Agile Development Spikes Applied to Computer Science Education

Clinton J Woodward[#], James Montgomery[*], Rajesh Vasa[#] and Andrew Cain[#]

[#]Swinburne University of Technology,
Faculty of Information and Communication Technologies
Hawthorn, Victoria, Australia
Email: {cwoodward,rvasa,acain}@swin.edu.au

[*]Research School of Computer Science
Australian National University
Canberra, Australia
Email: james.montgomery@anu.edu.au

***Abstract*—Spikes are an agile software development technique used by software teams to investigate, close gaps and reduce risk. Computer science education can benefit from the application of agile techniques. In this paper we document our definition of spikes, adapted from agile software development practice, applied to computer science education across a number of different units. Our view is that spikes align well with many educational objectives. We also reflect on our educational experiences to present guidance on how and why spikes might be applied, including specific benefits, limitations and drawbacks.**

Keywords—computer science; education; spikes; agile software development

I. Introduction

Agile software development is a loose collection of software engineering methodologies, the foundations of which were first outlined in 2001 [1], which have transformed the software development industry [2]. A central characteristic of agile approaches is iterative development through cycles of planning, implementation and testing [3].

One agile technique, used to structure small technical investigations or research, is known as a spike or spike solution [4]. The key objective of a spike is to close a gap that a development team has, in a way that can be shared among a development team; if successful, the gap becomes closed for the entire team through effective communication. Clearly, the behaviour involved in a spike predates agile software development practice, and indeed nothing in its description limits it to software development. Across a diverse range of fields, if a practitioner does not know something or how to do something, they can perform a small investigation to determine how to proceed.

Key objectives within computer science education are to help students construct meaningful knowledge and relevant skills [5]. For some students, the educational environment and expectations can appear too hard or trivial, and both can result in disengagement. The technical nature of the field requires incremental skill development, and hours of practical experience to attain professional levels. Strong constructive alignment and portfolio based assessment approaches have been developed and utilised with success, by some of the authors of this paper, for introductory programming units [6]. Communication is also vital as a way for students to demonstrate and gain knowledge.

In this paper we document our definition of spikes, adapted from agile software development practice, and applied to computer science education across a number of different units. Our view is that spikes align well with many educational objectives. We also reflect on our educational experiences to present guidance on how and why spikes might be applied, including specific benefits, limitations and drawbacks. It is the intent of this work to enable other educators, from various fields, to benefit from the technique.

II. Approach and Development

A. Spikes in Industry

Typically, each new software development project will have novel elements, perhaps in the application domain or in the skills required to complete it. These novel elements create risks for the timely and successful completion of the project. These risks can be considered twofold: a development team cannot provide an accurate quote (nor allocate team resources) if it does not know how some part of a project needs to be done; and a project will likely fail, or be significantly delayed, if a knowledge or skill gap is only recognised as it occurs. In agile, the practice of spiking is thus a risk minimisation activity, where gaps in knowledge, skill and technology can be identified and dealt with at an early stage.

In practice, at each stage of a project the development team can spike out the unknown aspects and determine what steps to take to cover the associated gaps. Each spike may involve the production of a working piece of software for the specific issue being addressed: the spike metaphor thus represents an ‘end-to-end’ or complete solution that is also very specific. Any software produced in this way is then discarded and not intended for use in the final product, in a manner similar to an artist drawing preliminary sketches which are discarded before creating a final work with a “fresh” canvas.

Despite the widespread use of spiking in industry, the approach has been only vaguely or weakly defined, with little structure to guide developers and an absence of convergence in approaches taken by different development teams. Spikes are also largely absent from software engineering literature.

In order to create a more structured and rigorous approach to spike for industry use, one of the authors of this paper defined the KoST model, which categorises gaps into three groups:

- **Knowledge** gaps represent what a person does not know about the domain for which they are developing.
- **Skill** gaps exist where a person knows what is required and broadly how to do it, but requires more practice or experience. A skill gap prevents accurate estimates of how long a task will take.
- **Technology** gaps address the question of whether, given a budget and timeline, there exists the technology to assemble the solution required.

Our experience suggests that the 'knowledge' category can actually be viewed as relating to both knowledge and skill of the domain, while the 'skill' category relates specifically to technical (but general) knowledge and skill.

Educationally, knowledge and skills are more relevant or important than technology, although both students and software developers in industry are typically mindful of new technology and its impact.

The industry-based approach to using spikes follows these steps:

1) Create a plan for the next phase of the software project
2) Identify the gaps in knowledge, skills and technology
3) Devise spikes to cover those gaps (i.e. determine what activities need to be performed)
4) Execute the spike over a short period of time (i.e. two days)
5) Rework the plan in light of what was learned
6) Perform the phase development work (i.e. two to three weeks) followed by a review to inform the next phase

This specific approach to spikes for industry use has been taught to both students and companies by one of the authors. Several companies have managed to dramatically reduce their risk profile as a result, validating the approach.

B. Spikes for Computer Science Education

We have adapted and applied spikes to several different university level computer science units, over a number of iterations. Through a deliberate reflective practice by the staff involved, observations and reflections on the use of spikes have been captured and changes made to improve the academic application in different context.

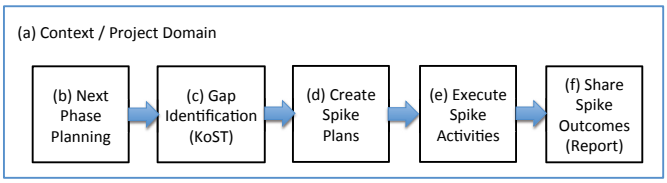

Fig. 1. Steps of an industry-based spike approach

Fig. 1 shows a representation of the spikes stages based on the industry model described in the Section II-A.

The overall context or domain (a) for a spike in industry is the product being developed for a client. In an education context this becomes either a domain selected by staff, or by students when appropriate, constrained within the objectives of the unit of study. For example, staff may create a product specification which students are expected to develop. Similarly staff may specify a domain type, such as a web application with particular features, in which students are able to select content related requirements.

Using a phase-based agile development model, such as sprints in Scrum methodology [2], developers plan their next iteration (b) of activity. The objective of this planning is to select features to implement in the next time-boxed amount of activity, which should result in working software. For this to happen risks should be identified and reduced in a practical manner. So, from the list and details of planned activity, KoST gaps can be identified (c). Note that the gaps are very specific to the teams' unique experience and skill; another development team given the same tasks would be expected to identify very different gaps.

Placed within the educational context, the specific details or features needed for the next iteration (b) can be specified by staff (as a way of allowing students to focus on later spike steps), or developed by students (allowing them to demonstrate their understanding). Similarly gap identification can be performed by staff or students, each creating different learning opportunities that can be selected based on curriculum needs.

Once gaps are identified, a set of Spike Plans (d) can be created to address each. We have created a Spike Plan template with headings and descriptions shown in Fig. 2. In practice, teaching staff have varied the descriptions and given additional example text to help students understand what is needed, especially if they are creating their own spike plans for the first time.

Name:
A meaningful and unique name for this spike plan
Context:
Outline the reason and context for the spike
Knowledge Gap:
Skill Gap:
Technology Gap:
State one or more specific gaps to be addressed
Goals/Deliverables:
List the specific goals and deliverables of this spike.
Describe a tangible and minimal end-to-end solution.
Start date:
Target date:
Good dates are needed for a controlled process.
Planning notes:
Outline a proposed plan of how this spike can be undertaken.
Capture as much team knowledge here as possible to make the work as effective as possible.

Fig. 2. Spike Plan Template

Spike plans can now be executed (e) in any order that suits. In an educational environment, students often attempt spikes in the order presented, particularly if they lack a deep understanding of the domain. However in the context of industry usage spikes can be done in any order that suits the team member preferences or dependencies. It is possible that spikes may be clustered or hierarchically dependent.

Students can borrow good associated practice of industry with respect to task allocation and source code management. Even for an independent student working with spikes it is useful that the execution of a spike plan, and its initial definition

and final outcomes, be documented and stored in a deliberate and careful manner.

Once the goals of the spike plan have been achieved, and any specific artefacts created, the results are shared with the team (f). In this way, not only is the risk associated with the original gap closed, but also the risk that only one person in the team knows it; results must be in a form that the entire team can take advantage of. Placed in a specific education context, the outcomes should be in a format that would close the same gap in another student of a similar background.

In the process defined and used here, a specific Spike Outcome Report Template has been developed and provided to students, shown in Fig. 3. Although not strictly needed, and containing repeated details from the spike plan, there is a value in having a single outcome document that captures not only the expected results but also any new issues or recommendations. There is also the value of productivity-by-convention where having a standard format can make the process of creating outcome reports, and understanding them, quicker and easier.

Name:
Should match the original spike plan name
Goals:
Restate the original spike plan goals (outcomes)
Personnel:
State who the contact person is and any backup.
Technologies, Tools and Resources Used:
Provide a specific list of useful resources for other team members to learn from this spike. Specific books, software (versions) and URLs are common.
Tasks Undertaken:
List key tasks likely to be needed to help another developer
What We Found Out:
Concise evidence: graphs, screen shots, list of outcomes, data, notes.
Open Issues / Risk (optional):
List any issues and risks that were not resolved. This may include a new range of risks previously known.
Recommendations (optional):
This might include suggesting additional spikes are needed, or that the gap has been completely solved and the team should move on.

Fig. 3. Spike Outcome Template

In industry practice, once spike outcomes have been captured and communicated to the team, the results can be used to inform or reform the planned development activities. For students, the spike outcomes are often skill based, and so the new skills can be applied to new domains and the activities of constructing new knowledge. This deliberate step of reflection is key to both good agile practice and deep learning outcomes for students.

The authors of this paper have utilised spikes in a number of different curriculum approaches, including some cases where spikes were a replacement for traditional assignments. To support this the following criteria were developed for assessment purposes. However they also provide a useful checklist for students to validate their own activities.

- Has the spike context and problem been clearly identified?
- Have the risks (symptoms as well as possible causes) been identified?
- Does the spike plan focus on the risks properly?
- Have the outcomes been captured at the end of the spike?
- Did the spike achieve its intended result? i.e. risk minimisation?
- Has the knowledge gained in the spike been passed on to other team members?

Although this list of criteria can be used as part of a weighted grading scheme, the stronger value, in the current opinion of the authors, is for spikes to be assessed qualitatively.

III. Application and Reflection

A. Agile Development Project

Spikes for computer science education were first used by one of the authors as part of a single semester (12 week) agile development project unit. The student cohort had strong technical ability and maturity, and had performed well in other technical units. A number of the students were also working in industry at the time.

To begin, staff predefined a number of topics and students required to identify gaps, develop spike plans and execute them. A short time of two days was set for the execution stage, which was appropriate given the narrow focus of spikes. The short time frame also restricted students from expanding the scope of the work. The final reporting stage initially consisted of simple documentation and demonstration by students to staff. This served the purpose of both demonstrating the students' achievements and providing an opportunity for formative feedback from staff.

After the initial predefined topics, students designed and undertook 10-12 spikes as a group on topics determined by their project requirements. The initial four spikes outcomes contributed to the final student results, and were graded using the criteria presented earlier.

The primary reason spikes were included in the unit was for their important role in agile development in industry. A key observation was that students tended to focus on skill gaps, while industry largely focus on domain knowledge gaps. This is to be expected as student are often yet to acquire the skills they will use in industry, while those in industry are applying those skills to new possibly unknown domains.

In the Agile Development Project unit the essential features of spiking were:

- practical experience with a risk minimisation process;
- a structure for solving small, specific problems;
- enabling a transition for students from abstract concepts to concrete experience; and
- a process that students could drive to identify gaps and how they could be covered.

B. Database Programming

The second application of spikes was within a second year programming unit where students developed software to interact with databases. Again, the cohort of students was from a

strong software development course. For each of the functional concepts covered, two competing technologies were used by students and compared.

Given that there were two distinct but related technologies that students had to master, each spike was actually presented as a pair of spikes, one for each technology. Students were assigned in pairs and each student responsible for one of the technologies. The intention was that the outcome reports from each spike could be shared with the team member.

Over the 12-week semester there were nine pairs of spikes. Although the spikes were intended to be formative assessment, in effect they represented a substantive portion of the students final grade. Each spike outcome was marked by a tutor who would (a) ask the student to demonstrate and explain what they had learnt or achieved and (b) assign a mark out of 5. As understanding is valued more highly than completing all spike-related tasks, marks were biased to reflect this.

In this application, students were given spike plans; gaps in skills and knowledge were predefined by teaching staff, along with the steps that should be taken to cover each gap. For this unit, the alternative of providing open plans for students to develop and define was not considered.

Spikes were used in this unit because they were relevant to the topic and industry practice, and it was thought spikes would encourage students to engage in the subject material more deeply. Another perceived advantage was that, when used as small specific assignments, spikes do not require students to draw on a wide range of knowledge. Instead they can focus on one concept without the overhead of a large assignment and develop skills related to one topic at a time.

The essential features of spikes in the Database Programming unit were:

- spikes can be used as a structure for solving small specific problems;
- students were given complete spike plans and did not have the freedom or responsibility to identify gaps or the main steps that should be taken to cover them;
- students were given the freedom to choose a domain in which to perform each spike; although many needed encouragement to choose something novel and interesting; and
- spikes were effectively a form of summative assessment, even thought the intent was as a formative activity.

C. *Games Programming and AI for Games*

Games Programming was an existing third-year technical unit, redeveloped to simplify the curriculum and refocus on practical game-related applications of data structures and patterns. The unit structure was moved to an entirely criterion-referenced, portfolio-based assessment scheme (in a manner similar to [6]), which allowed students to develop a portfolio of varied work in an environment more similar to that which they will experience in the workforce.

A number of simple game design documents were created as the context for spike work. Accompanying each design was a set of spikes designed to cover the necessary knowledge and skills to produce the game in question. In the later iterations of the unit there were 13 core spikes. The details provided to students in the planning notes of the spike plans were quite detailed for the early spikes to help students. This level of detail (help) was reduce and removed for the later spikes plans, so that students would take on greater responsibility in completing the spikes.

Spike plans often-included additional “Extension” ideas. For any extension work students were encouraged to develop their own spike plans, in which they would take full responsibility for identifying gaps and developing the plan.

Unlike database programming there was a reduced flexibility in the domain, although students have more flexibility in the distinction project. All spikes had the relevant gaps (and key deliverable outcomes) specified to ensure that student did not stray too widely from the core content; the remainder of the portfolio allows students to examine other topics.

In the assessment, tutors were expected to be strict and maintain a high standard, but also to offer constructive feedback and allow students to be reassessed multiple times. Assessment was based mostly on the report that students needed to print. Tutors often requested students to demonstrate a working implementation as well. Although setting high-standards and expectations of students can be daunting for them, the core work was essential and basic, which all passing students would master. The majority of students were proud of their achievement.

The curriculum of the AI for Games unit contained many design and concept elements, with less emphasis on technical development skills. The core of the unit involved a mixture of lab and six spikes. A strong constructive alignment and portfolio-based assessment approach was again used. The combination of portfolio assessment with lab and spike work was generally quite effective.

As used in Games Programming and AI for Games, the essential features of spikes are that they:

- are a structure for solving small, specific problems;
- are assessed as pass/fail, but with the requirement that core spikes must be passed to pass the unit;
- may be (frequently are) resubmitted in response to staff feedback;
- have decreasing guidance (plan details) through the semester;
- have ‘extensions’ where students are encouraged to generate entire plans themselves;
- have limited freedom for students to choose the domain of spikes; and
- are presented in the context of portfolio-based assessment.

IV. DISCUSSION

A. *Problems and Limitations*

From our experiences applying spikes to a number of units we encountered the following issues and limitations in various forms:

- Students need clear definitions and expectations to perform the process well.
- If a sensible time window is not set, students will tend to exceed the required outcomes.
- Students find it hard to create their own spike plans initially, but this improves with experience and guidance.
- Students tend to focus mainly on skill gaps, while industry gaps are typically knowledge related.
- Student need feedback in order to understand the purpose, audience and level of detail needed in outcome reports.
- The more staff constrained the spike process, the more it can appear as simply another form of assignments to students, which should be avoided.
- Summative grading or assessment of spike work drastically, if not completely, reduces the formative and deep learning value.
- All teaching staff involved need to fully understand and support the purpose of spikes in order for students to adopt and apply them effectively.

One issue for incoming students is that spikes are relatively unstructured compared to standard assignments with which most students are more familiar. In the first application the lack of strong definition and templates made it difficult to assess if students had done the work and to provide feedback. Tutors note that typically after two spikes have been approved, students generally appear more comfortable with the approach. Presumably this is because the feedback they receive from tutors is sufficient to provide the structure that is initially missing.

Some students complain that spikes are 'too much work'. We speculate that those students who consider the workload too onerous may be those with weak prerequisite skills, although many such students ultimately achieve good learning outcomes.

If there are no weekly deadlines for spikes, students do have a tendency to delay their completion until too late in the semester. In some cases students have submitted reports early, received feedback and merely delayed submitting the reviewed reports. In such a self-directed environment, there is typically much variation in progress among students. Note that this does not indicate that deadlines through the semester are necessarily required only that some mechanism to encourage more timely completion is needed..

B. Benefits

We have observed a number of benefits for both staff and more importantly students in the use of spikes:

- Spikes are grounded in an industry practice, making it relevant to students and easier for them to engage with the approach.
- Students learn a process of what to do in order to learn.
- Spiking is relatively simple process, making it easy to learn, with enough structure to enable productivity by convention.
- As spikes are a structured approach to learning, which enables life-long learning.
- By having narrow and well-defined activities, spikes help to avoid scope creep and misdirected effort.
- When students identify and define their own spikes and plans, they gain experience in valuable problem decomposition skills.
- Spikes support different solutions to problems, rewarding creative and unique approaches by students.
- When spike outcome standards are set high, students have a strong sense of achievement and are often very proud of their work.
- Plagiarism issues are minimised by the encouragement of unique and individual work, and the requirement for direct face-to-face communication and demonstration of their work.

Spikes are an analytical technique to solve problems, and this allows students to translate problems that are abstract into concrete solutions. A student may start by thinking that they do not know what to do, but in reality they often have some idea of what they want to achieve and how they might do it, which can be made clearer through the spiking process.

Our view is that spikes can promote better engagement and deep learning. It is certainly effective as a method for checking that minimum standards have been reached, and from the staff perspective, it is also good for identifying trends in class.

Students have voluntarily commented that they thoroughly enjoy the spike approach, and would like to see it adopted in other units instead of traditional lab work or assignment work.

In later units (particularly self-direct project units), as well as industry employment, students have continued to use all or some of the spiking process. For these reasons we contend that spikes remain relevant and help with life-long learning.

C. Recommendations

Based on our collective experience we suggest the following recommendations regarding spike use in computer science education:

- Allocate time to clearly communicating the purpose, format and structure of spikes. Repeat as needed. However, experience is the only effective way students become proficient with the process.
- The self-directed nature of spikes works best for mature and independent learners.
- The spike process works well for technical units. It is unclear at this stage how it could or should be applied to other educational domains.
- Sharing spike outcomes within the peer group of students is difficult.
- If using spikes as a required core feature of a unit, target low-level learning outcomes.
- Use a formative assessment process during a semester to avoid reassessment of spike work later.
- Utilise face-to-face interviews for outcome reports and assessment. This maximises the formative opportunity, encourages good standards and avoids plagiarism issues.

As noted earlier, many students struggle with independent time management when confronted for the first time with the

"unstructured" nature of spikes. This is compounded when a unit uses a highly formative process without "marks" as inventive for timely completion of work. As part of our work to address time management issues, we have also used the agile practice of scrum "burn down" charts [2]. We created an online burn down chart tool as way for students to be aware of, and to better manage, tasks including spike work. The tool itself has been reported in [7], and some analysis of student progress (reported as burn down charts) in [8].

We suggest that spikes are not appropriate for first year students. At such an early stage the overhead of a compound skill such as software development process (the context in which spikes exist) outweighs the possible benefits.

At one point staff trialled having each student use another's outcome report to see if they could follow it to achieve the same outcome, but this was found to be ineffective for both authors and readers. This suggested that spikes should be used differently.

D. *Spikes and Portfolio Assessment*

In promoting the use of portfolio-based assessment, Biggs and Tang [9](p82) state that "[m]uch learning at tertiary level, particularly in professional courses, is about getting students to behave differently in the sense of making theoretically informed decisions; it is not so much about declaring who-said-what and who-did-that." In the context of computer science, Plimmer [10] suggests that skills such as programming involve the distinct aspects of programming language knowledge, the ability to break problems into smaller parts, and creativity, which portfolio-based assessment allows students to develop and demonstrate better than traditional assessment approaches. Others, such as Jones [11], consider portfolio-based assessment as a mechanism to promote student engagement by giving students greater responsibility for their own work.

Whether they are the subject of summative assessment, as in Database Programming, or forming a body of required formative assessment, as in AI for Games and Games Programming, spike outcome reports implicitly represent a portfolio of work. Indeed, one of the motivations for using spikes has been to encourage student engagement with subject material. Further, even though staff in these units determined the topic of each spike, students are given some degree of autonomy in being able to select the domain in which to apply the relevant skills and knowledge and in pursuing extensions to the work through the development of new spikes for which they are entirely responsible. As Ramsden [12] notes, students' perceptions of greater autonomy in a course are associated with deep approaches to learning.

Biggs and Tang [9] characterise portfolio tasks as assessing either declarative knowledge (understanding of course content) or functioning knowledge (the application of that knowledge), and as relating to either a specific topic or the whole unit. Spikes as they have been used mostly relate to functioning knowledge about a specific topic, although there is nothing implicit in the activity that prevents it from being applied to declarative knowledge.

V. CONCLUSION

Spikes appear to represent a distinct model for problem solving that either produces a de facto portfolio or may be incorporated into a wider assessment portfolio. Some of the distinctive features of the approach, including the underlying KoST model and accompanying structured plan document, as well as the narrowness of the problem each spike addresses, may appear to restrict its applicability to technical subjects in the software development area. Certainly the key concept of an 'end-to-end' solution seems tightly bound to subject areas where students produce something that is recognisably complete even if it does not solve an entire problem. The question is how broadly a narrowly defined learning goal can produce a 'complete' product.

If an area of study can be broken down into highly specific and relatively isolated topics then the spike approach may be suitable, even if it is outside the software development domain. Gaps in knowledge and skill exist in all educational domains. Establishing a plan of actions to address the gaps identified is also universally applicable, as is writing a brief report describing what was found. Future work can investigate these possibilities by taking the characteristics of spikes described above and applying them to curricula in new fields.

REFERENCES

[1] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries et al., "The agile manifesto," http://www.agilemanifesto.org/principles.html, 2001.

[2] J. Sutherland and K. Schwaber, The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process. Citeseer, 2007. [Online]. Available: http://scrumfoundation.com/library

[3] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, Reading, Ma, 2000.

[4] J. Shore and S. Warden, The Art of Agile Development. O'Reily, 2008.

[5] M. Ben-Ari, "Constructivism in computer science education," in Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education. New York, NY, USA: ACM, 1998, pp. 257–261. [Online]. Available: http://doi.acm.org/10.1145/273133. 274308

[6] A. Cain and C. J. Woodward, "Toward constructive alignment with portfolio assessment for introductory programming," in Proceedings of the first IEEE International Conference on Teaching, Assessment and Learning for Engineering. IEEE, 2012, pp. 345–350.

[7] C. J. Woodward, A. Cain, S. Pace, A. Jones, and J. Funke Kupper, "Helping students track learning progress using burn down charts," in Proceedings of the 2nd IEEE International Conference on Teaching, Assessment and Learning for Engineering. IEEE, 2013, in press.

[8] A. Cain, C. J. Woodward and S. Pace, "Examining student progress in portfolio assessed introductory programming" in Proceedings of the 2nd IEEE International Conference on Teaching, Assessment and Learning for Engineering. IEEE, 2013, in press.

[9] J. Biggs and C. Tang, "Assessment by portfolio: Constructing learning and designing teaching," Research and Development in Higher Education, pp. 79–87, 1997.

[10] B. Plimmer, "A case study of portfolio assessment in a computer programming course," in Proceedings of the 13th Annual Conference of the National Advisory Committee on Computing Qualifications, 2000.

[11] M. Jones, "The redesign of the delivery of an introductory programming unit." 2007. [Online]. Available: www.ics.heacademy.ac.uk

[12] P. Ramsden, Learning to Teach in Higher Education. Psychology Press, 1992.