to new TAs, like "playing at 1½ or 2 times normal speed", "use multiple screens with screencast on one screen and IDE on code base plus grade sheet/commenting on the other", "stop video and comment immediately", "use (minute:second) marks to identify comment points", etc.

9. Conclusion

We set out to assess our three research questions regarding the use of screencasting: They better support assessing learning goals (R1), they are more cost efficient than written reports (R2), and they are more efficient to assess for the teaching assistants (R3). In conclusion we find support for R1 (for certain types of assignments), for R2, but less support for R3.

Our data indicates that student produced screencasts are a viable and relevant alternative to written reports as submission medium for specific types of assignments. These types include *process oriented assignments* in which students must demonstrate their ability to execute development processes, like test driven development, refactoring, or cleaning up code; *execution oriented assignments* in which students must demonstrate that their (complex) system indeed operate correctly and efficiently; and *overview oriented assignments* in which students provide overview of their code base by explaining specific elements while navigating the code structure. Used correctly, screencasts may lower the work effort required by students and highlight the process aspect

While students are generally favorable to screencasts, the teaching assistants are less so, as there seems to be little benefits to their process of assessment and marking. In particular, the inability to quickly overview a screencast to assess where all learning goals have been addressed is a major issue. Best practices are therefore important to help the teaching assistants, including running screencasts at 1½ or 2 times speed, having multiple screens, and doing commenting by "stop/go" the video while watching.

Finally, matching of expectations is important to convey clearly to both students as well as teaching assistants. The WWW has an abundance of high quality and professionally made screencasts, which may lead students to believe that this is the required level. It must be actively communicated what is expected before the first submission is due.

References

[1] M. E. Caspersen and J. Bennedsen, "Instructional design of a programming course: a learning theoretic approach," in *Proceedings of the third international workshop on Computing education research*, pp. 111–122, 2007.

[2] V. Berardi and G. E. Blundell, "A learning theory conceptual foundation for using capture technology in teaching," *Information Systems Education Journal*, vol. 12, no. 2, p. 64, 2014.

[3] J. O'Flaherty and C. Phillips, "The use of flipped classrooms in higher education: A scoping review," *The internet and higher education*, vol. 25, pp. 85–95, 2015.

[4] D. Youngberg, "Why online education won't replace college—yet," Aug. 2012. https://www.chronicle.com/article/why-online-education-wont-replace-college-yet/.

[5] P. J. Deneen, "We're all to blame for moocs," *The Chronicle of Higher Education*, no. Jun 3, 2013.

[6] L. M. Powell *et al.*, "Evaluating the effectiveness of self-created student screencasts as a tool to increase student learning outcomes in a hands-on computer programming course," *Information Systems Education Journal*, vol. 13, no. 5, p. 106, 2015.

[7] K. Shafer, "The proof is in the screencast," *Contemporary issues in technology and teacher education*, vol. 10, no. 4, pp. 383–410, 2010.

[8] S. Mohorovičić, "Creation and use of screencasts in higher education," in *2012 Proceedings of the 35th International Convention MIPRO*, pp. 1293–1298, IEEE, 2012.

[9] H. B. Christensen, "A story-telling approach for a software engineering course design," in *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '09, (New York, NY, USA), pp. 60–64, ACM, 2009.

[10] H. B. Christensen, *Flexible, Reliable Software—Using Patterns and Agile Development*. CRC Press, 2010.

[11] K. Beck, *Test-Driven Development by Example*. Addison-Wesley Signature Series, 2003.

[12] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[13] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson, 2008.

[14] H. B. Christensen, *Flexible, Reliable, Distributed Software—Still Using Patterns and Agile Development*. leanpub.com, 2020.

[15] K. Buffardi and S. H. Edwards, "Exploring influences on student adherence to test-driven development," in *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pp. 105–110, 2012.

[16] M. Q. Patton, *Qualitative Research & Evaluation Methods*. Sage Publications, Inc., 2002.

[17] H. B. Christensen, "Teaching devops and cloud computing using a cognitive apprenticeship and story-telling approach," in *Proceedings of the 2016 ACM conference on innovation and technology in computer science education*, pp. 174–179, 2016.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reuseable Object-Oriented Software*. Addison-Wesley, 1995.

A. Assignment Formulation Examples

A.1. Refactoring Example

This assignment has *clean code* [13] as learning goal:

> Create one approximately 5-8 minutes long screencasts with audio narration that demonstrates a method clean up to achieve one or several clean code properties. The screencast must be structured with
>
> - Intro: State your names and group name
> - Short explanation of what clean code property that your method does NOT have, and outline what refactoring you will do to achieve it.
> - *Do* the refactoring, explaining what you do as you go along.
> - Conclude and argue how the code has become more clean.
>
> (Note: it is not a screen cast of the refactoring of the full method; it is like one of my screencasts in which I focus on one or a couple of clean code property and handle that.)

A.2. Overview Example

This assignment has variant handling through *compositional design* and *dependency injection* [18, 10] as learning goal:

> Create one 5-8 minute screencast that details the architecture and implementation of one of the required variants, specifically, you should screencast/show the code and explain it. Be sure to show and explain a) the interface(s) introduced, b) the variability point(s), c) the place(s) where variable behavior is selected ("this code chooses whether it becomes an AlphaCiv or GammaCiv game"), and d) the implementing delegates. Remember to argue why your code is a compositional design, and relate to the 3-1-2 process.

B. Current Use of Screencasting

Since 2018 screencasted submissions have been on these areas.

Week	**Focus**	**Duration (min)**
2018–2020		
1	TDD I	6–12
2	TDD II Git	6–12
3	Refactoring	5–8
3	Strategy pattern	5–8
4	Clean code	5–8
8	Frameworks	6–12
10	Broker	4–8